Design Space Exploration Acceleration through Operation Clustering

Benjamin Carrion Schafer *Member, IEEE* and Kazutoshi Wakabayashi *Member, IEEE*

***Abstract*—This paper presents a clustering method called Clustering Design Space Exploration (CDS-ExpA) to accelerate the architectural exploration of behavioral descriptions in C and SystemC. The trade offs between faster exploration vs. optimality of results are investigated. Two variations of CDS-ExpA were developed: CDS-ExpA(min) and CDS-ExpA(max). CDS-ExpA(min) builds the smallest possible clusters while CDS-ExpA(max) builds the largest possible ones reducing further the design space. Results show that CDS-ExpA(min) and CDS-ExpA(max) explore the design space 90% and 92% faster on average than a previously developed annealer based exploration method at the expense of not finding 36% and 47% of the Pareto optimal designs and finding the smallest design that is 7% and 9% on average larger and the fastest design 28% and 32% slower respectively.**

***Index Terms*—Design space exploration, High level synthesis, acceleration, clustering**

I. Introduction

Satisfying consumers' increasing appetite for latest technologies implies for hardware design engineers shorter design cycles. The key to shorten design cycles is to have tools that convert system level descriptions, in any high level language, into efficient hardware designs in an easy and fast way, giving designers as much information as possible about the system at the earliest possible design stage to avoid time consuming re-designs. Raising the level of abstraction has a distinct advantage over traditional RTL design approaches. Multiple designs can be easily and quickly generated for the same source code, while RTL designs require major rework in the source code in order to modify the architecture. Moreover higher levels of abstraction combined with high level synthesis allow the architectural trade-off exploration of the behavioral description. The main problem with architecture exploration is its exponential order of complexity with the number of explorable operation, making it impossible to perform a full search space exploration. In this work we present a clustering method to accelerate the design space exploration (DSE) and compare it with a previously presented exploration method based on simulated annealing [1] that leads to a good result finding Pareto optimal designs compared to a brute force approach for smaller benchmarks. Our new method is based on clustering operations, assigning a fix set of synthesis attributes (pragmas) to 'explorable' operations at the source code reducing the design space and therefore runtime.

The main objective in design space exploration is to resolve minimizing conflicting objectives by finding the optimal points for different objective functions. In this work we restrict the exploration objectives to area and latency, but other objectives like power or throughput could also be added. These optimal points are called Pareto points. The objective is to find all the designs at the efficient frontier (also called Pareto frontier) shown in Figure 1 a, where each point on the frontier is a Pareto optimal point. The tradeoffs can easily be explored within this set rather than considering the entire design space, which would be impractical and irrelevant to the designer.

The main problem in DSE is its exponential nature. A brute force approach would eventually find all the Pareto optimal points for smaller designs at a cost of extremely high running time. Heuristics have been developed to reduce runtime, at the expense of finding less number of Pareto points and some fictitious ones. Figure 1 b shows the eventual result of some heuristics. Region 1 shows fictitious Pareto points found by the heuristic. Further design space exploration would eventually end up finding the real Pareto point. Region 2 shows that some heuristics might miss Pareto optimal points and region 3 illustrates that some of the Pareto points on the efficient frontier are actually found. A big problem with multi-objective functions is how to prove Pareto optimality. In this work we can only proof Pareto optimality for small benchmarks as these where benchmarked against a brute force method. For larger designs this is virtually impossible and we can only create solutions that are non-dominated. We will consider these solutions Pareto optimal as there is no way to proof their optimality.

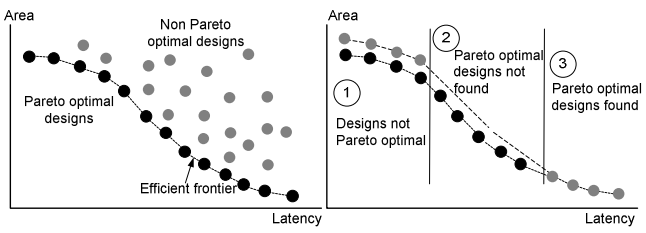

Figure 1 (a) efficient frontier with Pareto optimal designs overview (b) design space exploration result

Manuscript received June 3, 2009, revised August 11, 2009

Benjamin Carrion Schafer is with NEC Corporation Central Research Laboratories, EDA Center, Kawasaki, Kanagawa 211-8666 Japan (phone: +81 44 435 9486; fax: +81 44 435 9491; e-mail: schaferb@bq.jp.nec.com).

