An efficient Approach for Design Space Exploration using Static Constraints for IP-based SOC Design

Abhishek Agarwal,[*] Ananth.K.S,[*] Nikitas A Alexandridis, [*] Tarek EI-Ghazawi,[*]
and Sean X. Wang[†]

[*]*Department of Electrical and Computer Engineering, The George Washington University, Washington, DC 20052;*
[†]*Department of Information and Software Engineering, George Mason University, Fairfax, VA 22030*

Abstract

The growing demand for portable embedded computing devices has revolutionized SOC design, and in particular, the field of Intellectual Property. In this work, we propose a technique to form configurations from sets of IPs that satisfy the static system constraints. Here, 'set' refers to all IPs of same type (or class) that meets the user-specified conditions on IP attributes. The technical challenge lies in the huge number of potential configurations (i.e., the search-space exploration). A 'configuration' refers to a possible system configuration made up of one and only one IP from each of the above sets of IPs. The proposed approach extensively prunes this potentially large search space by taking advantage of vendor-specified IP attributes and employs the dynamic programming algorithm to yield a ranked list of configurations. The proposed approach also yields the total cost and area estimate of each configuration. The experimental results show that the proposed approach is effective in forming configurations that satisfy the static system constraints of cost and area.

1. Introduction

The growing requirements on the correct design of a high-performance system in a short time force us to use Intellectual Property in many designs. As the market pressures and product complexities increase, the pressure to reuse complex building blocks (also known as Intellectual Property or IP) increases significantly. This paper addresses the following aspects of IP-based embedded system design: search-space reduction, forming configurations and checking for optimality with respect to designer-specified static constraints. Moreover, we have many vendors providing IPs and there is no universal tool that searches for IPs from all the vendors simultaneously. The designer has to use search engines that are vendor-specific. Often, the designer will need to check products of various vendors in order to decide his optimal configuration. This process is cumbersome since the search tools currently used are vendor-specific and there is no generic tool that integrates the products of different vendors and thus helps the designer in his search for optimal configuration. The Intellectual Property Optimal Selection Tool or the IPOST [1] is an effort in this regard, which integrates all the IPs from various vendors in its search process and yields an optimal set as per the designer's needs. The IPOST is a complete tool, which facilitates the design of SOC from the behavioral specification to a complete SOC. The grand vision of IPOST has been shown in figure1 below. The IPOST has been divided into 5 levels [1]:

IPOST – Level -1

This level performs architectural exploration and a high level pruning of these generated architectures. This output is passed to level 0. This level is uncommonly called so because it was non-existent for some time and only later, figured in the design flow.

IPOST – Level 0

This level generates attributes for each of the architectural components generated by level -1. These attributes enable the designer to go into the web to search for the IPs. For example, an attribute of MB is generated for memory element and a value is also generated after analyzing the system requirements and application specifications. This output (i.e, attributes and its values) is passed to level 0.

IPOST – Level 1

This level aims more at facilitating the design of the system by optimizing the search and retrieval. This includes the selection of set of IP's for the system taking attributes and some user defined constraints (System and IP specific) during the selection procedure. The search optimization helps the designer to retrieve more results from many different databases by many different IP Providers. Each IP is searched on heterogeneous databases through the web-portal [2]. Level1 thus provides sets of IPs from a query depending on attributes.

IPOST – Level 2 (Optimal Set Generator)

These results obtained from level 1 are set for IP's and these set of IP's are now optimized to provide configurations. The Level 2 (also described as Optimal Set Generator) take user defined system constraints which help us in

defining the possible configurations depending on these constraints.

