Seamless Sequence of Software Defined Radio Designs through Hardware Reconfigurability of FPGAs

Amir Hossein Gholamipour Elaheh Bozorgzadeh Lichun Bao
Donald Bren School of Information and Computer Science
University of California, Irvine
E-mail :{amirgh, eli, lbao}@ics.uci.edu

Abstract- **Software Defined Radio (SDR) base stations can compensate for failures in disaster scenarios by assimilating different communication technologies. FPGAs play an important role in the platform of an SDR base station because of flexibility and DSP processing power that they deliver. The flexibility of FPGAs comes at the high cost of reconfiguration time overhead which can be a serious deterrence because of QoS requirements of real time traffic. In this paper we propose a solution to reduce reconfiguration time overhead at system-level where we are provided the configuration of each wireless system. Following that we step further and integrate our solution in to a floorplanner to generate placements for wireless systems which can systematically hide or reduce reconfiguration time overhead. Our experiments show the effectiveness of our approach**

I. INTRODUCTION

In wireless communications, it is essential to explore the means to provide the capability of assimilating different communication technologies on the base stations of network service providers. In software defined radios (SDRs), a wide variety of strategies and protocols can be implemented in software. Although the complete processor-based SDR systems provide the ultimate flexibility for SDR configurations, DSP processing mandates that SDR platforms be supported by both processors and hardware (mainly FPGAs). Hardware reconfiguration as well as software programmability not only enables the radio to implement multiple waveforms and standards while meeting the networking protocols' time budget, but also reconfigurable FPGAs bring significant savings in energy and cost compared to pure software implementation of SDR systems.

The main theme of this paper is hardware reconfiguration of SDR systems to support multiple concurrent and/or sequential network standards on FPGA devices. In order to provide QoS for real time traffic, such as VoIP (voice over IP), the operational period of each communication system on the SDR is around 10ms to 100ms so that all packets can receive prompt processing and transmission. In multiple systems on a single SDR platform, the latency of reconfiguring the SDR platform should also fall within that range so that the continuity of service is guaranteed. Hence, it is important to reduce or hide the reconfiguration delay overhead when using runtime reconfiguration in FPGAs for SDR platforms.

The resource allocation, layout and configuration of the existing communication standards on FPGAs play important roles on the overhead of partial reconfiguration of FPGAs. Reconfigurable physical layer can provide and suggest efficient sequences of reconfigurations as well as multiple implementations for different networking systems to networking layer. This paper is a step toward improvement of supporting tools for dynamic reconfiguration of FPGA devices in order to support multiple communication systems on SDRs.

In this paper, we define "Sequence Generation" problem. The goal is to find the sequence for executing the networking systems on FPGA such that the total reconfiguration delay overhead is minimized. We consider three scenarios with hardware reuse on FPGAs as follow:

Case 1: The configuration of each wireless system is already provided. It is pre-designed and configured on particular region on the chip.

Case 2: The relative configuration is given but they can be placed in a set of possible regions on the chip. Also, there exist different implementations for a given protocol. For example, network system CDMA can have a Viterbi or Turbo decoder [5].

Case 3: Among different configuration for each design, some configuration of a system may not overlap with configuration of another system. In such a case, multiple parallel communication system can be realized.

Several wireless communication and networking standards share some DSP functions. In order to reduce reconfiguration overhead, we can exploit reuse to overcome reconfiguration overhead. The shared component can be exactly the same or be parameterized components supporting multiple systems. For instance, for the Viterbi algorithm, K is 5 in GPRS where K is 9 in UMTS [4], and the Viterbi decoder is compatible between UMTS and GPRS with 10% area overhead compared to sole implementation of Viterbi decoder for UMTS [3]. In section III, we discuss the challenges in exploring FPGA resource reuse in generating FPGA configuration sequences. Since location of component reuse needs to be defined for configuration generation, we propose to incorporate sequence generation engine in a floorplanning tool which simultaneously floorplans the layout of all the designs in sequence while exploiting reuse.