Kazutoshi Wakabyashi is with NEC Corporation Central Research Laboratories, Design Methodology Center, Kawasaki, Kanagawa 211-8666 Japan (e-mail: wakaba@bl.jp.nec.com).

Previous work tightly integrate the exploration and the High Level Synthesis (HLS) steps in order to deterministically estimate the effect of each explorable transformation (e.g. loop unrolling vs. none) and therefore targeting directly during the exploration the transformation that lead to Pareto optimal designs. In this work we address the architecture exploration considering the HLS as a black box using a commercial HLS tool [2]. This HLS tool performs different transformations and applies different heuristics depending on the input description which are unknown before hand. The contributions of this work can be summarized as follows:

- Introduce a clustering method called Clustering Design Space Exploration (CDS-ExpA) to accelerate the design space exploration for behavioral descriptions given in C or SystemC. CDS-ExpA clusters all explorable operations based on a given cluster library assigning fix synthesis attributes to each cluster.
- We investigate the impact of the cluster size on the exploration speed and on the number of Pareto optimal points found, compared to a previously introduced method based on simulated annealing [1]. Two clustering methods are developed for this purpose. CDS-ExpA(min) and CDS-ExpA(max), where the first builds the smallest possible clusters and the latter the largest possible.
- Present a comprehensive set of results comparing our clustering methods with a previously presented annealer based exploration method.

II. MOTIVATIONAL EXAMPLE

Figure 2 shows the source code of a small program that continuously reads in 8 bit numbers and calculates the average of the last 8 values read (this program corresponds to benchmark ave8 used in the experimental section). The explorable operations have been highlighted and consist of an array where the last 8 numbers are stored, 2 loops and 1 function. The table next to the source code shows the result of the HLS for different synthesis attributes specified directly at the highlighted explorable operations using pragmas. As seen the difference between the smallest but slowest design and the fastest but largest is substantial, ranging from 1362 to 4352 gates and latencies from 24 to 1 cycle. There are a multiple of Pareto optimal combinations in between these designs based on different attribute combinations as well as sub-attributes like the number of memory ports in the array (only 5 shown here). Manually editing the source code in order to explore the different area vs. performance trade-offs is a tedious and time consuming task. An automatic efficient design space exploration method is therefore highly desirable. The main problem in DSE is how to explore the design space in a reasonable time, finding as many Pareto optimal points as possible.

III. RELATED WORK

In order to deal with quicker time to market design cycles high level languages extended with hardware specific constructs are being used for designing hardware combined with high level synthesis. Some examples of C/C++ extensions include

```
process main(){
  in  ter(0:8) in0 ;
  out ter(0:8) out0 ;
  var(0:8) sum;
  var (0:8)[fifo[8] /* Cyber array=reg */{ 0, 0, 0, 0, 0, 0, 0, 0 } ;
  var(0:4) i ;

  /* Cyber unroll_times=0 */
    for( i = 7 ; i > 0 ; i-- )
      fifo[i] = fifo[i-1] ;

    fifo[0] = in0;

  /* Cyber unroll_times=0 */
    for(i=1;i<8;i++){

        /* Cyber func=inline */
          sum = addition(sum, fifo[i]);
  }
    out0 = sum / 8;
}
```

array	loop1	loop2	function	Area	Latency
ram	0	0	goto	1362	24
ram	all	all	inline	2684	19
reg	0	all	inline	2915	6
reg	0	0	inline	3544	5
reg	all	all	inline	4352	1

Figure 2 Ave8 source code example highlighting the explorable operations and summary of HLS results using different set of attributes