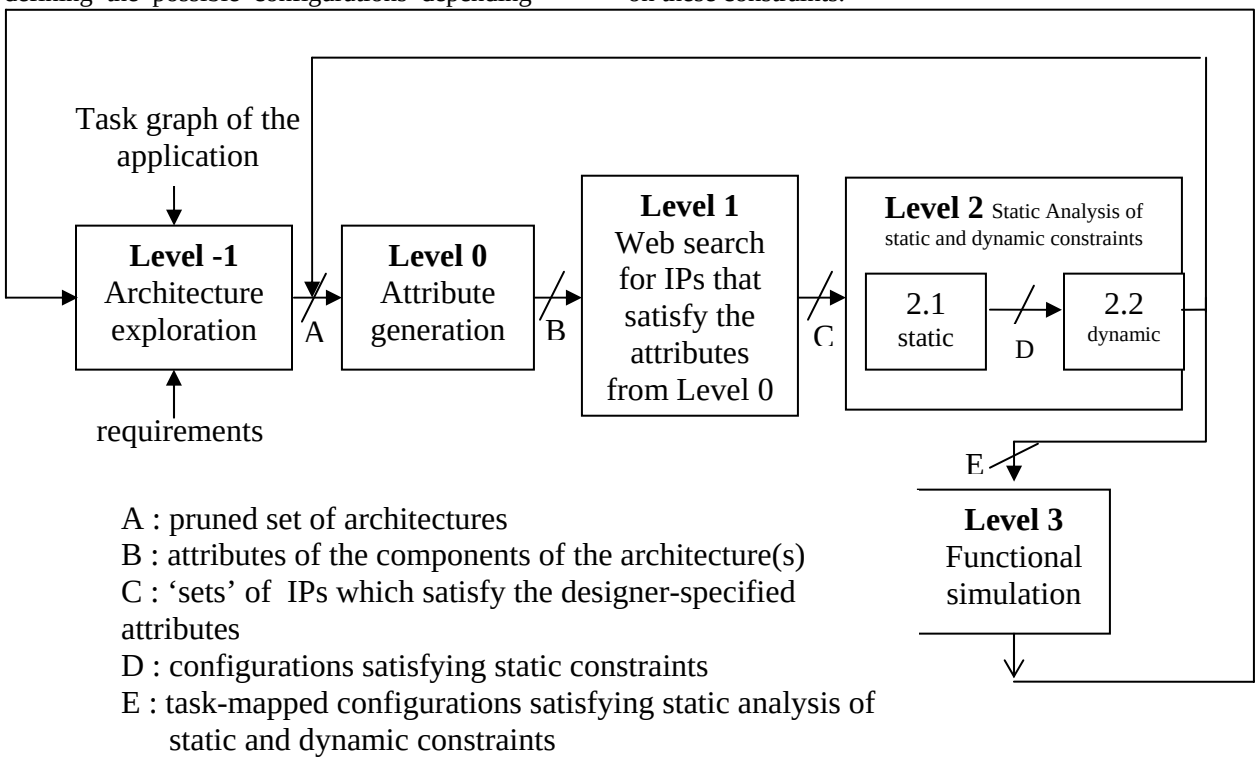


Fig.1 Block Diagram of IPOST

IPOST – Level 3

The Level 3 provides a dual purpose. It provides functional verification of the selected IP configurations and also provided a simulation to verify the user constraints. Level 3 has the following main functions:

- It provides the functional verification of the set of IP's and allows us to select the best set of IP's which meet all the user requirements.
- It provided the simulation of the SOC environment to make sure that the IP set meets all the user defined system level constraints.
- It also provides the mapping of behavioral specification to IP specific definitions at a higher level.

Level2 of the IPOST, also called the optimal set generator, forms configurations from the sets of IPs that satisfy static and dynamic system constraints. Level3 aims at providing functional verification of the IP configurations and also provides simulation of the SOC environment in order to verify that the IP configuration meets all system level constraints. In this paper, we explore the various search-space reduction algorithms available for the designer and propose an approach to form configurations from a reduced-search-space, which satisfy the static system constraints.

We have got sets of IPs coming in from Level1 and we want to form configurations of the IPs of these sets, which satisfy the system constraints, both static and dynamic. In our approach, we attack the static constraints first and then consider the dynamic constraints. The reason for this is that checking for dynamic constraints implies performing simulations on the configurations obtained. Instead of performing simulations on all the configurations, we break up the approach into 2 stages as Level2.1 and Level2.2. We first check the configurations for static constraints and perform a static analysis on these in order to evaluate the performance. We then feed this subset of configurations to the next stage where we check for dynamic constraints. Thus, we input a greatly reduced number of configurations to Level2.2.

To summarize, Level 2 is divided into 2 stages, Level2.1 and Level2.2. In Level2.1, we form configurations and check these for static constraints and at Level2.2, we check for dynamic configurations. In this paper, we propose an approach to solve Level2.1.

The Level 2.1 of the IPOST Tool aims at providing a configuration of IPs, which satisfies the static system constraints.

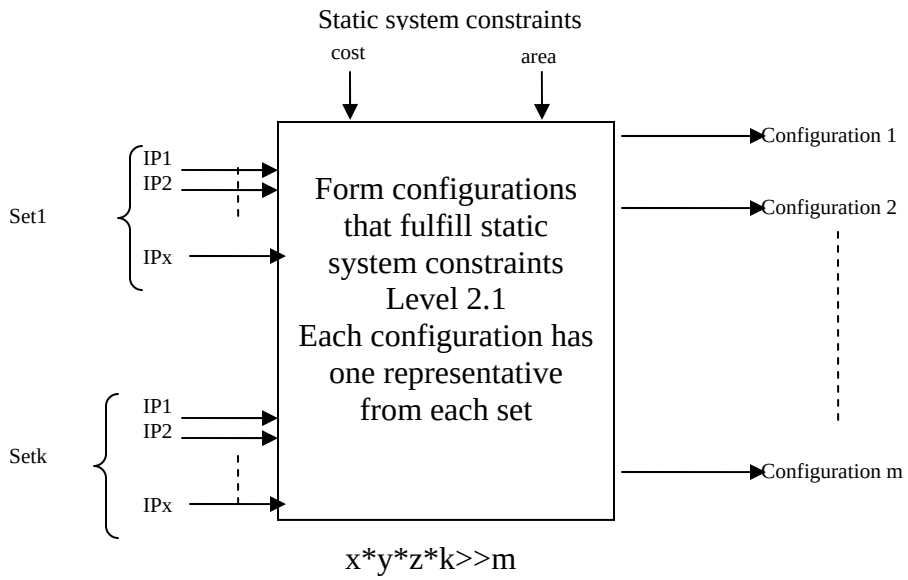


Fig.2 Block Diagram of Level 2.1 of IPOST

The figure above shows 'n' sets of IPs each having same or different number of IPs. These sets could represent sets of Processors, Memories, Peripherals, and the like. The objective is to form configurations out of these IPs, which satisfy the static system constraints of cost and area, with each configuration containing one IP from each of the sets. For example, if the outputs of Level1 are 3 sets of IPs namely, Processors, Memory and Peripherals, with 3 IPs in each, then, some of the possible configurations are Processor1_Intel, Memory1_ Philips, and Peripheral1_Motorola and so on. The designer will have fixed allocations of cost and area for the entire system design and the configurations thus formed must satisfy these static system constraints.