II. Related Work

There is a large body of work in architectures of SDR devices. SODA architecture [10] is a multi-processor platform that has been utilized in data-path and architecture for required signal processing for wireless communication. FLAI architecture [16] is a coarse-grained reconfigurable platform which can change functionality by downloading new configuration. Other research work target fine grained FPGAs like Xilinx and Altera FPGAs. WARP [11] is an FPGA based platform which is customized for wireless communication. In [4] the authors propose an FPGA based platform for implementing an SDR device capable of communication over 802.11 and WCDMA networks. In [9] the authors implement a system which can change functionality from GSM to EDGE by partially reconfiguring the FPGA in run time. In [7] the authors exploit reusing modules to avoid reconfiguration overhead. They consider all of the tasks to be of equal area and they exploit similarity between a given set of scheduled task. In [14] the authors propose FPGA area allocation algorithms to reduce total reconfiguration time overhead based on execution frequency of the tasks. In [1] the authors target reducing reconfiguration for a set of scheduled tasks by floorplanning all of them simultaneously. They try to enforce module reuse to minimize reconfiguration time. Some other research works target online resource management for reconfigurable devices. In [15] the authors propose online scheduling system that allocates tasks to a block partitioned reconfigurable device.

III. Sequence Generation of Reconfigurable Designs

Partial reconfiguration delay overhead is linearly dependent on the area on the chip to be configured. As shown in Figure 1, designs T_A and T_B have an overlap area C_{AB} in their configuration on the FPGA. We change the functionality from protocol A (design T_A) to protocol B (design T_B) by reconfiguring the FPGA to execute T_B after execution of T_A. While T_A is executing on the FPGA, the system reconfigures Non-overlapping section of T_B (NC_{B-A}) (area of T_B excluding the overlap area C_{AB}). This is referred to as *configuration prefetch* [8]. We assume that execution of T_A is long enough for reconfiguration of NC_{B-A}. After execution of T_A, C_{AB} is reconfigured after which protocol B begins to execute. Minimizing the area of C_{AB} can minimize reconfiguration time overhead.

In problem formulation and proposed solution, the term "task" refers to configuration of a network system (or protocol). We define Sequence Generation problem as follows:

Given a set of tasks $T=\{T_1, T_2, ..., T_n\}$ and cost c_{ij} associated with each pair of tasks T_i and T_j, as reconfiguration cost between T_i and T_j, the objective is define a sequence of the tasks such that the total cost of reconfiguration in the sequence is minimized.

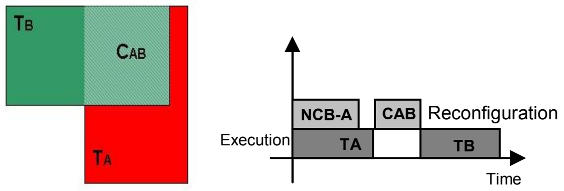
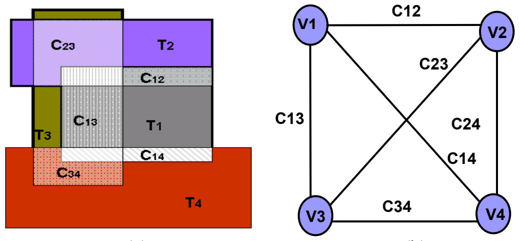


Figure1. Timing Diagram for Partial Reconfiguration from Protocol A to protocol B.

We assume the configuration bit stream of each task is given. The sequence of reconfiguration for the tasks in set T can affect total reconfiguration cost as shown in Figure 2 (a). If we choose the sequence $T_2 \rightarrow T_4 \rightarrow T_1 \rightarrow T_3$ as the sequence of task execution, the total reconfiguration cost is $C_{24} + C_{14} + C_{13} + C_{23}$. However, choosing the sequence $T_4 \rightarrow T_3 \rightarrow T_1 \rightarrow T_2$ results in total reconfiguration cost of $C_{34} + C_{13} + C_{12} + C_{24} > C_{24} + C_{14} + C_{13} + C_{23}$. The problem of finding optimal reconfiguration sequence to minimize total reconfiguration cost can be modeled as *Traveling Salesman Problem* (TSP).

The placement of the modules can be modeled as a graph $G (V, E)$ (Figure 2(b)). Each vertex v_i, in the vertex set $V = \{v_1, v_2, ..., v_n\}$ represents a task in set T. The weight of the edge w_{ij} is C_{ij}. If we start from node v_1 and travel through all the vertices in graph G and return to v_1 in a way that the total weight on the edges is minimized (TSP), we have an optimal solution to our **Sequence Generation Problem (SGP)**.



