Formulation-level Design Space Exploration for Partially Reconfigurable FPGAs

Rohit Kumar, Ann Gordon-Ross
NSF Center for High-Performance Reconfigurable Computing (CHREC)
University of Florida, Gainesville, FL 32611 USA
{kumar, ann}@chrec.org

Abstract—Exploiting the benefits afforded by runtime partial reconfiguration (PR) on modern field-programmable gate arrays (FPGAs)requires PR-capable applications and associated PR-architectures, both of which are challenging tasks due to competing implementation metrics(e.g., area, power, operating frequency, etc.) and results in unmanageable design spaces. PR design space exploration (DSE) techniques and tools assist designers in efficiently and effectively exploring this design space. This paper presents the first, to the best of our knowledge, formulation-level PR DSE tool – FoRSE. FoRSE leverages the application's PR-architecture and mathematical FPGA device models and vendor-specified PR technology to generate Pareto-optimal sets of PR-floorplans and devices based on designer-designated implementation metrics. FoRSE can prune an application's implementation design space by three to four orders of magnitude in approximately 15 seconds.

Keywords-FPGA; partial reconfiguration; design space exploration; floorplanning; resource saving

I. INTRODUCTION AND MOTIVATION

Partial reconfiguration (PR)[11] affords several *PR-benefits* as compared to non-PR field-programmable gate arrays (FPGAs) such as reduced cost, resource usage, power consumption, and memory requirements[10] as well as improved system maintainability, adaptability[6], reliability, and fault-tolerance[7]. However, maximizing PR-benefits is non-trivial, requiring application designers to have extensive device- and application design-specific expertise [12] and an effective *PR-architecture* that decomposes an application's functionality into static and partially reconfigurable modules (PRMs), which execute in the FPGA fabric's static and partially reconfigurable regions (PRRs), respectively.

PR-architecture design and implementation is an unconventional and tedious process with specialized design and tool flows that combine several tools and manual steps [11][12]. Application designers create the PR-architecture layout by defining a *PR-floorplan* in accordance with vendor-specified *PR-constraints*. The PR-floorplan defines the PRRs' attributes–location, shape, and size, which must be large enough to accommodate all mapped PRMs–and PR-constraints restrict these attributes. Therefore, the application designer must carefully select a target device since the device's unique resource architecture directly affects the PR-floorplan options. Since the number of device choices combined with the number of PRR attribute combinations for each device results in a large PR-floorplan design space, PR-floorplan selection is non-trivial and arduous.

Application designers evaluate PR-floorplan's using several *implementation metrics. Internal utilization* is the ratio of a PRR's mapped PRMs' resource requirements as compared to the PRR's available resources. *External utilization* is the ratio of the entire PR-architecture's resource requirements as compared to the device's available resources. *Expected* and *actual resource savings* are the percentage resource savings that the application designer expects to gain and actually achieves, respectively, by using a PR-architecture as compared to a non-PR-architecture. The *PR-overhead*, which is imposed by fixed PR-constraints on the PR-floorplan, is the difference between these savings and can only be reduced by modifying the PR-architecture. Additional implementation metrics include *bitstream size* and *operating frequency*.

Since these implementation metrics compete with each other, there is no single, optimal PR-floorplan on an optimal device. For example, devices and PR-architecture combinations with high external utilization may have low operating frequencies due to routing difficulties in dense designs. PR design space exploration (DSE) creates sets of Pareto-optimal PR-floorplans and devices trading off each pair of designer-designated competing implementation metrics (only implementation metrics of interest need to be evaluated).

Implementation-level DSE leverages vendor-supplied implementation tools to physically implement each PR-floorplan in the design space and executes these physical implementations to gather implementation metric values to create the Pareto-optimal sets. Since implementation-level DSE requires a specialized tool flow and long tool execution times [2][10][12], implementation-level DSE is impractical and requires significant design-time effort.

Formulation-level DSE significantly reduces design-time effort by performing design space pruning at a higher level of abstraction and leveraging mathematical models, which eliminates lengthy design implementation steps. Formulation-level DSE is feasible since the device resource architecture information is vendor-supplied and the PRR attributes and the PR-floorplans can be extrapolated. This extrapolation leverages the device resource architecture, the application's PR-architecture, and resource usage estimates of the static module and PRMs that can be obtained using resource estimation tools [9]. Formulation-level DSE evaluates

978-1-4577-1740-6/11/$26.00 ©2011 IEEE

implementation metrics using mathematical models of the PR-architecture's *implementation information*, which includes the device resource architecture, PRR attributes, PR-constraints, and implementation metrics.

