Buffer Size Driven Partitioning for HW/SW Co-Design

Ta-Cheng Lin
IBM Microelectronics
Austin, Texas 78758
tclin@ibm.net

Sadiq M. Sait
Computer Engineering Dept.
King Fahd University of Petroleum & Minerals
Dhahran-31261, Saudi Arabia
sait@kfupm.edu.sa

Walling R. Cyre
Electrical Engineering Dept.
Virginia Tech
Blacksburg, VA 24061-0111
cyre@vt.edu

Abstract

Partitioning is a very important task in hardware/software co-design. Generally the size of the edge cut-set is used to evaluate the communication cost. When communication between components is through buffered channels, the size of the edge cut-set is not adequate to estimate the buffer size. A second important factor to measure the quality of partitioning is the system delay. Most partitioning approaches use the number of nodes/functions in each partition as constraints and attempt to minimize the communication cost. The data dependencies among nodes/functions, and their delays are not considered. In this paper we present partitioning with two objectives: (1) buffer size, which is estimated by analyzing the data flow patterns of the CDFG, and solved as a clique partitioning problem, and (2) the system delay that is estimated using List Scheduling. We pose the problem as a combinatorial optimization and use an efficient non-deterministic search algorithm called Problem-Space Genetic Algorithm to search for the optimum. Results are compared with those produced by simulated annealing.

1. Introduction

In this paper, we integrate a new buffer size estimation algorithm and system delay for partitioning in a hardware/software co-design environment. The goal of HW/SW co-design is to map/partition the given system specifications to software, which is to be executed by microprocessors, and hardware, which is to be executed by co-processors (such as full custom ASICs or synthesized FPGAs), and satisfy the required system constraints. The constraints may comprise area, performance, power, *etc*. A communication channel must be established to transfer the data back and forth between software and hardware components. The communication channels can be queues, stacks, or common memory areas [1,2].

The size of the communication channel is a good way to estimate the communication cost for partitioning results at the high-level design stage [1]. The size of the edge cut-set is commonly used for the communication cost estimation. As a matter of fact, the size of the communication channel is usually smaller than the size of the edge cut-set if the communication is through buffers [1,2,3]. Furthermore, a larger edge cut-set does not imply a larger buffer size. For example, Figure 1(a) and (b) show the possible partitions for a design. The edge cut-sets for (b) is 5 and for (a) is 3. Although (b) has a larger edge cut-set than (a) has, (b) requires only 2 registers in its buffer as opposed to 3 registers for (a). Details of computing the buffer sizes will be presented in Section 4. However, estimation of the buffer sizes at the function-partition stage is not an easy task. The reasons are that the technology, physical layout, clock speed, wire delay, *etc*., are unknown at the high-level design stage. These unknowns cause the uncertainties of the variable lifetimes at the function-partition stage, which makes the estimation of buffer sizes very difficult.

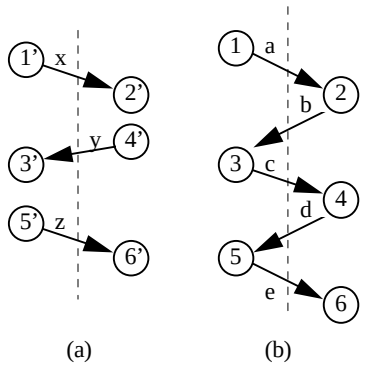

Figure 1. Two possible partitions for a design with different edge cut-sets.

Another important factor to measure the quality of the partitioning result is the system delay. Most of the partitioning approaches [4] set up the number of the nodes/functions in each partition as the constraint for partitioning, and then try to minimize the communication cost. It may be noted that the granularity of functions

referred in this paper can be as fine as operations or as coarse as tasks. The effects of the data dependency among these nodes/functions, the delays of each function, and the sequences of executing the functions are not considered. These effects have a great impact on the system performance. For example, in Figure 2, although (a) and (b) have the same communication cost and the same number of nodes in each partition, (a) is preferred. The reason is that (b) requires a longer execution delay because of the data dependencies among the functions and the blocks of the functions assigned to.

In our co-design environment, the system specifications are described using VHDL, and the VHDL code is translated into Control Data Flow Graphs (CDFGs). The CDFGs are used as inputs to a new partitioning algorithm to map the functions/nodes in the CDFGs into software or hardware. The communication cost between the HW/SW components are estimated by the buffer size. A new algorithm is proposed to estimate the upper bound of the buffer size by labeling each edge in the CDFG with a *path vector*. A procedure is then employed to transform each edge's path vector into a compatibility graph. The problem of finding the buffer size upper bound is transformed to a compatibility graph clique-partitioning problem. The purpose of getting an accurate buffer size estimation is to use the estimation as well as system delay as a metric to evaluate the quality of the partition results. We pose the problem as a combinatorial optimization and use an efficient non-deterministic search algorithm called Problem Space Genetic Algorithm (PSGA), which is a variant of Genetic Algorithm, to search for the optimum.