Level2.1 forms configurations from the sets of IPs that satisfy the static system constraints of cost and area. If the designer needs, it will also rank these configurations. This paper proposes an approach to solve level 2.1 of IPOST. The proposed approach efficiently explores the configuration space and outputs a set of optimal configurations, which satisfy the static system constraints.

The rest of the paper is organized as follows: In the next section, we state the problem and section 3 gives a background of the algorithms used for search-space reduction and some approaches followed to efficiently prune the vast search-space. In section 4, we explain the proposed algorithm. Finally, we present the results and conclusions in section 5.

2. The Problem

The primary problem faced by system designers is pruning the search-space to form optimal configurations which satisfy certain constraints. The formation of configurations involves exploration of a search-space which is extremely vast. Once the configurations have been formed, each one of these configurations ought to be checked for constraints. This process of analyzing several "functionally equivalent" implementation alternatives to identify an optimal solution is referred to as "design space exploration".

The problem addressed by this paper is as follows: Firstly, we have to prune the vast search space and form configurations of IPs from sets of IPs (which are the outputs of Level1 of IPOST). These configurations are subject to the static constraints of total cost and total area which are specified by the designer. Also, IPs are subjected to cost and area constraints specified by the respective IP vendor. Therefore, the configurations thus formed, must satisfy the static system constraints of total cost and total area. Secondly, we have to rank these configurations, taking into account the total cost and total area of each configuration. The proposed approach uses the dynamic programming algorithm [9] to form the optimal configurations.

3. Background

Efforts have been made in the past to reduce the search-space and form optimal configurations.

An approach to perform rapid design space exploration is explained in [3]. [3] discusses design space exploration based on constraint satisfaction and high-level performance estimation in the context of application design on heterogeneous embedded architectures. The integration of a design space exploration tool (DESERT) and high-level system wide latency and energy estimation tool (HiPerE) into the proposed MILAN framework facilitates multigranular simulation and updation of performance estimates. But, since the primary emphasis of [3] is performance estimation, we do not explore this tool in detail in our paper. On the contrary, our paper not only proposes an efficient approach to form configurations, but also performs optimality check on these configurations. Nevertheless, the reader can get a more clear insight into design space exploration in the above-mentioned paper.

The Strength Pareto Evolutionary Algorithm or the SPEA is an evolutionary algorithm [4], [5], [11], [17] which performs multi-objective optimization using the ideas of genetics [10] and clustering algorithms to perform search space reduction. But this algorithm uses dynamic constraints in its effort to perform optimization whereas in our problem, we are dealing only with static constraints.

The basic idea in most of the algorithms is to reduce the search space by forming subsets of the initial group based on some static or dynamic parameters, which are user-specified.

The brute force algorithm [12] is definitely the last option for a design engineer, mainly due to the enormous complexity of the algorithm, which increases as the search space is increased.

The clustering process [6], [16] is widely used in search space reduction in many approaches. The entire search space can be effectively pruned using this algorithm. In the SPEA algorithm discussed above, cluster distances between different individuals are used in the process. Thus, here, a dynamic property is used to merge multiple closeness metrics into a single closeness value. In our problem, we are dealing with static constraints. We propose to use the implicit static attributes of the IPs in order to reduce the search space. This approach is explained in detail later.

In group migration [6], ratio-cut [7] and simulated annealing [8], a variant of the idea employed in clustering is used. The metric used in clustering algorithms is ‘distance’ whereas in these three, ‘cost’ is the metric used, the primary idea being the same. In our approach, we use the ‘cost’ metric in the dynamic programming algorithm and hence, we restrict ourselves to the metric ‘attributes’ to perform the process of clustering.

The Pareto-optimal [11] approach requires simulations, which take a lot of time and the designer may not be interested to perform simulations so early in the design process. Moreover, this approach acts on dynamic constraints, whereas we are dealing with static constraints in our approach. Hence, it is not suitable for our case.

In the SPEA algorithm, the idea of fitness assignment can be used in our approach. The “fitness” is evaluated based on dynamic parameters, but we can use the idea for static parameters, which is explained in the subsequent sections. Also, used in this algorithm is the process of clustering, which can be employed in our approach as stated earlier.

The proposed approach forms configurations which satisfy the static system constraints of cost and area and ranks these configurations. It can thus be termed as a multi-objective optimization algorithm. This is accomplished by employing the dynamic programming algorithm and uses the idea of clustering to prune the search space.