However, the tradeoffs for this reduced design-time effort are inaccuracies in some implementation metric estimations, such as the operating frequency and bitstream size due to these metrics' high dependency on vendor tools and the underlying FPGA's technology. Since formulation-level DSE does not physically implement PR-floorplans, formulation-level DSE can only consider the area-centric implementation metrics (internal/external utilization, expected/actual resource savings, and PR-overhead). Additionally, due to the specialized tool flow, disparate device resource architectures, and the variation in area metrics with respect to PRR attributes [4], incorporating implementation information at the formulation level is challenging.

In this paper, we present a **Fo**rmulation-level Partial **R**econfiguration Design **S**pace **E**xploration tool– FoRSE – that is the first, to the best of our knowledge, methodology that determines the Pareto-optimal sets of PR-floorplans and devices for an application's PR-architecture. To facilitate PR-floorplanning and device selection, FoRSE provides accurate mathematical models of the PR-architecture's implementation information. Since FoRSE leverages formulation and mathematical modeling instead of physical implementation, FoRSE requires little designer effort and affords fast DSE time. We show that FoRSE can prune the PR-floorplan design space by three to four orders of magnitude and determines the Pareto-optimal sets of PR-floorplans and devices in less than one minute.

The remainder of this paper is organized as follows. In Section II, we present previous works in implementation-level DSE effort reduction. Section III describes FoRSE's design space pruning algorithm. In Section IV, we present a case study to demonstrate FoRSE's fast DSE capabilities. Finally, Section V concludes our work and discusses future work.

II. Related Work

To reduce implementation-level DSE design-time effort, several previous works [1][5][7][8] focused on reducing the implementation tools' execution time for generic FPGA designs. Other works [6][12] focused on fully or partially automating the specialized tool flow. Since to the best of our knowledge, no previous work leveraged formulation-level DSE to reduce the design-time effort, we present the most relevant previous works in implementation tool execution time reduction and implementation tool flow automation.

To reduce implementation tool execution time, Coole et al. [5] presented intermediate fabrics (IFs). An IF is a highly specializable, virtualized FPGA fabric that provides a ~550x reduction in implementation tool execution time with a 10% and 18% average resource and frequency overhead, respectively. The authors stated that an application can have several optimal IFs and that an IF can be implemented on many possible FPGA devices, however, the authors did not provide a method to select a Pareto-optimal IF or device. Additionally, IFs do not support PR since there is no mechanism for creating PRRs inside the virtual FPGA fabric.

In the area of implementation tool flow automation, Craven et al. [6] presented a high-level development environment for implementing PR-architectures and used a simulated annealing-based automated PR-floorplanner. The authors also presented a helper tool to alleviate manual steps, but the authors did not provide details on the helper tool's mechanism or effectiveness. Carver et al. [3] developed an automated simulated annealing-based helper tool and evaluated their tool using timing results generated by the vendor's tool. However, since the PR-floorplan was fixed, their work cannot be leveraged for DSE.

Yousuf et al. [12] presented DAPR, which completely automated the implementation tool flow for implementation-level DSE. DAPR used a simulated annealing-based PR-floorplanner to generate Pareto-optimal PR-floorplans that traded off operating frequency and partial bitstream size. However, even though the flow was automated, multiple executions of the implementation tools required lengthy design-time effort. Additionally, DAPR considered only a single device and did not consider area metrics.

III. Formulation-level Partial Reconfiguration Design Space Exploration–FoRSE

FoRSE alleviates PR DSE challenges and reduces design-time effort using three DSE phases: the DSE setup phase, the DSE formalization phase, and the DSE execution phase. The DSE setup phase models an application's PR-architecture and the PR-architecture's implementation information using set-theory nomenclature. The DSE formalization phase leverages the DSE setup phase's models to formalize PR-floorplanning definitions and restrictions and establishes a formal problem statement. The DSE execution phase applies a design space pruning algorithm to the DSE formulation phase's problem statement to generate the Pareto-optimal sets of PR-floorplans and devices trading off designer-designated pairs of competing area metrics.

A. DSE Setup Phase

The DSE setup phase formulates and models an application's PR-architecture using set-theory representation for both basic and advanced formulation setup. The basic setup models the devices and application formulation details, such as the devices' resource types and amount of each resource type, the application or application's component resource requirements, the modules to represent the application or application's components, and rectangular device regions where modules can be located. The advanced setup leverages the definitions established in the basic setup to model the PR-architecture's implementation information.