(a) (b)

Figure 2- Sequence of reconfiguration for 4 tasks and the graph representation of the problem

A. Multiple Configurations

We generalize Sequence Generation problem to include multiple implementations or configurations for each task. It is possible that we are given multiple configurations associated with each task located (e.g. at different regions on the chip). Hence, the more general problem can be defined as follows:

Given a set T of task implementation sets I, $T = \{\{I_{11}, I_{12}, ..., I_{1n}\}, \{I_{21}, I_{22}, ..., I_{2m}\}, ..., \{I_{k1}, I_{k2}, ..., I_{ky}\}\}$ where I_{ij} is implementation j of task T_i, define a sequence of implementation of reconfigurable tasks in such a way that we minimize total reconfiguration cost.

Figure 3 is an example of two tasks with two implementations for each. In graph $G=(V,E)$, vertex set $V = \{v_{11}, v_{12}, \ldots, v_{k1}, v_{k2}, \ldots, v_{ky}\}$ represents multiple task implementations in set T. Node v_{ij} represents implementation j of task T_i. An edge between two vertices is the reconfiguration cost between the two configurations. A set S_i is referred to the set of nodes that represent implementations of task T_i. In graph G, if we can find a cycle in which we start from an arbitrary node and pass through each set S_i just once (visiting just one node of each set) with minimum weight on the visited edges, we can apply the solution to the general case of the SGP in which exactly one configuration per task is chosen and total reconfiguration cost is minimized. In fact this problem is the general case of the TSP known as Generalized Traveling Salesman Problem (G-TSP) [12]. The (G-TSP) can be reduced to the TSP. In [12] the authors propose a transformation that can reduce a G-TSP instance to a TSP problem with the same number of nodes.

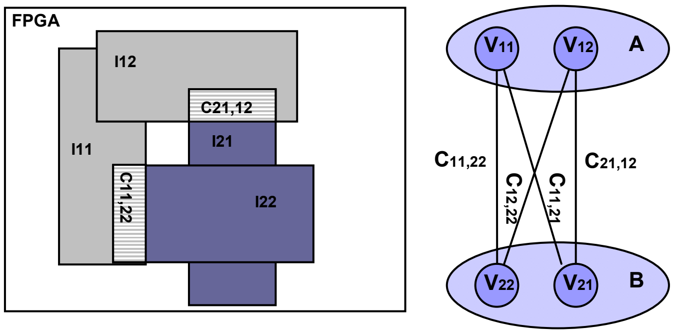


Figure 3- Multiple implementations for tasks *T1* $\{I_{11}, I_{12}\}$ and *T2* $\{I_{21}, I_{22}\}$ *in* graph representation.

B. *Parallel Tasks*

While all the previous cases consider reconfiguring just one task at a time, the general case of the problem should consider running protocols in parallel and also reconfiguring them in parallel. The SGP can be enhanced to cover parallel execution of the tasks. Parallel execution of the tasks means no overlap in their configuration. In Figure 4, implementation 1 of tasks T_1 (I_{11}) and task T_2 can run in parallel as their placement is overlap-free. The graph representation of the tasks includes one vertex for each implementation as well as an additional vertex $V_{11,2}$ which represents the parallel execution of tasks T_1 (implementation I_{11}) and T_2. Node $V_{11,2}$ is the common node across sets A and B meaning that if we visit this node in our tour, we do not need to execute any other task from sets A or B. This problem is also a version of generalized TSP which can be reduced to a TSP problem using the same algorithm mentioned in [12].

IV. Exploiting Component Reuse In Sequence Generation

Component reuse incurs additional physical constraints during implementation of each configuration [6], and therefore the other design objectives like clock speed may not be met. In order to ensure the most benefits from reuse, we need a systematic physical implementation flow which aims at maximizing the overlap between the configuration bits of the two designs while meeting the design constraints.

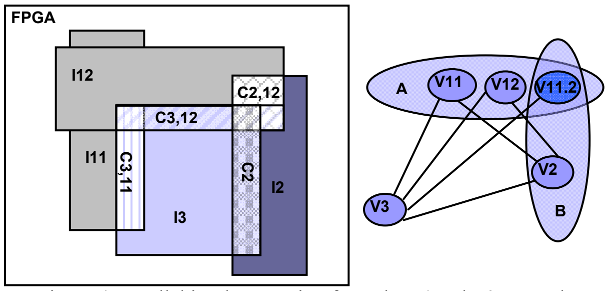


Figure 4. Parallel implementation for tasks *T1* and *T2 in* graph representation.