4. Proposed Approach

The objective of the proposed approach is to form configurations which satisfy the static system constraints of cost and area. Each configuration must contain one IP from each of the sets of IPs obtained from level1.

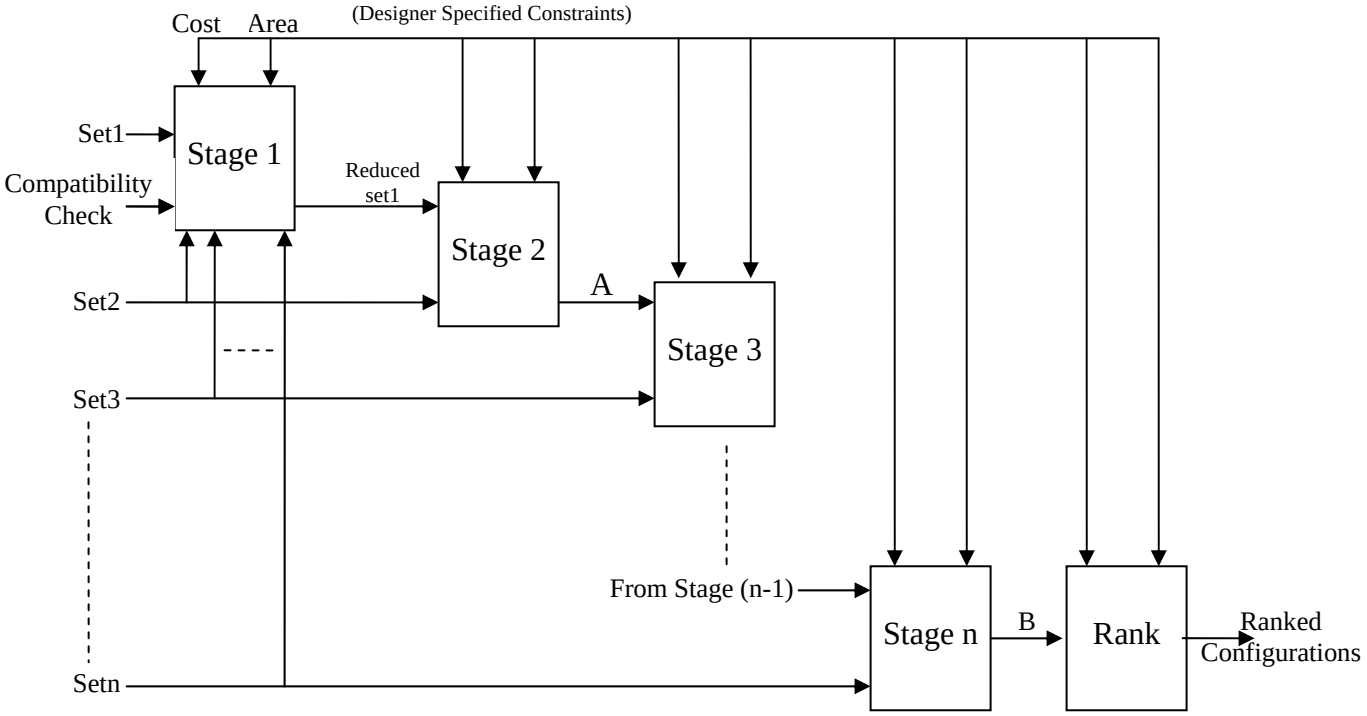
The proposed approach uses the ideas of fitness assignment and clustering algorithms discussed above to prune the search space and finally apply the reduced set of inputs to the dynamic programming algorithm, which forms configurations, which satisfy the static system constraints namely cost and area. The IP vendor provides the cost and area for each IP and the designer is allocated a fixed cost and area for the entire system design. Hence, we have taken cost and area as static constraints. A more detailed explanation of static and dynamic constraints in the process of SoC design is provided in [13].

The proposed approach consists of the following ideas.

Fitness evaluation: The cost and area specifications of the IP are normalized. Thus, the idea of fitness, which is used in SPEA algorithm on dynamic constraints, is used on the static constraints in our approach. We then compare

these normalized values with a threshold and eliminate those IPs, which are beyond our range. For example, if the designer wants to allocate 60% of the total cost allocation to the class of processors, then we can eliminate all the IPs, which are beyond this range.

Dynamic Programming Algorithm: Dynamic Programming [9], [12] is a technique for solving problems with overlapping sub problems. Typically, these sub problems arise from a recurrence relating a solution to a given problem with solutions to its smaller sub problems of the same type. Rather than solving overlapping sub problems again and again, dynamic programming suggests solving each of the smaller sub problems only once and recording the results in a table from which we can obtain a solution to the original problem. The IPs of the first set are taken and then, tested for compatibility with IPs of other sets using the idea of clustering. We propose to use the implicit system constraints, that is, the VSIA specified IP attributes [14] to perform clustering. This step helps to avoid unnecessary inputs from entering the next step of the dynamic programming algorithm, which in-turn increases the efficiency of the approach.



