An Automated Hardware/Software Co-Design Flow for Partially Reconfigurable FPGAs

Shaon Yousuf* and Ann Gordon-Ross
*Currently affiliated with Intel Corporation
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Department of Electrical and Computer Engineering
University of Florida, Gainesville, FL-32611
{yousuf, ann}@hcs.ufl.edu

Abstract-Partial reconfiguration (PR) enhances traditional FPGA-based reconfigurable embedded systems with benefits such as reduced resource requirements and increased functionality. Since fully realizing these PR benefits requires extensive PR design flow knowledge, as well as the target FPGA's low-level architectural details, PR has not yet gained widespread usage. To alleviate manual design-time effort, we present the design automation for partial reconfiguration (DAPR) design flow for hardware/software (HW/SW) co-designed systems. DAPR's design flow isolates low-level PR design complexities involved in analyzing PR designs with different performance parameters to make PR more amenable to designers.

Keywords – FPGAs, Reconfiguration, Partitioning, Floorplanning, Hardware/Software, Automation

1. INTRODUCTION AND MOTIVATION

For execution flexibility in field programmable gate arrays (FPGAs)-based reconfigurable systems, application design is typically modularized [9]. An application's functionality can be represented as a set of interacting modules, where each module represents an application's task, and collectively, all tasks perform the complete application's functionality. FPGAs with partial reconfiguration (PR) provide added flexibility using isolated reconfiguration when loading/unloading application tasks (i.e., only the hardware (HW) being reconfigured halts operation while the remainder of the system remains operational). Designers specify partially reconfigurable regions (PRRs) to execute these tasks, which enables time-multiplexing of HW resources between mutually exclusive tasks, which provides multiple benefits such as: reduced HW, power, and memory requirements as compared to application specific integrated chips (ASICs) or non-PR FPGAs.

Cutting-edge PR FPGAs also include software (SW) processors, which enables HW/SW (HW/SW) co-designed systems where application tasks can be executed on the SW processor and/or in the HW PRRs. Mapping tasks to execute in SW can potentially reduce an application's HW requirements even further in addition to the PR benefits. However, SW task mapping must be carefully considered since executing too many tasks in SW can severely degrade system performance by not leveraging the specialized HW resources and parallelization available in the HW PRRs. Alternatively, mapping too many tasks to the HW PRRs may increase overall HW requirements, and may not even improve system performance since not all tasks benefit from HW execution. Thus, incorporating PR in a HW/SW co-design flow introduces new challenges and requires a more specialized design flow.

Much research focuses on methods for designing and evaluating HW/SW co-designed systems to ensure that a designer's goals are met. Typically, the first step in the HW/SW co-design flow is HW/SW PR partitioning, which is divided into two phases: HW/SW partitioning and PR partitioning. HW/SW partitioning does a high-level mapping of tasks into SW and/or HW, and PR partitioning allocates the HW tasks from the first phase into one or multiple PRRs. The second step in the HW/SW co-design flow is PR floorplanning, which is divided into two phases: PRR floorplanning and partition pin floorplanning [9]. PRR floorplanning sets the physical placements of the PRRs on the FPGA fabric. Partition pin floorplanning sets the physical placements of the Xilinx stipulated specialized hardwired communication interfaces or partition pins, on PRR boundaries. The final step of the HW/SW co-design flow is to generate the design bitstreams and binary executables.

HW/SW PR partitioning is a resource and performance critical steps in the HW/SW co-design flow. Typically, applications have a very large number of functionally-equivalent HW/SW PR partitions with different resource and reconfiguration time [9] tradeoffs, and choosing the PR partitioning that most closely adheres to a designer's goals requires manually exploring all of these HW/SW partitions. Given this extremely large design space (finest to largest grained ranging from a single instruction per PRR to one PRR with the entire application, respectively), this process is difficult, cumbersome, and requires lengthy design exploration time to evaluate the different partitioning options.

Once the designer selects a HW/SW PR partition that meets the designer's goals—the *chosen* partition—the next resource and performance critical step is PR floorplanning, which assigns physical FPGA resources to the chosen partition's PRRs, and the partition pins surrounding the PRRs boundaries. PRR resources are assigned by defining constraints that specify the PRR's dimensions (size and shape) in terms of an XY coordinate system [9] on the FPGA. Partition pin resources are assigned by defining constraints

978-1-4673-9039-2/16 $31.00 © 2016 IEEE
DOI 10.1109/ISVLSI.2016.73

that specify the partition pins' locations around the PRR boundaries in terms of the XY coordinate system. Given a device's resource layout and the chosen partition's PRRs' resource requirements, there are many different functionally-equivalent PR floorplans with different resource and clock frequency requirements, resulting in an extremely large PRR and partition pin design space requiring manual exploration to meet a designer's goals.