The reconfigurable (or reused) components are not compatible for all communication systems. For example, if the Viterbi decoder in [3,4] is used, it can only be used for UMTS and GPRS. If the systems to be implemented are listed as {GPRS,UMTS,CDMA} and the device is currently executing UMTS, the sequence {UMTS $\rightarrow$ GPRS $\rightarrow$ CDMA} provides minimum delay overhead since the Viterbi from UMTS can be used for GPRS but not for CDMA. Statically generated sequences can guide the scheduler to be aware of reconfiguration overhead. In this section, the configuration bit stream for each design is independently generated. However, component reuse between two subsequent designs can be deployed during implementation in order to reduce the reconfiguration overhead. Next, we propose to incorporate the sequence generation engine inside a floorplanner to exploit component reuse while generating the sequence of reconfigurable designs.

V. Floorplan-Aware Sequence Generation

We can effectively reduce reconfiguration time overhead to switch between different protocols by properly floorplanning the designs. In this case while it is important how we floorplan each design it is also very important to floorplan the designs while they are aware of each other so that we can effectively reduce total reconfiguration time overhead. Our goal is to floorplan the designs that are in sequence, with minimum overlap but exploiting reuse on common modules. We exploit white spaces in the floorplan of one design for placing the modules of another design to reduce the overlap. We assume that while one design is running, we can reconfigure the white spaces for the next design. Therefore we pay no overhead to reconfigure the chip in the white space regions.

We present a Floorplan-aware Sequence Generation tool. It is based on multi-layer floorplanner presented in [1] to find the best sequence. In this work, we present our method to integrate Sequence Generation problem with floorplanning to better exploit the module reuse and reduce configuration overlaps. Multi-layer floorplanner is a simulated annealing based on Parquet [2] using sequence-pair representation. In [1], reconfiguration of the chip is not initiated until execution of the current task is over. Hence there is no configuration prefetch [8]. The floorplanner aims at maximizing the overlap between the configurations of the given designs under particular given sequence.

In simulated annealing floorplan, there are three essential steps: design representation, random moves, and cost function. The tool iteratively applies random moves to the current solution and based on temperature and cost of new solution, it may reject or accept the new solution after applying the moves. Next, we propose our methods used in design representation, cost function, and moves of the floorplan. Since our floorplanner is based on Parquet and Multilayer floorplanner, we only highlight the new functions and methods that we have introduced in our floorplanner in order to adapt the floorplanner to incorporate reconfiguration delay overhead minimization and best sequence generation.

The modules of each design are represented using multi-layer sequence pairs [1]. Multi-layer sequence pair representation is a sequence pair representing all the modules involved in all of the designs. Reuse modules across multiple designs are represented as one node in multi-layer sequence pair representation. Hence, they are treated the same way in the moves and hence they are placed at the same location with the same orientation and aspect ratio.

A. *Layout Compaction for Overlap Reduction*

The sequence pair representation of the blocks is efficiently used to define the relative location of the modules. Layout compaction algorithm is used to determine the location of each module. In layout compaction algorithm of multi-layer floorplan, the blocks can overlap only if they are not of the same design [1,2]. The cost of reconfiguration between two designs is directly related to the size of the overlap in placement of the two designs. Multi-layer floorplanner aggressively tries to increase this overlap to minimize the area. However this would increase the cost of reconfiguration in our problem. Instead, our floorplanner tries to exploit the white spaces left in each design to place the blocks of the next design. Hence layout compaction algorithm used in multi-layer floorplanner is counter productive for our purpose. For determining the location of the blocks we use the same layout compaction algorithm, however we introduce a new move (***Block Injection*** as we call it) to enforce white spaces among the modules of one design.