A: initial configuration of set1 and set2 satisfying the cost and area constraints

Fig.3 Proposed Approach

B: configurations which satisfy the static system constraints of cost and area, each containing n IPs

The figure above illustrates our approach. The IPs of set1 are copied to stage 1 of dynamic programming. The cost and area constraints are input to stage 1 and the IPs whose cost or area exceeds the constraint values are discarded. A compatibility check is also performed at this stage with the IPs of other sets. The VSIA-defined attributes [14] are used to perform this check

These ideas are summarized in the following steps.

Step1: Normalize [15] the cost and area of each IP of all the sets using the formula:

$Cost = a*w_{cost}*cost/c' + b*w_{area}*area/a'$

where

Cost=dynamically weighted cost function

a=a weight on a scale of 10assigned by the designer

w_{cost} =cost constraint specified by the designer

cost=cost of the particular IP considered

c'=average cost of all IPs in the particular set

b=a weight on a scale of 10 assigned by the designer
w_{area} =area constraint specified by the designer
area= area of the particular IP considered
a'=average area of all IPs in the particular set

Compare these normalized values of cost and area with some threshold, which is a static value in terms of cost and area. And neglect those IPs which cannot be potential candidates in the optimal configuration.

Step2: Here, we apply the dynamic programming algorithm. This step is divided into stages and the number of stages equals the number of sets of IPs.

In stage1, we form an initial table consisting of the cost and area of set1. After this, we check for the compatibility of IPs of this set with the IPs of other sets. For this, we use the inbuilt IP attributes specified by the VSIA [14]. This helps us to eliminate unwanted inputs from entering the subsequent stages of the dynamic programming algorithm.

In the subsequent stages, combinations of IPs from sets are formed which satisfy the static system constraints. The termination condition is met when configurations have been formed with each configuration containing one IP from each of the sets.

Step3: Here, we rank the configurations formed by calculating the normalized values of cost and area obtained in the previous table by using the formula:

(x-min)/ (max-min)*weight

where

min=minimum of the list

max=maximum of the list

weight=value of cost or area, whichever the designer wants to emphasize more.

4.3 Toy Example

We shall now take a toy example (in the form of outputs of Level1) and carry it out all the way to the production of outputs of level 2.1.

Table1:

Outputs of Level1: Sets of IPs

IP	Set1		Set2		Set3	
	Cost	Area	Cost	Area	Cost	Area
1	7	2	2	7	3	6
2	5	3	1	4	4	8
3	3	5	3	2	3	2
4	6	5	1	3	1	4
5	2	4	5	2	5	3
6	1	3	4	3	6	3
7	5	8	6	4	4	6
8	6	3	5	2	5	8

We have to make configurations that meet the static system constraints of cost=$10 and area=10 m^2, where each configuration has 1 representation from each of the IP sets.

Step1: Normalization [15] is calculated as follows: We shall express the cost and area of each of the IPs by a dynamically weighted cost function as follows:

$Cost = a*w_{cost}*cost/c' + b*w_{area}*area/a'$

where

Cost=dynamically weighted cost function
a=a weight on a scale of 10assigned by the designer
w_{cost} =cost constraint specified by the designer
cost=cost of the particular IP considered
c'=average cost of all IPs in the particular set
b=a weight on a scale of 10 assigned by the designer
w_{area} =area constraint specified by the designer
area= area of the particular IP considered
a'=average area of all IPs in the particular set

Table2: Normalization

IP	Cost of Set1	Cost of Set2	Cost of Set3
1	2.606	3.33	2.46
2	2.33	1.85	3.29
3	2.37	1.85	1.46
4	3.22	1.11	1.32
5	1.78	2.59	2.36
6	1.19	2.59	2.68
7	3.85	3.7	1.62
8	2.62	2.59	3.61

Comparing these normalized values of cost and area with some threshold, which is a static value in terms of cost and area, we neglect those IPs that cannot be potential candidates in the optimal configuration.

Thus, normalization helps us to evaluate the so-called 'Fitness' or the weight of each of the IPs, and these are compared with a threshold and the unwanted IPs are removed, thus reducing the search space, as shown in table below.

Step2: **Application of Dynamic Programming**

We have used the dynamic programming algorithm on two constraints, namely cost and area. Hence we shall revert back to the cost and area values of each of the IPs. Now, lets assume that the total cost allocated to the designer =$10 and the total area allocated for the entire system=$10m^2$.

The conclusion at the end of stage1 is that all these IPs can be potential members of the optimal configuration.

After stage1, we intend to check for compatibility of IPs. For example, if the designer is looking for a 16-bit processor, then an IP of 32-bit word length from the class of memory will not be compatible with the required search. Hence it is discarded. In-fact using the IP attributes, we can define some rules for checking compatibility. Thus, we are making use of implicit system constraints to discard unwanted solutions from entering the subsequent steps of dynamic programming.

Table3**: Stage1 of Dynamic Programming Algorithm**

IP	Set1	
	Cost	Area
1	7	2
2	5	3
3	3	5
4	6	5
5	2	4

Stage2: The possible configurations of sets 1 and 2, which satisfy static system constraints, are evaluated in this stage.