1) Basic Setup: The basic setup establishes the following definitions and lemmas:

Definition 1: *A resource requirement* $RR = (RR_L, RR_B, RR_D)$, *where* RR_L, RR_B, *and* $RR_D \in \mathbb{N}$ *and represent the number of CLBs, BRAMs, and DSPs, respectively, is a 3-tuple representing the amount of each resource type available in a device region, an application, or an application's component.*

Definition 2: *A module M depicts an application or an application's component with resource requirements* $RR(M) = (RR_L, RR_B, RR_D)$. *Modules can either be static modules or PRMs.*

Definition 3: *A device region G is a rectangular device region spanning 'R' rows and 'C' columns starting at row origin 'm' and column origin 'n'. G is represented as a matrix with six properties: dR depicts row dimension 'R' (i.e.,* $dR(G) = R$*), dC depicts column dimension 'C' (i.e.,* $dC(G) = C$*), oR depicts row origin 'm' (i.e.,* $oR(G) = m$*), oC depicts column origin 'n' (i.e.,* $oC(G) = n$*), G[i][j] depicts the resource type at the* i^{th} *row and* j^{th} *column of G (i.e.,* $G[i][j] = \{rs\}$*), and RR depicts G's resource requirements(i.e.,* $RR(G) = (rr_L, rr_B, rr_D)$ *where* rr_L*,* rr_B*, and* rr_D *are the number of CLBs, BRAMs, and DSPs, respectively, and* $rs \in RS$*).*

Lemma 1: *A module M fits into a region G if* $RR(M) \leq RR(G)$ *(i.e.,* $RR_L(M) \leq RR_L(G)$*,* $RR_B(M) \leq RR_B(G)$*, and* $RR_D(M) \leq RR_D(G)$*).*

Lemma 2: *Two regions* G_1 *and* G_2 *are different if there exists two sets* $S_1 = \{(x,y) \mid oR(G_1) \leq x \leq (oR(G_1) + dR(G_1))$ *and* $oC(G_1) \leq y \leq (oC(G_1) + dC(G_1))\}$ *and* $S_2 = \{(x,y) \mid oR(G_2) \leq x \leq (oR(G_2)+dR(G_2))$ *and* $oC(G_2) \leq y \leq (oC(G_2) + dC(G_2))\}$ *such that* $S_1 \cap S_2 = NULL$*.*

2) Advanced Setup: The advanced setup establishes the following definitions and lemmas:

Definition 4: *An application's PR-architecture PA =* $(SM, \{RM_i\})$ *is a tuple of modules where SM represents the static module and* $RM_i = \{m_{i,j}\}$ *represents the set of PRMs*$(m_{i,j})$*,*$i \leq |RM|$*,* $j \leq |RM_j|$*, and* $i, j \in Z^+$*.*

Definition 5: *A device's resource architecture DA is a region G where* $oR(DA) = oC(DA) = 0$ *and* $dR(DA)$*,* $dC(DA)$*,* $RR(DA)$*, and* $G[i][j]$ *(for all* $i \leq dR(DA)$*,* $j \leq dC(DA)$*, and* $i, j \in Z^+$*) are vendor-specified (e.g., Xilinx) for each device.*

Xilinx provides device resource architecture information in *.nph* (native package header) files, however the files' format is unknown and human unreadable. Xilinx's *xdl–report* tool converts an *.nph* file to a text file containing all device resource architecture information in Xilinx Description Language (XDL). Since Xilinx provides no XDL documentation, we use in-house Python scripts to automatically parse the DA from the XDL file by discarding unnecessary low-level details (e.g., clock and I/O resources).

Some Xilinx FPGAs contain on-chip processors that span several rows and columns and distribute the device resources in a column-wise format, which results in non-uniformity in the device resource architecture model's column values. Since PRRs cannot contain on-chip processors [11], without loss of generality we can assume that the explored devices do not contain on-chip processors and incorporate this assumption by discarding all rows and columns spanned by on-chip processors in our design space pruning algorithm. Additionally, due to the resource types' column-wise distribution, resources contained in a region *G* do not vary with *G*'s row origin. We define this invariance in resources contained in two regions in the following lemma:

Lemma 3: *If* G_i *and* G_j *are two device regions such that* $dR(G_i)=dR(G_j)$*,* $dC(G_i)=dC(G_j)$*,* $oC(G_i)=oC(G_j)$*, and* $G[m][p] = G[n][p]$ *for any G where* $p \leq dC(G_i)$*,* $m, n \leq dR(G_i)$*, and* $p, m, n \in Z^+$ *then* $RR(G_i) = RR(G_j)$*.*

∴

Definition 6: *A region G is a PRR iff G satisfies vendor-specified PR-constraints, such as restrictions on G's row dimension and location of resource types:*

$$dR(G) = i * R^{prc} \quad (1)$$

$$G[any][1] = G[any][dC(G)] = L \quad (2)$$

where $i \in Z^+$ *and* R^{prc} *is a vendor-specified fixed positive integer representing the minimum PRR row dimension. Together,* (1) *and* (2)*constitute the PR-constraint model and are fixed for an FPGA family.*