B. *Moves to Inject Blocks*

The moves in our floorplanner are the ones (except for Block Injection) that have been used in multi-layer floorplanner. These moves include changing the aspect ratio of the soft blocks, changing the orientation of the blocks and also changing the relative placement of the modules by swapping their representative in sequence pair. In multi-layer floorplanner (and also our floorplanner) module reuse is forced for common modules. Moves in the floorplanner treat the reused modules across different designs in the same way, meaning that the aspect ratio, placement or orientation of the reused modules changes for all the common modules involved.

Module reuse enforcement can be systematically exploited to improve the quality of placement in terms of reconfiguration overhead. As mentioned earlier instead of minimizing total area of placement (which inherently increases overlap and reconfiguration cost) we would prefer exploiting the available resources i.e., area of the chip. Exploiting white spaces of each design for placing the modules of the other design can particularly be effective to reduce total reconfiguration time overhead. In Block Injection move, for each design we try to increase the utilization of the device for that design to a certain portion (which is a random number from 0% to 70% of the device area) by inserting white space blocks with the same area and aspect ratio of some of the blocks of other designs (which we randomly select). These enforced white space blocks fit among other blocks of the design during layout compaction. The benefit of this move is two folds. First, it enforces white space which can be fully/partially exploited for placing the modules of the other designs which results in reduced overlap among designs. Second, it increases similarity among designs (by injecting a block of one design as white space block to the other design) which can effectively reduce reconfiguration time between the two designs.

C. *Reconfiguration Cost Function*

We calculate the cost of the new moves using our cost function. Our cost function addresses area of the design, the wire length, the aspect ratio and also configuration cost. In our problem configuration cost is the cost of reconfiguring the designs one after the other one while we just consider the overlapped area of the two consecutive designs for reconfiguration.

Once we have the floorplan of each design after layout compaction, we first calculate the cost of reconfiguration for every two designs. The algorithm to compute the cost is simple. We sort all the modules involved in floorplanning based on their current X coordination. Starting from the left-most block (with lowest X coordinate) we calculate the overlap with the blocks of the other designs. Finding the overlap of the sorted blocks in worst case is $\Theta(n^2)$. In this stage, we are ready to incorporate the sequence generation engine. Given that we have the reconfiguration cost between every two design, we apply our algorithm to find the sequence of the designs which results in lowest reconfiguration overhead. For efficiency of the floorplanner, we apply existing fast heuristic algorithms for TSP problem if needed (in our floorplanner we integrate Concord TSP solver [13]). The reconfiguration cost of the best sequence is in our cost function. Our cost function is as follows:

$$Cost = \alpha \times Area + \beta \times WireLength + \varphi \times AspectRatio + \gamma \times ConfigCost$$

Therefore, associated with each floorplan generated for the tasks, the sequence of the tasks with minimum reconfiguration delay overhead is generated. Hence, the tasks with their corresponding configuration, generated from the solution with the best cost function in this floorplanner, can potentially be sequenced with very low reconfiguration delay overhead since it is incorporated in the cost function of the floorplanner. The early physical planning during floorplanning of the tasks enable maximum module reuse as well as minimum overlap in the layouts while trying to improve the performance of each task (by wirelength minimization).

VI. Benchmarks Setup And Experiments

We do our experiments on the physical layer implementation for 4 different protocols. These protocols are very similar to 802.16a (Protocol A), 802.11a (Protocol B), WCDMA (Protocol C) and CDMA (Protocol D). A selection of distinctive and important components of these protocols like Turbo Enc/Dec, RS Enc/Dec, FFT/IFFT, etc is implemented on the target FPGA. Table 1 presents information regarding the size of the protocols. Similarity among the protocols shows the savings in reconfiguration if we were to reconfigure between the two protocols assuming that reuse of similarities is enforced.

The FPGA architecture that we have assumed for our benchmarks is Virtex 4 architecture. We run the experiments for different device sizes. We assume the setup mentioned in [17] as the setup for our self-reconfiguring platform through ICAP. The bitstreams are downloaded from the external memory to the local memory of the processor which controls the reconfiguration (phase 1 of reconfiguration). From the processor's local memory the bitstreams are transferred to ICAP's internal buffer (phase 2) where ICAP fetches the configuration bits and reconfigures the FPGA device (phase 3). In [18] the authors measure the delay of each phase for a Virtex 2 device.

Throughout our experiments we assume that during execution time of the current design, partial bitstream is already downloaded in the local memory and placed in ICAP's buffer. Hence, reconfiguration time overhead is meant to include the last phase of delay for reconfiguration. This delay for one configuration frame is the same across all V4 devices and is equal to 13.1us for one frame.