Stage3: The possible configurations of sets 1,2 and 3, which satisfy the system constraints, are:

Thus the optimal configurations that satisfy the system constraints are: IP3_Set1, IP3_Set2, IP3_Set3; IP3_Set1, IP2_Set2, IP3_Set3 and so on.

Table4: Stage 3 of dynamic programming algorithm

Cost combinations	Total cost	Total area consumed
3,3,3	9	9
3,1,3	7	10
6,1,3	10	10
2,1,3	6	10
2,3,3	8	8
2,3,5	10	9
2,1,3	6	9
2,1,5	8	10

Step3: Ranking of the configurations:

We use the following formula for ranking the configurations obtained: (x-min)/ (max-min)*weight

where

min=minimum of the list

max=maximum of the list

weight=value of cost or area, whichever the designer wants to emphasize more.

Thus, the configurations are ranked as follows: IP5_Set1, IP2_Set2, IP1_Set3; IP5_Set1, IP2_Set2, IP3_Set3; IP3_Set1, IP2_Set2, IP3_Set3 and so on.

5. Results

This section is divided into the following 2 sub-sections: In 5.1, we discuss the experimental results

5.1 Experimental Results

We shall compare the proposed approach with the brute force algorithm in terms of number of configurations explored. We used C++ as a programming tool to verify our results. We took a simple input with 3 sets of IPs with 5 IPs in each. The system designer specifies the static constraints of cost and area. The designer is allocated a certain cost and area for his system. Our algorithm has to form configurations of these sets with each configuration having one representation from each of the sets.

	Set1		Set2		Set3	
	cost	area	cost	area	Cost	area
IP1	7	2	2	7	3	6
IP2	5	4	1	4	4	8
IP3	3	5	3	2	3	6
IP4	6	5	1	6	1	4
IP5	2	4	5	5	5	3

For the above set of inputs, if the designer has the overall system constraints of 10 and 10 respectively for cost and area, then the brute force algorithm output is as follows:

"The valid configurations are

1. IP1_Set1, IP2_Set2 and IP4_Set3;
2. IP2_Set1, IP3_Set2 and IP4_Set3;
3. IP5_Set1, IP3_Set2 and IP4_Set3;
4. IP5_Set1, IP3_Set2 and IP5_Set3.

Number of configurations generated in total: 125"

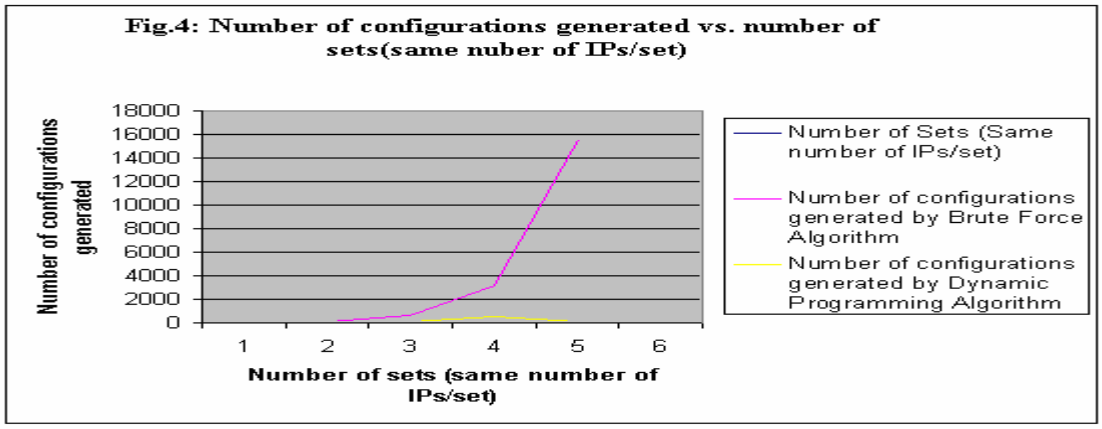
For the same set of inputs, the dynamic programming yields the same result as above, but the major change is that "Number of configurations generated in total: 85".

Now, if the designer specifies the overall system constraints of cost and area as 6 and 8 respectively, then, no configuration formed from the above sets satisfy these constraints, yet there is a marked difference in the total number of configurations generated: The brute force algorithm generates 125 configurations, whereas the dynamic programming generates only 20 configurations.

The Brute Force Algorithm looked at 5^3 =125 configurations and compares the total cost and area of each configuration with the total allocation and retains the configuration only if it is below the total allocation. On the other hand, the dynamic algorithm does not look at all the configurations; it only looks at those configurations that satisfy the static system constraints. Hence, the number of configurations looked at will depend on the total allocation of cost and area for the system. Unlike the brute force algorithm, which forms all possible configurations, dynamic algorithm forms only those, which satisfy the static system constraints. In the above example, for the same values of cost and area constraint (10 and 10 respectively) for both the algorithms, 85 configurations are looked at, thus, providing a 32% improvement. If the cost and area allocation were reduced to lower values, say 6 and 8, for the same set of inputs, the number of configurations explored remains 125 in the brute force, but is drastically reduced to 20 in the dynamic programming algorithm, thus providing an improvement of 84%.