B. DSE Formalization Phase

FoRSE leverages *PA*, *DA*, and the PR-constraint model from the DSE setup phase to characterize PR-floorplanning in the DSE formalization phase. Since PR-floorplanning is specialized and complex, PR DSE requires an accurate representation of PR-floorplanning to ensure accurate DSE results. The DSE formalization phase determines an accurate PR-floorplanning model by incorporating device- and vendor-specific constraints using the following definitions:

Definition 7: *PR-floorplanning for a formulated application takes as input a PR-architecture* (Definition 4) *and a device resource architecture* (Definition 5) *and determines all bijective mappings F from* $RM=\{RM_i\}$ *to* $G_{prr}=\{G_i\}$ *such that:*

$$F: RM \rightarrow Gprr$$

$$\max_j(RR(M_{i,j})) \leq RR(G_i) \quad (3)$$

$$\sum_i^{|RM|} RR(F(RM_i)) + RR(SM) \leq RR(DA) \quad (4)$$

$$\max_i(dR(F(RM_i))) \leq dR(DA) \quad (5)$$

$$\max_i(dR(F(RM_i))) \leq dC(DA) \quad (6)$$

where G_i *is a PRR based on* (1) *and* (2)*, and all* G_i*s are different* (Lemma 1)*. Equation* (3) *restricts all PRMs to fit into the PRMs' associated PRRs and* (4) - (6) *ensure that the PR-floorplan F fits on the selected device DA.*

∴

In addition to PR-constraints, vendors provide PR- and tool-specific recommendations, such as requiring a PRR's height to be less than the full device height (PRRs spanning the entire device's height may produce unknown behavior for static routes, which are routes not contained within PRMs) and requiring the PR-floorplan implementation to contain more CLBs than the estimated number of CLBs in the PR-architecture [4]. These recommendations are incorporated into the DSE formalization phase by including two designer-tunable parameters *maxH* and L_{adj} such that:

$$dR(G_i) \leq maxH * dR(DA) \quad (7)$$

$$RR_{adj} = \left((1 + L_{adj}) * RR_L,\ RR_B,\ RR_D\right) \quad (8)$$

where G_i is a PRR. Increasing *maxH* and L_{adj} increases the PR design space because PRR row and/or column dimensions increase with *maxH* and L_{adj}. Since we selected the Xilinx Virtex-4 family for our experiments and Virtex-4 device sizes vary, we varied *maxH* and L_{adj} to determine appropriate values that yielded FoRSE results reflecting all of the area metric tradeoffs generated in the DSE execution phase and fixed

maxH and L_{adj} at ¾ and 5%, respectively.

The following equations define area metrics for PR-floorplan evaluation:

$$I_u = \frac{RR(M_{i,j})}{RR(G_i)} \quad (9)$$

$$E_u = \frac{\sum_i^{|RM|} RR(F(RM_i)) + RR(SM)}{RR(DA)} \quad (10)$$

$$E_s = \frac{\sum_i^{|RM|}\sum_j^{|RM_i|} RR(M_{i,j}) - \sum_i^{|RM|} max_j\left(RR(M_{i,j})\right)}{\sum_i^{|RM|}\sum_j^{|RM_i|} RR(M_{i,j}) + RR(SM)} \quad (11)$$

$$A_s = \frac{\sum_i^{|RM|}\sum_j^{|RM_i|} RR(M_{i,j}) - \sum_i^{|RM|} RR(F(RM_i))}{\sum_i^{|RM|}\sum_j^{|RM_i|} RR(M_{i,j}) + RR(SM)} \quad (12)$$

$$PR_{Ov} = E_s - A_s \quad (13)$$

where I_u is the internal utilization, E_u is the external utilization, E_s is the expected resource savings, A_s is the actual resource savings, and PR_{ov} is the PR-overhead.

C. DSE Execution Phase

The DSE execution phase determines the Pareto-optimal sets of PR-floorplans and devices using a design space pruning algorithm. The pruning algorithm leverages three functions: *Prepare*, *Probe*, and *Analyze*. *Prepare* takes as input the PR-architecture and device resource architectures and produces a set of potential PRRs. A potential PRR is a device region that is large enough to fit (Lemma 1) each PRM mapped to that PRR (Definition 7), but the PRR may not satisfy (1) and (2). *Probe* takes as input the set of potential PRRs and applies the DSE formalization phase to determine all potential PR-floorplans for a device. Finally, *Analyze* iterates through *Prepare* and *Probe* for each device to generate the Pareto-optimal sets of PR-floorplans and devices for each designer-designated area metric pair.