SystemC, BDL [2] or SA-C [3]. These high level language subsets simplify the design process as designers do not need to deal with low level Hardware Description Languages (HDLs). However designers still have to manually analyze the design to specify i.e. bit widths, parallelism, operator binding and resource sharing. The design space exploration does this step automatically generating a number of designs that meet a set of constraints (i.e area, latency and power). Much of the previous research has been focused on system level design exploration where the number and the type of functional units and bus size are explored [4]-[6]. We call this macro-architectural design space exploration vs. micro-architectural which is the method we present in this work. Previous work in the micro-architectural design space exploration on high level synthesis has been focused on applying source code transformation starting from CDFGs using multi-objective function optimizations. Ahmad et. al [7] studied the tradeoffs between the control step and area in data flow graphs using genetic algorithms. Holzer et. al [8] used a similar approach using an evolutionary multi-objective optimization approach to generate Pareto optimal solutions. Haubelt et al. [9] use Pareto-Front-Arithmetics (PFA) to reduce the search space in embedded systems decomposing a hierarchical search space. Early estimators of area and delay for FPGA implementations where used in [10] to evaluate the design space before any behavioral synthesis. Anderson et al [12] collect system information before the exploration starts doing a configuration sweep and use a genetic algorithm for the exploration of a parameterized RISC processor. A compiler approach to perform hardware design space exploration is presented in [11] where parallelization techniques are used to map computations to FPGAs. So et al. [13] developed a design space exploration technique using compiler direct techniques to perform several code transformations. The starting point in all of these approaches is the direct transformations at the CDFG level applying different compiler and optimizations techniques to generate new architectures combined with quick estimators. In this work we explore the design space using a commercial high level synthesis tool [2] seen as a black box by the explorer. We do not have access to it and execute it every time a new exploration design is generated. Previous work estimates the impact of each transformation as they have full control over the resultant synthesized circuit and can deterministically establish the cost of each transformation. The number of transformation

allowed in these cases is very limited and are normally restricted to the number and/or type of functional units (FUs) and in some cases to loop unrolling. The commercial HLS tool used in this study has over 500 different optimization options using attributes (pragmas for individual operations) and global synthesis options and behaves differently in each case based on the different heuristics applied. It makes it therefore impossible for the exploration tool to accurately model and predict the behavior of the HLS tool, which leads to the need to adopt a different approach.

IV. Design Exploration

The design space exploration method presented in this work is based on a clustering algorithm called Clustering Design Space Exploration (CDS-ExpA) for a simulated annealer explorer introduced in [1] called Adaptive Simulated Annealer Exploration Algorithm (ASA-ExpA). The ASA-ExpA method generates a set of Pareto optimal designs for a given design written in untimed C or SystemC by inserting HLS directives directly into the source code. These directives are in the form of pragmas that the HLS tool processes and in turn synthesizes the instrumented source code accordingly. The method presented in this work explores loops, arrays and functions. Table 1 shows all the explorable operations and their pragmas. A more comprehensive explorer could also explore global synthesis options and the number of functional units. The goal of the exploration is to find as many as possible points on the efficient frontier or as close as possible. The tool developed around the exploration methods is called cwbexplorer. Figure 3 shows an overview of the exploration flow. SystemC or C is read by cwbexplorer. A new unique set of attributes is generated for the explorable operations found in the source code. A set of global synthesis options, the newly instrumented source code (.IFF) and a functional unit constraint file are passed to the HLS tool which then synthesizes the new designs. The result of the synthesis is read back by cwbexplorer to analyze the synthesis results (area and latency). If the design is dominated it is deleted. The exploration continues until no more Pareto optimal designs are found or a given exit criterion is met e.g. N number of non-optimal designs are created consecutively.