Given this arduous design process, PR designers require automated design assistance to develop efficient designs, however, currently PR designers have little automated support, and vendor-supported PR design flows still require manually exploring HW/SW PR partitioning and PR floorplanning. Most existing research on HW/SW co-design for PR FPGAs presents various techniques to determine a HW/SW PR partition and PR floorplan, but considers these design flow steps independently, providing only partial solutions, and not an integrated, holistic PR design flow solution. Moreover, no previous work considered the effect of partition pin floorplanning, which is important, since the partition pin floorplan can greatly affect the final design's total wire length, and thus, the design's clock frequency. To the best of our knowledge, there exists no previous effort to completely automate design exploration for both HW/SW PR partitioning and PR floorplanning.

To address this dearth in design flow automation, we present the design automation for partial reconfiguration (DAPR) design flow, which automatically performs HW/SW PR partitioning and PR floorplanning for a HW/SW co-designed system. The DAPR design flow surpasses traditional PR design flows using automation that eliminates the need for designers to be familiar with low-level FPGA details and complexities, and thus significantly reduces manual design time effort, making PR more amenable to a larger range of designers.

2. RELATED WORK

Currently, two FPGA device vendors support PR: Xilinx and Altera. Xilinx's and Altera's PR design flows follow the same principle steps, but in our case we consider the more mature and prevalent Xilinx PR design flow [9].

Xilinx's PR design flow requires a manual hierarchical logical partitioning of the design files into a top module, static module(s), and one or more non-overlapping partially reconfigurable modules (PRMs), where the PRMs constitute the application's tasks and are mapped to PRR(s). Next, the designer must execute the following steps: (1) manually synthesize each module's files separately to generate the module's respective netlists; (2) set the PR design's floorplanning design constraints in a Xilinx-specific user constraints file [9]; (3) implement non-PR designs for every PRM-to-PRR mapping combination and perform timing analysis on each design to verify timing requirements; (4) generate place and route (PAR) information for the static region to create a static design with "holes" (un-placed and un-routed regions) for the PRMs and then generate PAR information for each respective hole's (i.e., PRR's) PRMs; and (5) merge the static design's PAR information with each PRM's PAR information to generate the PR design's full bitstream and partial bitstreams. Currently, both Xilinx's PR design flow and Altera's PR design flow do not support HW/SW co-design.

Previous work [2] proposed a framework for an automated PR design flow. The framework proposed partitioning PR designs from an application's task flow graph and performed automated floorplanning using heuristics, but no methodology for partitioning or floorplanning was presented. The authors manually investigated and evaluated clock frequency and partial bitstream size with respect to the PRR aspect ratio (PRR height divided by PRR width) for PRMs with different resource requirements. The authors did not investigate PRR floorplanning with respect to PRR placement locations on the device.

The GOAHEAD PR design flow framework [1] aided PR designers in PR partitioning and PR floorplanning, but presented no PR partitioning methodology. The design flow did include an automated PRR floorplanning methodology that placed PRRs starting from the center of the device and a partition pin floorplanner that placed partition pins on the left and/or right borders of the PRR starting from the bottom. The authors did not present any results evaluating the effectiveness of the floorplanning methods.

Montone et al. [6] presented a PR partitioning and floorplanning methodology that leveraged a scheduled application's task flow graph. The floorplanning methodology considered minimizing area requirements. Since the presented partitioning methodology leveraged a single scheduled task flow graph, the effects of partitioning for PR systems with varying functionality was not considered.

Vipin et al. [7] presented a PR design flow that performed automated partitioning of a PR design's HDL design descriptions, performed automated PRR floorplanning, and outputted the PR designs' bitstreams using Xilinx utilities, similarly to Xilinx's PR design flow. The partitioning methodology considered PR systems with varying application functionalities but did not consider the effects of HW/SW co-design during partitioning. The authors did not include results on the quality of the final generated bitstreams and no investigation on how partition pin floorplans affect the final PR design were presented.