1) Prepare: Prepare generates a set of potential regions ($g_{i,j}$) for each PRR (POT_i)based on the PR-architecture (*PA*) and the device resource architecture (*DA*). Fig.1 depicts *Prepare*'s algorithm. Since the distribution of resource types on the device does not change with respect to row location (Lemma 3), lines 1 and 2 determine the column distribution of the resource types by creating a string representation of the resources available in an arbitrary row (row 1 in this algorithm)

PREPARE input: PR-architecture (*PA*), Device resource architecture (*DA*)
PREPARE output: Set of attributes for potential PRRs without applying PRR constraints, $POT = \{POT_i\}$, $POT_i = \{POT_{i,j}\}$

1. $R_{da} = dR(DA)$, $C_{da} = dC(DA)$, fetch N_L, N_B, N_D from *DA*
2. *rowResString* = *make_string*(*DA*[*1*][:])
3. L_{da} = *rowResString.count*('*CLB*'), B_{da} = *rowResString.count*('*BRAM*'), D_{da} = *rowResString.count*('*DSP*')
4. *for i = 1 to |RM|*
5. $RL_{max} = N_L * max(RR_L(m_{i,j}))$, $RB_{max} = N_B * max(RR_B(m_{i,j}))$, $RD_{max} = N_D * max(RR_D(m_{i,j}))$ for all $1 \le j \le |RM_i|$
6. $TL_i = \{(r,c) \mid (1+L_{adj}) * RL_{max} \le r*c,\ r \le R_{da},\ c \le L_{da}$, and $r,c \in Z^+\}$
7. $TB_i = \{(r,c) \mid RB_{max} \le r*c,\ r \le R_{da},\ c \le L_{da}$, and $r,c \in Z^+\}$
8. $TD_i = \{(r,c) \mid RD_{max} \le r*c,\ r \le R_{da},\ c \le L_{da}$, and $r,c \in Z^+\}$
9. $POT_i = \{(r,c) \mid r = max(dR(TL_{i,j}), dR(TB_{i,k}), dR(TD_{i,m}))$ and $c = dC(TL_{i,j}) + dC(Tb_{i,k}) + dC(TD_{i,m})\}$
10. Go to line *4*

Fig.1. DSE execution phase's prepare function's algorithm to generate sets of potential PRRs

PROBE input: *POT*, *DA*, R_{da}, C_{da}, *rowResString*, RL_{max}, RB_{max}, RD_{max}
PROBE output: All feasible PR-floorplans (*F*)

1. *for i – 1 to |POT|*
2. *for j = 1 to* $|POT_i|$
3. *if* $dR(POT_{i,j})$ == *integer multiple of*(R_{pre})
4. *No: go to 2*
5. *Yes: for k 1 to* {*len*(*rowResString*) - $dC(POT_{i,j})$}
6. *for m –* {$k + dC(POT_{i,j})$} *to len*(*rowResString*)
7. L_c = *rowResString.count*('*CLB*', *k*, *m*)*$dR(POT_{i,j})$
8. B_c = *rowResString.count*('*BRAM*', *k*, *m*)*$dR(POT_{i,j})$
9. D_c – *rowResString.count*('*DSP*', *k*, *m*)*$dR(POT_{i,j})$
10. *if* $L_c \ge (1+L_{adj})*RL_{max}$ and $B_c \ge RB_{max}$ and $D_c \ge RD_{max}$ and *rowResString*[*k*] == '*CLB*' and *rowResString*[*m*]== '*CLB*'
11. Found a valid region *g* for *i*th PRR with $oR(g)=1$, $oC(g)=k$, $dR(g) = dR(POT_{i,j})$, and $dC(g) = dC(POT_{i,j})$, append *g* to G_i
12. *for* loop 6, 5, 2, 1 ends
13. Select a combination *Gprr* of one region $g_{i,j}$ from each G_i
14. $Gprr = (g_{0,j0}, g_{1,j1}, g_{2,j2}, g_{3,j3}, \ldots, g_{n,jn})$ where $1 \le jn \le |G_n|$ *and* $1 \le n \le |G|$
15. *if* $RR_{sl}(SM) + \sum_i RR(g_{i,jn}) \le RR(DA)$ *and* mutually *different*(G_{prr})
16. Feasible floorplan found append G_{prr} in F_i
17. Go to *12* unless all combinations have been checked

Fig.2. DSE execution phase's probe function's algorithm to generate sets of PR-floorplans

of the device (e.g., *rowResString* = *LLBLLDLL*). Line 3 determines the amount of each resource type (L_{da}, B_{da}, and D_{da}) in the resource string. Lines 4 and 5 select a set RM_i of PRMs and determine the maximum required amount of each resource type (RL_{max}, RB_{max}, and RD_{max}) across all PRMs in RM_i. Lines 6 through 9 use this information to determine the shapes of all of the potential PRRs for RM_i. Line 10 iterates over lines 4 through 9 for all RM_i to generate sets of POT_i of all potential PRRs for all PRMs.