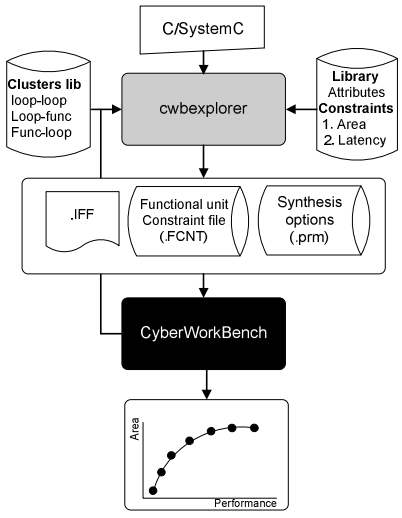

Figure 3 Exploration flow overview

Table 1 Attributes of explorable operations

Operation	Attribute	Description
Loop	Unroll=0	Do not unroll the loop
	Unroll=x	Partial loop unroll
	Unroll=all	Unroll the loop completely
	Folding=N	Fold loop N times
Functions	Func=inline	Inline each function call
	Func=goto	Single function instantiation
	Func=seq_opr	Sequential operator
	Func=pipeline_opr	Pipeline operator
	Func=operator	Function treated as a functional unit
Array	Array=RAM	Array synthesized as memory
	Array=logic	Constant array synth. as logic
	Array=expand	Expand array
	Array=reg	Synthesize array as registers

Two version of the CDS-Exp have been developed. The first, called CDS-ExpA(min) clusters a group of operations into operation clusters (*OC*) as small as possible. This approach reduces the design space compared to the ASA-ExpA method developed previously, while at the same time still allows a large number of cluster combinations combined with non-clustered operations that leads to a smaller possibility of missing Pareto optimal designs. The second approach called CDS-ExpA(max) builds the largest possible clusters reducing further the design space compared to the CDS-ExpA(min) approach and hence accelerating the design space exploration even further at the expense of missing more Pareto optimal designs and generating more non-Pareto optimal designs. In this work we will compare both methods with the original simulated annealer method.

The construction of the cluster is performed by firstly parsing the input C or SystemC code and building a dependency parsed tree of all the explorable operations (loops, functions and arrays). Figure 4 shows the parse tree of the motivational example (ave8). The parse tree is then traversed using a breadth first tree search method and a tree pattern matching algorithm applied to find clusters given in a previously manually created external cluster library. This ensures that clusters are built from top to bottom as transformation at higher levels of the tree have larger impact on the final synthesis results (e.g. when unrolling 2 nested loops, unrolling the outer loop has a bigger impact on the final synthesis). The cluster library contains the sequences of operations that form a cluster and the set of pre-defined attributes associated to each cluster for different optimization targets e.g. reduce area or latency. The cluster types are all possible 2-3-4-tuples combination of arrays, functions and loops.

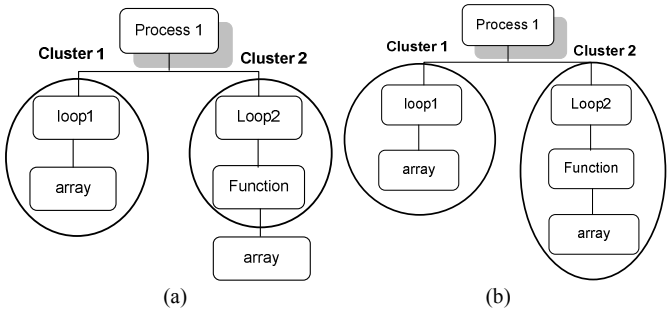

Figure 4 Parse tree and clustering methods (a) Min cluster CDS-Exp(min) (b) Max cluster CDS-Exp(max) for ave8

objective should be minimized. The GCF changes dynamically during the exploration and the attributes assigned to each operation in the each cluster are re-assigned to minimize the GCF objective. E.g. if the GCF targets area reduction the loop1-array cluster in Figure 4 a will have the attributes Cluster1=[loop1=0,array=ram] assigned. When the GCF is updated to minimize latency, the attributes associated to this cluster will change to Cluster1=[loop1=all, array=reg]. It has to be noted that functions and arrays can be called/accessed from within different clusters. In order to be consistent, the attributes applied to these shared operation have to be the same throughout all the clusters.