3. The DAPR Design Flow

HW/SW PR partitioning determines the number of HW tasks and number of PRRs, which directly affects the HW resource requirements and reconfiguration time overheard of a HW/SW co-designed application. HW/SW PR floorplanning determines the HW tasks physical placements and

wire routing on the FPGA assigned during HW/SW PR partitioning, which directly affects the clock frequency and overall execution time of a HW/SW co-designed application. Therefore, it can be logically concluded that HW/SW PR partitioning is an ideal design step to optimize the HW resource requirements and reconfiguration time for HW/SW co-designed applications, and PR floorplanning is an ideal design step to optimize a HW/SW co-designed application's clock frequency. Figure 1 shows the proposed DAPR design flow based on these considerations.

The DAPR design flow takes as input a modularized application, performs initial analysis to generate input parameters (Section 3.1) for HW/SW PR partitioning, which leverages the input parameters and runs an exhaustive search algorithm to generate a set of functionally-equivalent HW/SW PR partitions with associated total resource requirements, reconfiguration time, and HW and SW execution times. The designer then analyzes this set and chooses the HW/SW PR partition that most closely adheres to the designer's goals—the *chosen* partition.

Next, PR floorplanning improves the clock frequency of the chosen partition with respect to a system designer specified target architecture and device. The clock frequency is improved using a simulated-annealing (SA)-based algorithm (number of iterations are designer specified). After floorplanning completes, the PR design's bitstreams, and binary executables are generated.

3.1 DAPR's Initial Analysis

PR systems can dynamically change the application's currently executing functionality during runtime and thus can have multiple application task flow graphs. We designate each unique task flow graph as a *configuration*. Figure 2 shows an example of two possible configurations for the same application with five tasks T1-T5.

Typically for HW/SW PR partitioning, a simple method for HW and SW task assignment is to choose any suitable configuration and generate a HW/SW PR partition optimized for that chosen configuration. However, using this simple method may severely impact the performance of another configuration by requiring significantly higher reconfiguration time. Another simple method for HW and SW task assignment is to assign control flow related tasks to SW and computationally-intensive tasks to HW, which is a logical and feasible assignment. However, assigning all computationally-intensive tasks to HW may require a lot of HW resources and may be infeasible due to system designer goals and device constraints. Furthermore, due to the availability of high speed processors in modern FPGAs, some computationally-intensive tasks executing in SW may have comparable speeds to HW, so there is no need to run that particular task in HW, which would incur additional reconfiguration time overhead. Thus, there is no one simple solution to determine a HW/SW PR partition and to properly determine a HW/SW PR partition best suited to the system designer's goals—every combination of HW/SW task assignment must be evaluated to determine an optimal partition that balances resource requirements, reconfiguration time, and total execution time within a large design space. To perform this evaluation for every combination of HW/SW assignments for an application, for each task i, we define three parameters: HW resource requirements (RR_i), HW reconfiguration time (TR_i), and HW or SW execution time (CI_i). Calculating RRi, TRi, and CI_i enables exhaustively searching the HW/SW partition design space quickly to estimate a HW/SW partition's design performance and area.

RR_i is determined from the Xilinx's synthesis tool [10], which outputs the HW tasks' resource requirements, which includes: configurable logic blocks (CLBs), block RAMs (BRAMs), first-in first-out buffers (FIFOs), and digital signal processors (DSPs) [11]. TR_i is determined by calculating the total number of frames (the smallest addressable unit of a device) with respect to RR_i. Example Xilinx's Virtex-5 requirements one CLB, DSP, and BRAM contain 36, 28, and 36 frames, respectively [10]. CI_i is estimated by analyzing the modularized application's required operations. For each task's operation, the number of operators (arithmetic, relational and logical, increment and decrement, etc.) are counted, multiplied by the clock cycles required for that operation, and then summed to determine the corresponding total SW or HW execution time. SW and HW clock cycles for operators are acquired from available documentation. For example, Xilinx's Virtex-5 multiply and divide require 3 and 34 clock cycles, respectively [11].

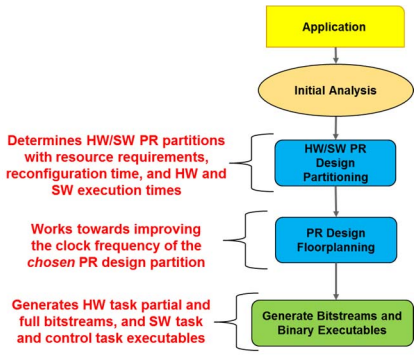

Figure 1. The DAPR design flow.