2) Probe: Probe generates a set of PR-floorplans *F* for the device using a set of potential PRRs (*POT*) and the device resource architecture (*DA*). Fig.2 depicts *Probe*'s algorithm. Lines 1 through 3 select a potential region $POT_{i,j}$ that satisfies (1). Lines 5 through 12 determine $POT_{i,j}$'s row origin on the device's *DA*, verifies that all PRMs mapped to $POT_{i,j}$ can fit into $POT_{i,j}$, and verifies that $POT_{i,j}$ satisfies the PR-constraints in (2). Lines 5 through 12 output a set of regions G_i for each PRR that satisfies all PR-constraints. Lines 13 and 14 select a combination *Gprr* of PRRs by selecting exactly one PRR at a time from each G_i. *Gprr* is stored as a PR-floorplan *F* if *Gprr* fits on the device (line 15) and each region in *Gprr* is different (Lemma 2). Since a region's resources can only be used by the modules targeted for that region, Lemma 2 ensures that no two regions with different modules overlap and share resources. Line 17 iterates over lines 13 through 16 to determine all potential PR-floorplans for the device.

3) Analyze: Analyze evaluates each designer-designated area metric pair for each PR-floorplan using (9) through (13) and iteratively executes *Prepare* and *Probe* for each device to generate Pareto-optimal sets of PR-floorplans and devices. Since area metrics are 3-tuples, comparing area metrics requires designer-defined comparison functions, such as:

$$RR_1 = RR_2 if L_1 = L_2, B_1 = B_2, and D_1 = D_2 \quad (14)$$

$$RR_1 > RR_2 if L_1 > L_2 \ or B_1 > B_2 or D_1 > D_2 \quad (15)$$

which compare two arbitrary 3-tuples $RR_1 = (L_1, B_1, D_1)$ and $RR_2 = (L_2, B_2, D_2)$.

TABLE 1
SAMPLE PR-ARCHITECTURE (*PA*). *PA* CONTAINS FIVE PRMS: $M_{1,1}$, $M_{1,2}$, $M_{1,3}$, $M_{2,1}$, AND $M_{2,2}$; AND TWO PRRS: RM_1 AND RM_2. PRMS $M_{1,1}$, $M_{1,2}$, AND $M_{1,3}$ ARE MAPPED TO PRR RM_1. PRMS $M_{2,1}$ AND $M_{2,2}$ ARE MAPPED TO RM_2.

Module (M)	PRR (RM_i)	CLBs ($RR_L(M)$)	BRAMs ($RR_B(M)$)	DSPs ($RR_D(M)$)
SM (static)	n/a	1365	36	0
$M_{1,1}$; $M_{1,2}$; $M_{1,3}$	RM_1	277; 224; 195	8; 4; 4	0; 2; 2
$M_{2,1}$; $M_{2,2}$	RM_2	611; 772	16; 16	8; 8

IV. EXPERIMENT AND RESULTS

We evaluated FoRSE's design space pruning abilities using a sample PR-architecture DSE case study with several different devices, which represents an example of how a designer would leverage FoRSE during application design.

A. Experimental Setup

We evaluated FoRSE's Pareto-optimal device selection using the Xilinx Virtex-4 FPGA family, which contains seventeen devices of varying row and column sizes categorized into three device series: LX, SX, and FX. The LX and SX devices predominately contain CLBs and DSPs, respectively, and the FX devices contain equal amounts of CLBs, BRAMs, and DSPs. The size and resource diversity across these devices affords a huge design space and requires FoRSE to evaluate a wide range of potential PRRs and PR-floorplans. The design space consists of all PR-floorplans on all devices of the selected family that satisfy (1)-(6) but may not satisfy (7) and (8) and results in ~10^7 potential designs.

We implemented FoRSE in Python for cross-operating system portability and generated the Xilinx family-specific device resource architecture libraries using Python's scientific library *NumPy* and the '*xdl –report*' tool in Xilinx ISE v13.1.

Since FoRSE's design space pruning is independent of the application's functionality and run-time behavior and is only dependent on the formulated PR-architecture, we performed a FoRSE design space pruning case study using Conger et al.'s sample PR architecture [4]. The diversity in resource requirements across the static module, PRMs, and resource types makes this sample PR-architecture an ideal choice for our case study. Table 1 depicts Conger et al.'s sample PR-architecture *PA*, which contains five PRMs: $M_{1,1}$, $M_{1,2}$, $M_{1,3}$, $M_{2,1}$, and $M_{2,2}$ and two PRRs: RM_1 and RM_2. PRMs $M_{1,1}$, $M_{1,2}$, and $M_{1,3}$ are mapped to PRR RM_1 and PRMs $M_{2,1}$ and $M_{2,2}$ are mapped to RM_2. The first column lists the static module (SM) and the PRMs; the second column lists the PRRs associated with the PRMs; and the third through fifth columns list the resource requirements for the PRMs.