The fixed attributes assigned to each clusters have been empirically determined based on the study of the typical impact on each attribute on the synthesis on a set of different benchmarks. In order to avoid local minima a probabilistic component is inserted to the attribute assignment to each cluster, allowing clusters to be assigned attributes that do not minimize the GCF objective. The GCF has 3 states. Minimize area; minimize latency and an intermediate state. Each state has a unique set of attributes for each cluster stored in the external cluster library. These states target the exploration of Pareto points that minimize area, minimize latency and intermediate points in the curve. If after N designs no more Pareto designs are found the exploration moves to a new state updating the cost function and re-assigning the clusters' attributes.

Figure 5 summarizes the procedure of our clustering based exploration method:

Step1: Construction of Clusters: For the given untimed C or SystemC source code a dependency parse tree with the explorable operations (loops, functions and arrays) is built. Operation clusters are built by traversing the parse tree matching operations groups with clusters given in the external cluster library. The cluster size depends on the selected clustering method. CDS-ExpA(min) builds smallest possible clusters, while CDS-ExpA(max) builds the largest possible ones. Each cluster will be assigned a fix set of attributes specified in this library depending on the GCF state. The cluster attributes are modified when the global cost function state change during the exploration, where e.g the target is to generate Pareto optimal designs that minimize area changes to created designs that maximize performance and probabilistically based on the simulated annealers temperature (ASA-ExpA).

Step2: Creating Pareto optimal designs: Our method generates a unique new set of attributes for the operations given in the parse tree that do not belong to any cluster. The simulated annealer cost function is given by COST = αA + βL. The weighting factors α and β are adaptively updated during the exploration to represent the importance of minimizing the total area (A) or latency (L). This adaptive coefficient adjustment is made each time no more non-dominated designs could be generated for a given coefficient combination. Every time a new design is generated it is synthesized and checked for Pareto optimality. If it is not, it gets deleted. On the other hand each time a new Pareto optimal design is found, the rest of Pareto optimal designs are re-checked for optimality as this new design could render previous designs not Pareto optimal anymore. In this case these are deleted. Every time the GCF is updated the annealer is executed until the exit condition is reached.

CDS-ExpA: Clustering Design Space Exploration(*S, LC, LE, I*)
/* *S* : Source code (C or SystemC)
LC : Cluster library
LE: Explorable operations library
I: Input parameters (e.g. type of exploration) */
/* Step 1*/
- Parse source code *S* and build dependency parse tree of all explorable operations given in *LE.*
- Build exploration clusters by traversing the parse tree and apply tree matching algorithm with clusters given in library *LC.*

/* Step 2*/
- GCF = A =0, L=10 /* initialize global cost function to generate designs that minimize latency*/

while (GCF != L=0, A=10) /* explore until all GCF states passed */
while(annealer temp > X) **do** /* explore until annealer exit criteria reached*/
- Assign attributes to operation in clusters from *LC* library attributes based on GCF state
- Assign attributes to un-clustered operations from *LE* based also on the GCF state /* if L=10 the probability to assign a latency reduction attributes is larger than an area reduction attribute */
- Randomize cluster assignment /* probabilistic re-assignment of cluster attributes to escape local minima */
- If sequence of attributes is unique synthesize design
- Check if design is non-dominated. If not delete, if yes check if previous designs are still Pareto optimal

endwhile;
GCF= L-=delta, A+=delta /* update GCF to intermediate or minimize area state */
endwhile;
return Pareto optimal designs (*POD)*

Figure 5 Summary of the procedure of our exploration method (CDS-ExpA)

The most time consuming part in CDS-Exp is the inner while-loop which is bounded to $O(p^n)$, where p is the number of explorable attributes for each operation and n is the number of explorable operations. Although the order of complexity is exponential, clustering operations n will reduce the order of complexity to $O(p^m)$, where $m=n-(C\ S)$, C is the number of clusters and S the cluster size, so that m is a much smaller value than n reducing therefore the design space considerably, at the expense of missing Pareto optimal designs.

V. Experimental Results

First, we describe the experimental setup for the evaluation of out proposed method. Then, we show a set of comprehensive results obtained, together with the explanation and implication of the analysis of the data.