We have shown the experimental results for various scenarios in the form of graphical representations. Figure 4 is a plot of number of configurations vs. number of sets of IPs, with each set having the same number of IPs. Number of configurations explored is directly proportional to the time required for the execution of the algorithm. As can be seen from the graph, fewer configurations are explored by the dynamic programming algorithm when compared to the brute force algorithm. The next three scenarios involve varying one of the constraints namely cost/area and varying both while calculating the number of

configurations explored. In all the cases, the number of sets of IPs considered was 5 and each set had 5 IPs. It is evident from the graphs that the number of configurations explored by the proposed approach is much less compared to the brute force algorithm.



As is evident from the above graph, the brute force algorithm follows an exponential curve in terms of number of configurations explored. As the number of sets of IPs increase, the number of configurations explored by brute force increases. But in the case of dynamic programming algorithm, the number of configurations explored depends on the total constraint allocation for the whole system. Nevertheless, the number of configurations explored will be much less than the brute force algorithm.

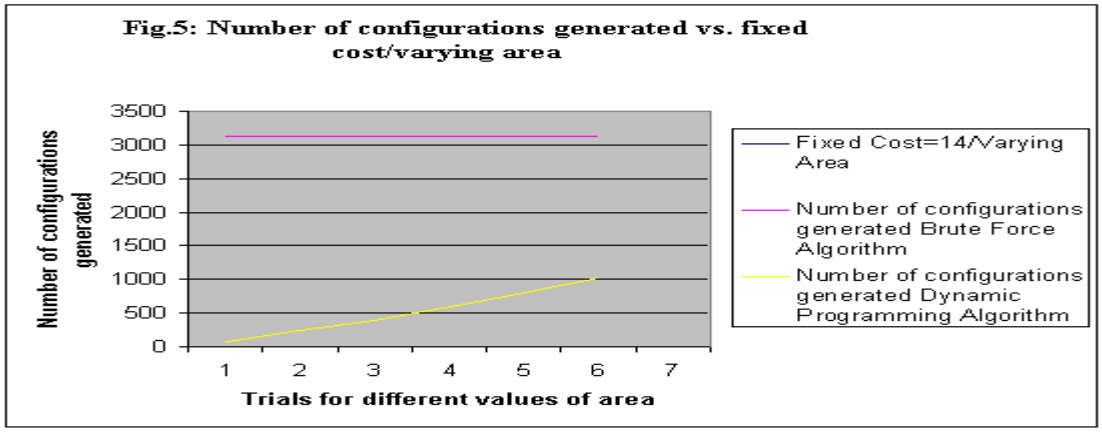


Fig. 5 illustrates the plot of number of configurations explored for different values of area, with the cost constraint being constant. The number of configurations explored by the brute force remains constant at 3125 irrespective of the area constraint.

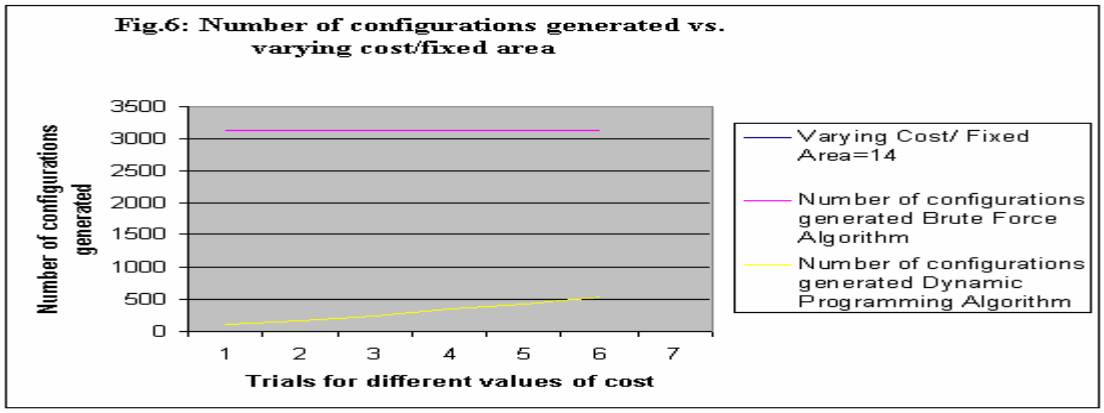


Fig.6 shows the graph of number of configurations explored for different values of cost, with the area constraint being constant. As in the previous scenario, the number of configurations explored by brute force algorithm remains constant at 3125 irrespective of the value of cost constraint.