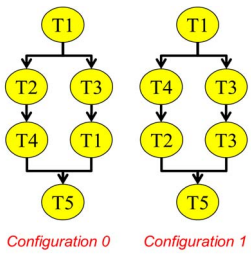

Figure 2. Two functionally-equivalent configurations for an application with five tasks: T1 to T5

3.2 HW/SW PR Partitioning

Figure 3 depicts our HW/SW PR partitioning algorithm, which takes as input a modularized application's task list T_i, and the tasks' corresponding RR_i, TR_i, and CI_i determined from the initial application analysis (line 1). The algorithm also takes as input the list of possible configurations, CO_i, provided by the designer (line 1), where each CO_i specifies the tasks in that configuration. Line 3-6 initialize four dynamically allocated hash table data structures: %Partition_list contains the possible HW/SW PR partitions for each configuration; %Partition_max_resources contains the maximum possible resource requirements for each configuration's all possible HW/SW PR partitions; %Partition_reconfig_time contains the maximum possible reconfiguration time for each configuration's all possible HW/SW PR partitions; %Partition_exe_time_configuration contains the maximum possible execution time for each configuration's all possible HW/SW PR partitions.

Line 7 populates %Partition_list by taking as input the list of tasks in each configuration CO_i and partitioning the tasks into separate pieces that satisfy the Bell recursion, where each piece is non-empty and disjoint, and the union of the pieces is the complete set of tasks. Bell recursion enumerates all possible HW/SW PR partitions, which enables an exhaustive search of all HW/SW PR partitions. For example, the number of all of the possible partitions in a set of n elements are known as the Bell number, and satisfy the recursion: B(0) = 1, and B(n+1) = (n over 0)B(0) + (n over 1)B(1) + ... + (n over n)B(n). The primary key in the hash function represents the configuration number, the secondary key represents the HW/SW PR partition number, and the tertiary key represents the task's HW or SW allocation.

Exhaustive search is feasible since T_i is not large for typical real-world PR applications, and the corresponding RR_i, TR_i, and CI_i inputs (line 1) are pre-determined. For example, for a PR design with 16 tasks, our algorithm determined all possible partitions in approximately five minutes, when running on an Intel® Core ™ 7 2.5 GHz CPU with 8 GB of RAM. Typical modularized real-world PR applications such as in [3][5][12], require the same or fewer numbers of tasks, and thus this estimation is realistic.

```
1  Input:  @Ti, @RRi, @TRi, @CIi, @CO_i
2  Output: List of HW/SW PR partitions
3  %Partition_list;
4  %Partition_max_resources ← 0;
5  %Partition_reconfig_time ← 0;
6  %Partition_exe_time ← 0;
7    %Partition_list ← Partition_task($CO_i);
8    %Partition_max_resources ← Find_max_resources (%Partition_list, @Ti, @RR_i);
9    %Partition_ reconfig_time ← Find_ reconfig_time (%Partition_list, @Ti,  @TR_i);
10   %Partition_exe_time ←Find_exe_time(%Partition_list, @Ti, @CI_i);
11 Return list(%Partition_max_resources, %Partition_ reconfig_time,%Partition_exe_time)
```

Figure 3. HW/SW PR partitioning algorithm

Lines 9-11 populate %Partition_max_resources, %Partition_reconfig_time, and %Partition_exe_time for each configuration's HW/SW PR partition using Find_max_resources(), Find_reconfig_time(), and Find_exc_time(). This populates the %Partition_max_resources, %Partition_reconfig_time, and %Partition_exe_time hash tables, respectively, using the task list T_i in conjunction with the tasks' resource requirements RR_i, reconfiguration times TR_i, and execution times (HW or SW) CI_i, respectively.

At the algorithm's completion (line 11), *list()* uses the *%Partition_max_resources*, *%Partition_reconfig_time*, and *%Partition_execution_time* hash tables to output the set of all possible HW/SW PR partitions and associated resource requirements, reconfiguration times, and HW and SW execution times. These outputws results can be evaluated quickly by the designer to select the *chosen* partition.