A. Experimental Setup

10 different benchmarks written in C and SystemC used in in-house designs were chosen to validate our method shown in Table 2. The first column shows the benchmark name. The second column indicates if it is a C or SystemC (SC) benchmark. The third column shows the size of the benchmarks denoted by the number of lines of code. It should be noted that 1 line of code of a high level language description is approximately equivalent to 10 lines of RTL code [15]. We compare our proposed method (CDS-ExpA(min) and CDS-ExpA(max)) to a previously presented simulated anealer based approach (ASA-ExpA) [1]. For each method the number of Pareto optimal designs found, the complete exploration runtime, the number of gates of the smallest design and the latency of the fastest design is given. As the anealer can take extremely long time to run, especially for the larger benchmarks it was decided

TABLE 2 EXPERIMENTAL RESULTS

			ASA-ExpA (Annealer)				CDS-ExpA(min)					CDS-ExpA(max)				
Bench	Type	# line	# Pareto	Run [s]	Min gates	Min Latency	# Clust.	# Pareto	Run [s]	Min gates	Min Latency	# clust	# Pareto	Run [s]	Min gates	Min Latency
ave8	C	44	4	17872	1271	1	2	3	1507	1362	1	2	3	1362	1362	1
Mult_Inst	C	55	4	2793	1024	3	2	4	346	1263	6	2	2	199	1263	6
Seql_comb	C	60	5	643	1387	3	2	3	36	1503	3	2	2	19	1503	3
add_func	C	69	7	13223	1692	6	3	3	194	2041	5	2	2	908	2041	7
combi_mult	SC	93	9	16877	1002	3	3	4	254	1233	3	2	4	254	1233	5
neststruct	C	116	5	6400	2338	6	3	4	413	2338	7	2	2	205	2338	6
adpcm	SC	198	7	3531	5121	5	4	5	155	5121	6	2	3	151	5121	7
gfilter	C	270	6	38283	9681	12	5	3	10384	9868	10	3	2	8123	9866	14
rsa_core	SC	501	9	58941	54874	12	26	5	15784	55711	13	23	13	10974	58711	14
SwitchFabric	C	584	12	43962	39990	20	15	7	3134	40365	22	12	4	3134	46365	23
Avg.		**199**	**6.8**	**20253**	**11838**	**7**	**6**	**4.1**	**3221**	**12081**	**8**	**5**	**3.7**	**2533**	**12980**	**9**
Δ Avg. methods [%]								**36%**	**90%**	**7%**	**28%**		**47%**	**92%**	**9%**	**32%**

to exit the exploration if after 100 newly generated designs no new Pareto optimal design is found. We make the assumption that the Pareto design found by the ASA-ExpA approach are actual Pareto optimal and not fictitious. As we showed in [1], this holds true for the smaller benchmarks used in this study compared to a brute force approach. As the CDS-ExpA method might create fictitious Pareto optimal designs, the generated designs are compared against the ones generated by ASA-ExpA and only the designs that match the Pareto designs generated by ASA-ExpA are considered real Pareto optimal designs. The experiments were run on an Intel Xeon running at 3.20GHz machine with 3Gbytes of RAM running Linux Red Hat 3.4.26.fc3. The running time given comprises the entire exploration process including the HLS.

B. *Results and Discussion*

Table 2 shows the results of the design space exploration. From the experiments it can be observed that our clustering method is on average 90% and 92% faster than the annealer method for the CDS-Exp(min) and CDS-Exp(max) method respectively. The drawback is that on average only 36% and 47% of all Pareto optimal designs are found. Table 2 also shows that our methods on average finds the smallest design that is 7% and 9% larger than the actual smallest case and 28% and 32% longer latency respectively. Although approximately one third and one half respectively of the Pareto optimal points are not found the smallest and fastest designs are found in some cases and in most cases almost found. This is important as in most cases these are the designs that are finally used and provide the boundary points of the exploration. In order to expand the search space using our method more clustering stages could be introduced changing the clusters attributes more often. This would find more optimal points at the expense of increasing the design space increasing therefore the runtime.