The resource requirements for each PRR in Table 1 is calculated with $RR(RM_i) = max_i(RR(M_{i,j}))$, which results in resource 3-tuples (277, 8, 2) and (772, 16, 8) for RM_1 and RM_2, respectively.RM_2's resource requirements are significantly larger than RM_1, therefore RM_2's PRR will be larger than RM_1but RM_1will have more Pareto-optimal PRRs than RM_2. Additionally, since larger PRRs increase a PR-floorplan's E_u(10) and decrease a PR-floorplan's A_s(12) and a higher number of Pareto-optimal PRRs can enable fine tuning

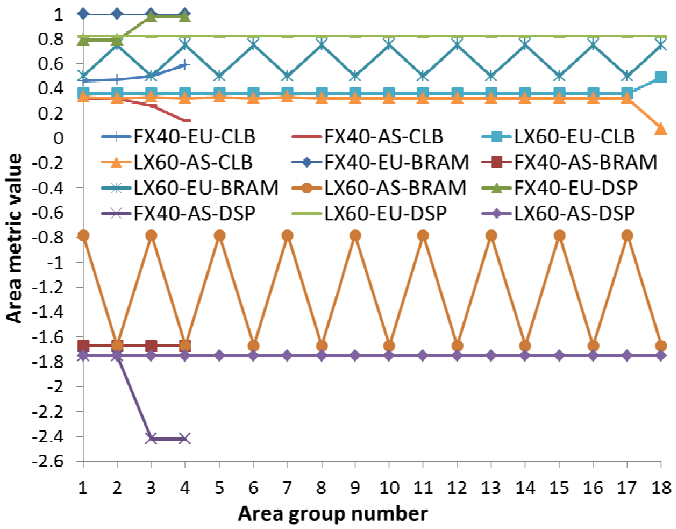

Fig.3 E_u(EU) andA_s(AS) area metric pair values for CLBs, BRAMs, and DSPs for the FX40 andLX60 devices with respect to area group number

of E_u and A_s, we select E_u and A_s as the competing area metric pair. We evaluated all devices in each device series based on this area metric pair to determine the Pareto-optimal set of PR-floorplans and devices. We note that a designer may choose several area metric pairs and FoRSE would generate a Pareto-optimal set of PR-floorplans and devices for each pair.

B. Results and Analysis

To determine the Pareto-optimal set of devices, we determined all devices that can fit at least one PR-floorplan. Our analysis revealed that out of the seventeen devices, eight devices were large enough to fit at least one PR-floorplan. FoRSE considered these eight devices and all PR-floorplans that fit on these devices for design space pruning.

FoRSE pruned the PR-floorplan design space by 98.8%, leaving 1,786 potential designs, in approximately 15 seconds(measured with the bash shell *date +%T* command) on a Ubuntu 32-bit PC with 3GB RAM and 2.7GHz Intel core 2 duo processor. Since several PR-floorplans have the same area metric values for each resource type, we divided the PR-floorplans into per-device area groups, where PR-floorplans within the same area group have the same area metric values. Selecting the best PR-floorplan and device based on the application designer's application implementation goals requires evaluating and comparing all PR-floorplans, but such an exhaustive comparison even with the pruned design space is infeasible. Therefore, we compare two devices, the V4FX40 and V4LX60, with the lowest number of area groups. The V4FX40 had 258 Pareto-optimal PR-floorplans divided into 4 area groups and the V4LX60 had 62 Pareto-optimal PR-floorplans divided into 18 area groups.

Fig.3 depicts the E_u and A_s area metric pair values for CLBs, BRAMs, and DSPs for the V4FX40 and V4LX60 deviceswith respect to the area group number. The graphs contain six plots for each device, area metric, and resource type combination and the plots are denoted by the device name prefixed with EU for E_u and AS for A_s and resource type. Since A_s represents the difference between the resource requirements for the non-PR-architecture and the application's

PR-floorplan, a negative A_s value for a resource type means that the PR-floorplan requires more of that resource type than the non-PR architecture.

The results reveal that not all area groups contain enough resources for the PR-floorplans. For example, all area groups for both devices have negative BRAM and DSP $A_s$values. These designs are important to designers because these designs indicate a re-architecting potential. We point out that a designer could make these A_s values positive by modifying the PR-architecture (i.e., changing the application partitioning) such that the new modified PR-architecture requires fewer BRAMs and DSPs.