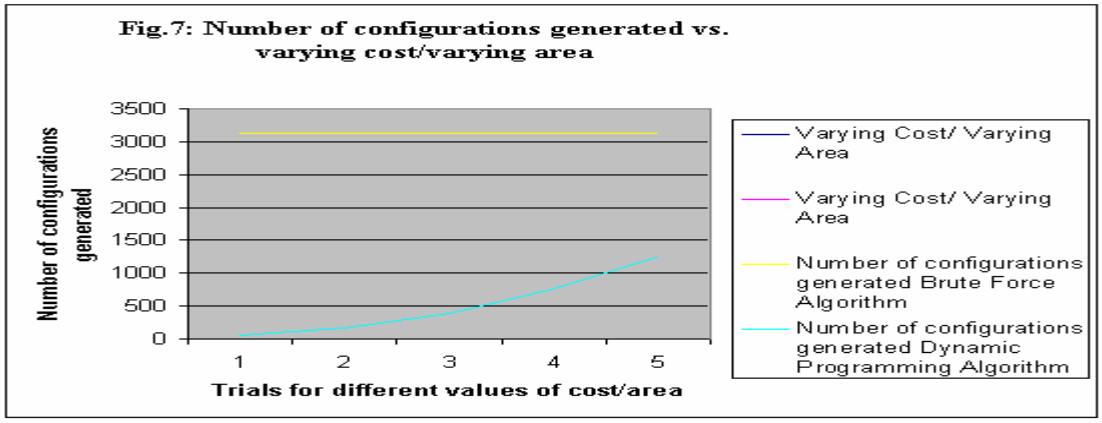


Fig.7 illustrates a plot of number of configurations explored with varying values of cost and area constraints. Again, the number of configurations explored by the brute force algorithm remains constant at 3125 whereas the proposed approach explores much less configurations.

5.2 Conclusions

In this paper, we have presented an approach, which performs an efficient search space reduction and forms configurations from sets of IPs, which satisfy the static system constraints of cost and area. This was possible by using ideas of fitness evaluation from the Strength Pareto Evolutionary Algorithm (SPEA), the clustering algorithm and the dynamic programming algorithm. We also used the VSIA-defined attributes in this approach in order to make the tool more intelligent. These attributes enable us to check for compatibility of IPs and hence, help in reducing the search space. Thus, the proposed approach is very efficient as it explores fewer configurations and hence execution time is also higher than the brute force algorithm. The experimental results and the graphs clearly illustrate the efficiency of the proposed approach. The graphs were constructed in various scenarios and all the plots show that the proposed approach is superior to the conventional brute force approach in terms of the execution time, which is a measure of the number of configurations explored. This approach can be adopted by IP-based embedded-system designers to form optimal configurations which satisfy the static system constraints of cost and area.

References:

[1] Abhishek Agarwal, Anuj Mallick, Suboh Suboh, Nikitas Alexandridis, Tarek El-Ghazawi (GWU): An Open Source Intellectual Property Optimal Selection Tool (IPOST – Level1): The 2004 International Multiconference in Computer Science and Engineering, 2004

[2] Abhishek Agarwal, Nikitas A Alexandridis, Tarek EI-Ghazawi, Zhengrong Yao, and Sean X. Wang "An Open XML IP Search Portal Prototype"

[3] S.Mohanty, V.K.Prasanna and S.Neema J.Davis: "Rapid Design Space Exploration of Heterogeneous Embedded Systems using Symbolic Search and Multi-Granular Simulation" Berlin, Germany Session: Synthesis and Design Space Exploration, 2002

[4] E.Zitzler and L.thiele. Multiobjective evolutionary algorithms: A comparative case study and the Strength Pareto approach. IEEE transactions on Evolutionary Computation, 4(3): 257-271, Nov.1999

[5] E.Zitzler, M.Laumanns, and L.Thiele. SPEA2: Improving the performance of the strength pareto evolutionary algorithm. Technical Report TIK-Report 103, Computer Engineering and Communication Networks Lab, Swiss Federal Institute of Technology (ETH) Zurich, Gloriastrasse 35, CH-8092, May 2001

[6] "Specification and Design of Embedded Systems" by Gajski (chapter 6)

[7] Wei, Y.-C. Cheng, C.-K : "Ratio cut partitioning for hierarchical designs": IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Jul 1991

[8] S. Kirkpatrick, C. D. Gelatt, Jr., M. P. Vecchi : "Optimization by simulated annealing": 13 May 1983, Volume 220, Number 4598

[9] A tutorial on Dynamic Programming http://mat.gsia.cmu.edu/classes/dynamic/dynamic.html

[10] Maurizio Palesi and Tony Givargis. "Multi-Objective Design Space Exploration Using Genetic Algorithms"

[11] Multi-Objective Optimization using Evolutionary Algorithms by Kalyanmoy Deb

[12] Introduction to The Design and Analysis of Algorithms by Anany Levitin

[13] Tony .D Givargis, Frank Vahid: "Parameterized System Design

[14] Virtual Socket Interface Association, VC Attributes with formats, March 2001: http://www.vsi.org

[15] Jorg Henkel and Rolf Ernst "High-Level Estimation Techniques for usage in Hardware/Software Co-Design": Asia and South Pacific Automation Conference, pp. 353-360, Yokohama, Japan, February 1998.

[16] Tony Givargis, Frank Vahid, and Jorg Henkel, "System Level Exploration for Pareto-Optimal Configurations in Parameterized System-on-a-Chip" December 2002 IEEE. IEEE transactions on Very Large Scale Integration (VLSI) systems, vol. 10, no. 4, August 2002

[17] J.Heitkotter and D.Beasley. The hitch-hiker's guide to evolutionary computation.

http://surf.de.uu.net/encore/www/,Apr. 12 2001