VI. CONCLUSION

High level synthesis is becoming a must in state of the art hardware designs. Designers can no longer describe and model entire SoCs in low level languages and need to raise the level of abstraction. Tools that bridge the gap between untimed high level languages and RTL are needed. In this paper we present a design space exploration method to speed up the exploration of high level language design descriptions given in C and SystemC. The presented method, called CDS-ExpA, is based on a clustering method that clusters explorable operations and assigns a fix set of attributes to these based on the global cost function in order to reduce the design space. Two variations of the CDS-Exp are presented. CDS-ExpA(min) creates the smallest possible clusters and CDS-ExpA(max) the largest possible ones. The trade-offs between further reducing the design space by building larger clusters vs. smaller is also investigated. Results show that the design space exploration dramatically reduces the runtime by around 90% at a cost of missing on average 36% and 47% of all Pareto optimal designs. On the other hand the smallest and fastest designs are found in many cases and on average 7% and 9% respectively larger than the actual smallest case and 28% and 32% longer latency designs are found. We believe that this exploration method is a valid solution for initial design space explorations as half of the Pareto points are found and closely the smallest and fastest ones which provides valuable design information to the designer at the earliest design stage extremely fast.

REFERENCES

[1] B. Carrion Schafer, T. Takenaka and K. Wakabyashi, "Adaptive Simulated Annealer for High Level Synthesis Design Space Exploration", VLSI-DAT, 2009

[2] BDL: http://www.cyberworkbench.com

[3] SA-C, http://www.cs.colostate.edu/cameron

[4] C. Haubelt, T. Schlichter, J. Keinert and M. Meredith, "SystemCoDesigner: Automatic Design Space Exploration and Rapid Prototyping from Behavioral Models", *DAC*, 2008.

[5] M. Kim, S. Banerjee, N. Dutt and N. Venkatasubramanian, "Design space exploration of real-time multi-media MPSoCs with heterogeneous scheduling policies", *CODES+ISSS*, pp. 1621, 2006

[6] S. Mamagkakis, D. Atienza, C. Poucet and F. Catthoor, D. Soudris, and J. M. Mendias, "Automated exploration of pareto-optimal configurations in parameterized dynamic memory allocation for embedded systems", *DATE*, pp. 874-875, 2006.

[7] I. Ahmad, M. Dhodi and F. Hielscher, "Design-Space Exploration for High-Level Synthesis", Computers and Communications, pp. 491-496, 1994.

[8] M.Holzer, B.Knerr and M. Rupp, "Design Space Exploration with Evolutionary Multi-Objective Optimisation", Proc. Industrial Embedded Systems, pp. 125-133, 2007.

[9] C. Haubelt and J. Teich, "Accelerating Design Space Exploration", International Conference on *ASIC*, pp. 79-84, 2003.

[10] V. Kianzad and S. S. Bhattacharyya, "CHARMED: A Multi-Objective Co-Synthesis Framework for Multi-Mode Embedded Systems", *ASAP*, pp. 28-40, 2004.

[11] S. Bilavarn, G. Gogniat, J.-L. Philippe and L. Bossuet, "Design Space Pruning Through Early Estimation of Area/Delay Tradeoffs for FPGA Implementations", ICCAD, vol. 25, pp. 1950-1968, October 2006.

[12] I.D.L Anderson and M.A.S. Khalid, "SC Build: a computer-aided design tool for design space exploration of embedded central processing unit cores for field-programmable gates arrays", IET Computers & Digital Techniques, pp 24-32, Vol.3, Issue1, January 2009

[13] So B., Hall M. W, and P. C Diniz, "A Compiler Approach to Fast Hardware Design Space Exploration in FPGA-based Systems", PLDI, pp. 165-176, June 2002.

[14] B. So, P. C. Diniz and M. W.Hall, "Using Estimates from Behavioral Synthesis Tools in Compiler-Directed Design Space Exploration", *DAC*, pp. 514-519, 2003.

[15] P. Coussy and A. Moraweic, "High-Level Synthesis from Algorithm Digital Circuit", *Springer*, ISBN 978-1-4020-8587-1, 2008.