When comparing the devices' area groups for PR-floorplan and device combinations, V4FX40's area groups 1 and 2 provide better or equal CLB and BRAM external utilization (E_u) and actual area savings (A_s) with up to 2.14 times lower DSP actual area savings as compared to all of the V4LX60's area groups. Therefore, a designer can select either area groups 1 or 2 on the V4FX40 or any area group on the V4LX60 for implementation-level DSE based on whether the designer's application implementation goals require moderate CLB and BRAM external utilization and actual area savings or high DSP actual area savings. Alternatively, V4FX40's area groups 3 and 4 show high CLB external utilization and 100% BRAM and DSP external utilization but show lower actual area savings for all resource types as compared to V4LX60's area groups 3 to 17. Therefore, a designer can select either area groups 3 or 4 on the V4FX40 or area groups 3 to 17 on the V4LX60 for implementation-level DSE based on whether the designer's application implementation goals require 100% BRAM and DSP external utilization or high actual area savings for all resource types.

Based on typical designer goals, the best PR design space for an application's PR-architecture likely contains designs with high external utilization and actual area savings in as many resource types as possible. Given our case study PR-architecture, a designer should select V4LX60's area groups 3, 5, and 7 because these area groups show the highest CLB and BRAM external utilization, and the highest CLB, BRAM, and DSP actual area savings.

V. Conclusions and Future Work

In this paper, we presented FoRSE – an application formulation-level design space exploration (DSE) tool for partially reconfigurable (PR) FPGA architectures. FoRSE leverages a partitioned application's static and reconfigurable modules' estimated resource requirements to formulate the application's PR-architecture. In addition, FoRSE provides mathematical models of vendor-specified FPGA device and tool information to rapidly prune the PR-architecture's floorplan and implementation design space. FoRSE compares the PR-architecture's PR-floorplans using implementation metrics, such as device utilization and area requirements, and outputs Pareto-optimal sets of PR-floorplans and devices that tradeoff competing designer-designated implementation metrics. A case study evaluation showed that FoRSE achieves orders of magnitude reduction in the design space and can generate Pareto-optimal sets of PR-floorplans and devices for a sample PR-architecture in less than one minute.

Future work includes exploring a formulation-level application partitioning methodology to automatically create PR-architectures in early application development stages and integrating FoRSE with existing implementation-level PR-DSE tools, such as IF [5] and DAPR [12], which would provide a one-click solution for complete PR DSE.

Acknowledgments

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.

References

[1] P. Athanas, J. Bowen, T. Dunham, C. Patterson, J. Rice, M. Shelburne, J. Suris, M. Bucciero, and J. Graf, "Wires on demand: Run-time communication synthesis for reconfigurable computing," in *Proceedings of the International Conference on Field Programmable Logic and Applications*(*FPL*), Aug. 2007, p. 513–516

[2] V. Betz. The FPGA place and route challenge. http://www.eecg.toronto.edu/~vaughn/challenge/challenge.html

[3] J.M. Carver, R.N. Pittman and A. Forin, "Automatic Bus Macro Placement for Partially Reconfigurable FPGA designs," in *Proceedings of the Seventeenth ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*(*FPGA*), Feb. 2009.

[4] C. Conger, A. D. George, and A. Gordon-Ross, "Design Framework for Partial Run-Time FPGA Reconfiguration," in *Proceedings of The International Conference on Engineering of Reconfigurable Systems and Algorithms* (*ERSA*), Jul. 2008

[5] J. Coole and G. Stitt, "Intermediate fabrics: virtual architectures for circuit portability and fast placement and routing" in *Proceedings of the eighth IEEE/ACM/IFIP International Conference on Hardware/software Co-design and System synthesis*(*CODES/ISSS*), Oct 2010, p. 13–22.

[6] S. Craven and P. Athanas, "Dynamic Hardware Development", in *International Journal of Reconfigurable Computing,* 2008, p. 1-10.

[7] A. Jacobs, A. George, and G. Cieslewski, "Reconfigurable Fault Tolerance: A Framework for Environmentally Adaptive Fault Mitigation in Space", *Proc. of Intl. Conference on Field-Programmable Logic and Applications (FPL)*, Prague, Czech Republic, Aug. 31 - Sep. 2, 2009

[8] C. Mulpuri and S. Hauck, "Runtime and quality tradeoffs in FPGA placement and routing," in *Proceedings of the 2001 ACM/SIGDA Ninth International Symposium on Field Programmable Gate Arrays* (*FPGA*), 2001, p. 29–36.

[9] C. Shi, J. Hwang, S.Mcmillan, A. Root, and V. Singh, "A system level resource estimation tool for FPGAs", in *Proceedings of the International Conference on Field Programmable Logic and its Applications* (*FPL*), 2004.

[10] NSF Center for High-Performance Reconfigurable Computing. FPGA tool-flow studies workshop hosted by CHREC. June 2008. http://www.chrec.org/ftsw/

[11] Xilinx Partial Reconfiguration Tools & Techniques http://www.xilinx.com/training/fpga/partial-reconfiguration-tools-and-techniques.htm

[12] S. Yousuf and A. Gordon-Ross, "DAPR: Design Automation for Partially Reconfigurable FPGAs", in *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms* (*ERSA*). 2010.