3.3 PR floorplanning

Figure 4 shows our PR floorplanning methodology, which takes as input the *chosen* partition, a modularized application, target architecture and device information, and the designer's goals to create an initial *candidate* PR floorplan. The candidate PR floorplan's clock frequency is then evaluated using the PR design's bitstreams. If the clock frequency meets the designer's goals, the design's bitstreams and binary executables are output.

If the clock frequency goals are not met, the *candidate* floorplan is updated by our specialized SA-based algorithm's perturbation function until the clock frequency goals are met or for a design-specified number of iterations (defaults to 100 iterations). On each iteration, the floorplan is perturbed by updating either the partition pin floorplan or the PRR floorplan, but perturbs PRR floorplans more frequently, which more quickly explores the PR design space (partition pin floorplans have less effect on the overall clock frequency [13]). We refer the reader to [13] for DAPR's PR floorplanning methodology details.

4. DAPR Design Flow Evaluation

4.1 Experimental Setup

We evaluated the DAPR design flow on a JPEG CODEC application, which consists of the encoding process

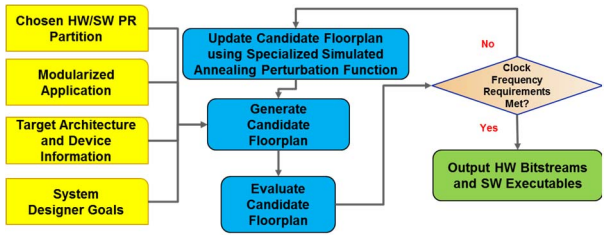

Figure 4. DAPR Floorplanning Methodology

and decoding process [8]. Initial analysis of the JPEG CODEC application revealed two possible corresponding configurations for the JPEG encoding and decoding process. Figure 5 (A) and (B) show the encoding and decoding configurations, respectively. The encoder's configuration tasks are: (1) red, green, and blue to luma, chroma blue, and chroma red (RGB2YCbCr); (2) forward discrete cosine transform (FDCT); (3) run length encoding; (4) Huffman encoding; (5) byte stuffer and header encoding; (6) quantization; and (7) zigzag. The decoder's configuration tasks are: (1) YCbCr to RGB; (2) inverse discrete cosine transform (IDCT); (3) run length decoding; (4) Huffman decoding; (5) byte stripper and header decoding; (6) dequantization; and (7) re-order.

We leveraged VAPRES [4] for our target architecture, which consists of a MicroBlaze processor, a set of static peripherals, and a system designer specified number of PRRs. DAPR design flow's generated HW full bitstream sets up VAPRES and HW partial bitstreams run on VAPRES's PRRs. DAPR design flow's binary executables (SW tasks and application control flow related tasks) run on the MicroBlaze, where loading HW tasks (reconfiguration) is performed via the internal configuration access port (ICAP). The control flow related tasks are automatically generated by the DAPR design flow, and intervenes SW task calls in the modularized application with corresponding HW task calls (leveraging the ICAP) with respect to the chosen HW/SW PR partition.

Our JPEG CODEC experiments executed on a PC with an Intel® Core ™ 7 2.5 GHz CPU with 8 GB of RAM. Algorithms were coded in Perl 5. Xilinx ISE Design Suite 14.7 generated JPEG CODEC's HW bitstreams and SW executables for VAPRES running on Virtex-5 LX110T.

4.2 Results and Analysis

Table 1 shows the JPEG encoder and decoder tasks' resource requirements RR_i (CLBS, BRAMS, and DSPs), the corresponding reconfiguration times TR_i in frames, and the HW or SW execution times CI_i in cycles, determined from initial analysis.

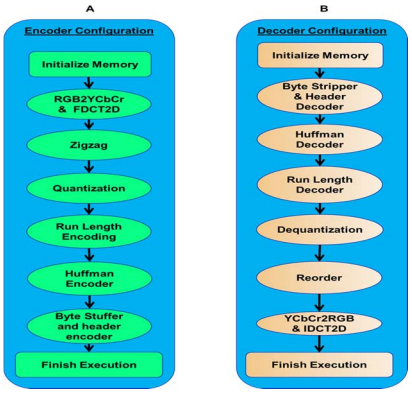

Figure 5. JPEG CODEC configurations: (A) encoder, and (B) decoder

Table 1. JPEG CODEC estimated task resource requirements, reconfiguration times, and HW and SW execution times