Table 1. Protocol Configuration Statistics and Protocol-to-Protocol Similarity

	Design Size (slices)	Similarity to A (%)	Similarity to B (%)	Similarity to C (%)	Similarity to D (%)
Protocol A	20640	100	67	21	33
Protocol B	20160	67	100	25	40
Protocol C	14240	21	25	100	45
Protocol D	12640	33	40	45	100

Our experiments are intended to show the importance of several key features of our floorplanner. For this we have designed two sets of experiments as follows:

Experiment 1 is designed to floorplan the four designs at the same time without considering configuration cost in the cost function. The floorplanner is intended to find a placement with minimum wire length inside the device boundaries while exploiting maximum reuse among the designs. Once we find the best placement for our objective we find the best sequence (for that placement) that minimizes reconfiguration cost. This is the scenario we described in Section III. After generating the layout of all four designs, we want to find the best sequence which minimizes total reconfiguration time. To show the importance of finding the best sequence we also report the worst sequence which increases total reconfiguration time. The results of this experiment are shown in Table 2. The cost numbers shown in Tables 2, 3 and 4 are total reconfiguration time overhead in **ms**. The results show the importance of finding the best sequence in a predesigned floorplan. As can be seen wrong selection of the sequence can increase total reconfiguration time overhead by more than two times.

Table 2. Best/Worst Sequence/Cost for Floorplanner of Experiment 1

(Cost in ms)	Device (row x column) (slices)			
	Custom FPGA (100 x 320)	LX80 (112 x 320)	LX100 (128 x 384)	LX160 (176 x 384)
Best Seq/Cost	ABCD/26.9	ABCD/35.3	ABCD/6.7	ABCD/2.9
Worst Seq/ Cost	ACBD/53.7	ACBD/67	ACBD/14.7	ACBD/5.9

In Experiment 2, we include configuration cost as part of our cost function and apply our proposed floorplanner coupled with TSP-based sequence generation. The best placement is selected if it meets the device constraints and improves wire length and reconfiguration cost of the protocols. Table 3 reports the best sequence and corresponding reconfiguration overhead resulted in Experiment 1 and Experiment 2. In all the cases above, our floorplanner could find a placement which can actually totally hide reconfiguration cost. This efficiency in eliminating the cost is the result of reusing common modules across different designs and as well the efficiency of the floorplanner in exploiting white spaces. This table shows the importance of our approach to attack the problem at floorplanning level instead of just determining the best sequence for a set of already placed designs.

Table 3. System-Level Approach Vs. Floorplan-Level Approach

(Cost in ms)	Device (row x column) (slices)			
	Custom FPGA (100 x 320)	LX80 (112 x 320)	LX100 (128 x 384)	LX160 (176 x 384)
System-level	ABCD/26.9	ABCD/35.3	ABCD/6.7	ABCD/2.9
Floorplan-level	ABCD/0	ABCD/0	ABCD/0	ABCD/0

In Table 4 the first row presents the best reconfiguration sequence and cost we can get by our floorplanner. We also show the worst sequence and cost for the same placement on the second row. The third row shows the worst sequence and reconfiguration cost among all feasible placements (the placements that meet device constraints and improve wire length) we found in this experiment.

Table 4. Best/Worst Sequence/Cost from the proposed Floorplanner

(Cost in ms)	Device (row x column) (slices)			
	Custom FPGA (100 x 320)	LX80 (112 x 320)	LX100 (128 x 384)	LX160 (176 x 384)
Best Seq/Cost	ABCD/0	ABCD/0	ABCD/0	ABCD/0
Worst Seq/ Cost	ABDC/25.2	ABDC/9.0	ABDC/0	ABCD/0
Worst Seq/ Cost in Solution Space	ACBD/159.5	ACBD/159.1	ACBD/175.9	ACBD/100.8

Results show that reconfiguration cost for feasible placements can be extremely high however the right selection of floorplan and sequence can totally hide the reconfiguration cost as is the case in Table 4. In devices LX100 and LX160, for the same placement, the best and the worst reconfiguration sequence is the same and the cost is 0. This is because the device was so large that all the distinct modules in the design of the 4 protocols could fit on the device without any overlap with the uncommon modules of the other designs. Although placement becomes somehow trivial under this condition this observation also shows that our floorplanner is capable of exploiting these chances for efficient placement.