Task Name	RR_i (CLBs)	RR_i (DSPs)	RR_i (BRAMs)	TR_i (Frames)	Ci_i (SW cycles)	Ci_i (HW cycles)
RGB2YCbCR and FDCT	406	7	5	1650	40,000	10,000
YCbCrtoRGB and Inverse DCT	400	7	5	1600	42,000	10,000
Run Length Encoding	42	0	2	480	10,000	2,000
Run Length Decoding	42	0	2	480	10,000	2,000
Huffman Encoder	280	1	2	1000	25,000	10,000
Huffman Decoder	350	1	2	1100	30,000	10,000
Byte Stuffer and Header Encoder	14	0	1	100	20	20
Byte Stripper and Header Decoder	14	0	1	100	20	20
Quantization	14	2	3	170	40	40
Dequantization	14	2	3	170	40	40
Zigzag	14	0	2	120	40	40
Re-order	14	0	2	120	40	40

Figure 6 (A-1) and Figure 6 (A-2) depict our JPEG CODEC encoder and decoder configurations' resource requirements and reconfiguration times for each generated HW/SW PR partition, respectively. As expected, there are a multitude of possible HW/SW PR partitions available depending on the designer goals. If the designer's goal is to select a HW/SW PR partition with low resource requirements and reconfiguration time, the designer should choose any HW/SW PR partition between 1 and 1000. However, these HW/SW PR partitions may have unacceptable total system execution times with respect to designer goals. Figure 6 (B-1) and Figure 6 (B-2) depict the encoder and decoder configurations' total system execution times for each HW/SW PR partition. Holistically evaluating the data in these figures enables designers to easily make an informed choice on a HW/SW PR partition that will meet designer goals for multiple configurations. For example, if a PR designer's goal is a HW/SW PR partition with resource requirements below 12,000 slices, a reconfiguration time below 9,000 frames, and a total system execution time below 25,000 cycles for both the encoder and decoder, the designer can choose any HW/SW PR partition between 1 and 1000. To ease analysis, we leveraged an in-house generated Pareto function to parse the generated results and aid in quickly choosing a HW/SW PR partition [13]. The Pareto-optimal function takes three parameters as input (resource requirements, reconfiguration time, and total execution time) and outputs the Pareto-optimal designs that tradeoff the selected parameters. Similarly, a system designer can easily write their own customized parser to analyze the results.

We leveraged our Pareto-optimal function to determine the Pareto-optimal HW/SW PR partitions with respect to execution time and resource requirements. Then, from these Pareto-optimal results we arbitrarily picked the HW/SW partition with the lowest execution time and evaluated our DAPR tool floorplanner on this *chosen* HW/SW PR partition to reduce the execution time further by improving the clock frequency.

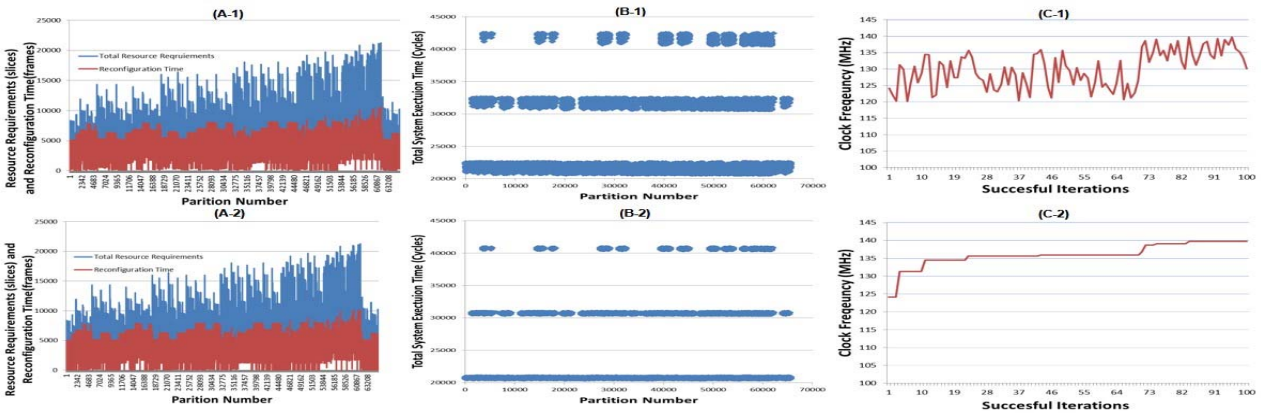

Figure 6. JPEG CODEC configuration's resource requirements and reconfiguration time for generated HW/SW PR partitions: (A-1) encoder configuration and (A-2) decoder configuration. JPEG CODEC configuration's total system execution time for generated HW/SW PR partitions: (B-1) encoder configuration and (B-2) decoder configuration. Chosen HW/SW partitions iteration's clock frequency improvement: (C-1) Clock frequency discovered versus number of successful iterations, and (C-2) highest clock frequency discovered versus successful iterations.