VII. CONCLUSION

In this paper we studied Sequence Generation Problem (SGP). We showed that to minimize total reconfiguration overhead sequence of reconfiguration is very important and we proposed a TSP based solution to the problem at system-level. We studied the problem as a floorplanning problem. The objective is to minimize total reconfiguration time overhead by properly floorplanning the designs as well as finding the best sequence which minimizes the overhead. The result of our experiments at system-level shows that finding the optimal reconfiguration sequence can be very crucial to minimize reconfiguration overhead.

ACKNOWLEDGMENT

This work is partially supported by NSF ECCS-IHCS grant #0725914.

REFERENCES

[1] L. Singhal and E. Bozorgzadeh, "Multi-layer Floorplanning on a Sequence of Reconfigurable Designs", *FPL* 2006.
[2] S. N. Adya and I. L. Markov, "Fixed-outline floorplanning through better local search," in *IEEE International Conference on Computer Design (ICCD)*, 2001.
[3] L. Bissi and P. Placidi and G. Baruffa and A. Scorzoni, "A Multi-Standard Reconfigurable Viterbi Decoder using Embedded FPGA Blocks", *DSD '06: EUROMICRO Conference on Digital System Design,* 2006.
[4] H. Harada, "Software defined radio prototype for W-CDMA and IEEE802.11a wireless LAN", IEEE Vehicular Technology Conference, 2004.
[5] Cavallaro J.R. and Vaya M. "Viturbi: a reconfigurable architecture for Viterbi and turbo decoding", ICASSP '03: Proceedings of Acoustics, Speech, and Signal Processing, 2003.
[6] S. Banerjee and E. Bozorgzadeh and N. Dutt, "Physically-aware HW-SW partitioning for reconfigurable architectures with partial dynamic reconfiguration", pages 335—340, *DAC 2005.*
[7] S. Ghiasi, M. Sarrafzadeh, "Optimal Reconfiguration Sequence Management", *ASPDAC*, 2003.
[8] S. Hauck, "Configuration pre-fetch for single context reconfigurable processors", *ACM FPGA Symposium*, 1998.
[9] Delahaye, J.P., Gogniat, G., Roland, C. and Bomel, P. 'Software radio and dynamic reconfiguration on a DSP/FPGA platform', in Rykaczewski, P. and Schmidt, M. in special issue on *Software Defined Radio of Frequenz*,
[10] Yuan Lin, Hyunseok Lee, Mark Woh, Yoav Harel, Scott Mahlke, Trevor Mudge, Chaitali Chakrabarti, Krisztián Flautner, "SODA: A High-Performance DSP Architecture for Software-Defined Radio," *IEEE Micro 2007*
[11] P. Murphy, A. Sabharwal, and B. Aazhang. "Design of warp: A wireless open-access research platform". *In proceedings of European Signal Processing Conference Program, 2006*
[12] Behzad, A. and Modarres, M, "A new efficient transformation of the generalized traveling salesman problem into traveling salesman problem", *Proc. of International Conference on Systems Engineering, 2002*
[13] http://www.tsp.gatech.edu/concorde.html
[14] E. Moscu Panainte, K.L.M. Bertels, S. Vassiliadis, "FPGA-area Allocation for Partial Run-Time Reconfiguration", Proceedings of ProRISC, November 2005
[15] Herbert Walder, Marco Platzner, "Online Scheduling for Block-Partitioned Reconfigurable Devices", DATE 2003
[16] Mei B., Vernalde S., Verkest D., De Man H., Lauwereins R. "ADRES: An Architecture with Tightly Coupled VLIW Processor and Coarse-Grained Reconfigurable Matrix", *FPL 2003*
[17] Blodget, B., James-Roxby, P., Kelle, E., McMillan, S., Sundararajan, P.: "A Self-Reconfiguring Platform", *FPL 2003*
[18] Papadimitriou, K., Anyfantis, A., Dollas, A., "Methodology and Experimental Setup for the Determination of System-level Dynamic Reconfiguration Overhead," *Field-Programmable Custom Computing Machines, 2007. FCCM 2007.*