Using our SA-based algorithm, Figure 6(C-1) shows the chosen partition's clock frequency design space exploration versus successful iteration (an iteration with a valid floorplan that passes PAR), and Figure 6 (C-2) shows the highest clock frequency discovered versus successful iteration. As expected, Figure 6 (C-2) shows that the greatest clock frequency improvements are achieved during the first several iterations (12-20 on average).

5. Conclusions

In this paper, we presented and evaluated our design automation for partial reconfiguration (DAPR) design flow. The DAPR design flow aids designers with performance critical partitioning and floorplanning steps involved in a HW/SW co-designing of PR systems. DAPR design flow's partitioning methodology leverages initial analysis to quickly and exhaustively determine an application's all possible partitions' resource requirements, reconfiguration times, and total execution times. A designer then analyzes these designs to choose a PR partition that most closely adheres to the designer's goals. Next, the chosen partition's clock frequency is improved using DAPR's customized simulated annealing-based floorplanning methodology, which improved the clock frequency of the partition by ~10% within an average of 12-20 iterations.

The DAPR design flow's key contributions include: making PR designs more accessible to a wider range of designers; facilitating rapid design prototyping; and creating high-performance systems with reduced design time effort.

6. Acknowledgements

This work was supported in part by the I/UCRC Program of the National Science Foundation under Grant No. EEC-0642422. We gratefully acknowledge tools provided by Xilinx.

7. References

[1] Beckhoff, C.; Koch, D.; Torresen, J., "Go Ahead: A Partial Reconfiguration Framework," Field-Programmable Custom Computing Machines (FCCM), IEEE 20th Annual International Symposium, April 29 2012.

[2] Conger, C.; Gordon-Ross, A.; George, A., "Design Framework for Partial Run-Time FPGA Reconfiguration," Engineering of Reconfigurable Systems & Algorithms (ERSA), July 2008.

[3] Gonzalez, I.; Gomez-Arribas, F.J.; Lopez-Buedo, S., "Hardware-accelerated SSH on self-reconfigurable systems," Field-Programmable Technology (FPT), 2005 IEEE International Conference on, Dec. 2005.

[4] Jara-Berrocal, A.; Gordon-Ross, A., "VAPRES: A Virtual Architecture for Partially Reconfigurable Embedded Systems," Design, Automation & Test in Europe Conference & Exhibition (DATE), March 2010.

[5] McDonald, E.J., "Runtime FPGA Partial Reconfiguration," Aerospace Conference, 2008 IEEE, March 2008.

[6] Montone, A., Santambrogio, M., Sciuto, D., Memik, S., "Placement and Floorplanning in dynamically reconfigurable FPGAs," ACM Transactions on Reconfigurable Technology and Systems, Nov 2010.

[7] Vipin, K.; Fahmy, S.A., "An Approach to a Fully Automated Partial Reconfiguration Design Flow," Field-Programmable Custom Computing Machines (FCCM), IEEE 21st Annual International Symposium April 2013.

[8] Wallace, G.K., "The JPEG still picture compression standard," Consumer Electronics, IEEE Transactions vol.38, Feb 1992.

[9] Xilinx Inc., "Partial Reconfiguration User Guide," UG702 (v14.5) April 26, 2013.

[10] Xilinx Inc., "Virtex-5 FPGA Configuration Guide", UG191 (v 3.11), October 19, 2012.

[11] Xilinx Inc., "Virtex-5 Family Overview", DS100 (v 5.0), Feb. 2009.

[12] Yousuf, S.; Gordon-Ross, A., "Partial Reconfiguration for Image Processing Applications," Military and Aerospace Programmable Logic Device (MAPLD), 2009.

[13] Yousuf, S., "Design Automation for Partially Reconfigurable FPGAs: Design Flows, Tools and Architectures", Ph.D. dissertation, Dept. Electrical and Computer Eng., University of Florida, Gainesville, FL, 2015