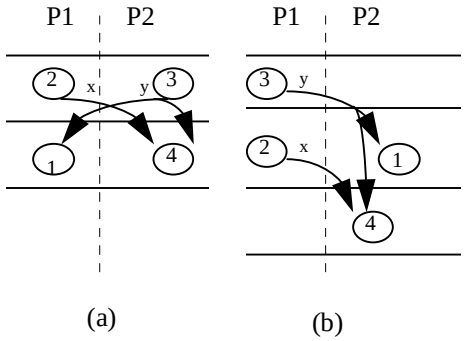

Figure 2. Partition results with different system delays.

The rest of this paper is organized as follows. Section 2 reviews related partition algorithms and applications of genetic algorithms in digital design. Section 3 models partitioning as a multi-objective optimization problem. Section 4 presents the buffer size estimation algorithm. Section 5 addresses the PSGA modeling technique. Experimental results are summarized in Section 6 and conclusions are in Section 7.

2. Previous Related Work

The min-cut partitioning algorithm [5] uses the size of the edge cut-set to measure the quality of partitioning. The algorithm interchanges subsets of nodes between two blocks to get a maximal partitioning improvement. Agrawal and Gupta stated that a min-cut partitioning algorithm which uses the size of an edge cut-set is not accurate enough to estimate the number of buffers needed, which is a better partitioning quality measure, for inter-processor communications [1]. They proposed a data-flow assisted behavioral partitioning algorithm that can more accurately estimate the inter-processor communication cost. Their algorithm is limited to two-way partitioning, and the number of functions in each partition is used as a constraint instead of the system delay which is a better measure of performance.

Partitioning is known to be an NP-complete problem [4]. Approaches that have been proposed by previous researchers are usually based on constructive heuristics such as the min-cut algorithm, or some non-deterministic hill-climbing algorithms such as the simulated annealing [4]. The major disadvantage of constructive heuristics is that they get trapped in local optima and are therefore unable to attain global optimum solutions. For the simulated annealing approach, in order to achieve a satisfactory result, the cost of CPU time is usually very high [4]. Another non-deterministic optimization algorithm called the Genetic Algorithm has been applied successfully in numerous research areas [6,7]. Problem Space Genetic Algorithm was first proposed by Storer *et al.* [8]. Storer *et al.* realized that infeasible solutions occurred during the evolution process in each generation for conventional genetic algorithms. The infeasible solutions must be either corrected by a repairing mechanism or be discarded. They proposed an alternative way to handle the occurrence of infeasible solutions by perturbing the problem space instead of the solution space. A fast heuristic algorithm is then used to map the problem space into the solution space, which guarantees that the solutions are always feasible.

3. Problem Formulation

The CDFG G = (V, E), which is used to describe the system behavior, consists of a set of functions which are represented by vertices $V = \{v_i \mid i = 1,2,..m\}$, and a set of data dependencies which are represented by edges $E = \{e_{ij} \mid e_{ij} = (v_i,v_j),\ v_i, v_j \in V\}$.

The CDFG is then used as input to our partitioning algorithm. The problem of partitioning V into two or more interacting blocks can be expressed as $P = \{Pi \mid i = 1,2,....N\}$, where $Pi = (Vi, Ei)$, $\cup Vi = V$, and $Vi \cap Vj =$

Ø if i ≠ j, with the constraints of minimizing the communication cost and system delay. The functions in the same block, i.e., Vi, are assigned to the same functional units or processors for execution. The delay for each partition block is the sum of the functional execution delays, and the time during which the functional unit is idle. The system delay is therefore the maximum delay of the functional unit delays, which can be expressed as:

$$\mathbf{T} = \max_{j=1,N} \left(\sum_{i=1,|Vj|} Ti + idle(j) \right) \qquad (1)$$

Where Ti is the execution delay for function v_i. The interface communication cost is expressed as:

$$\mathbf{R} = \sum_{j=1,N-1} \sum_{i=j+1,N} (b_{ij} + b_{ji}) \qquad (2)$$

Where b_{ij} and b_{ji} are the buffer sizes required to support the two-way communication between partitions **i** and **j**. The algorithm for buffer size estimation will be presented in Section 4. Having defined the system delay and communication cost, the objective of behavioral partitioning can be formulated as follows:

Objective: Given a CDFG G = (V, E), find a partition **P** such that the cost function

$$\mathbf{C} = \alpha\mathbf{T} + \beta\mathbf{R} \qquad (3)$$

is minimized, where α and β are weights used to control the desired tradeoff between system delay **T**, and communication cost **R**.

4. Buffer Size Estimation

The buffer size between two partitions can be estimated by tracing the data flow in the CDFG. The algorithm is a two-step process, which includes labeling the edges of the CDFG with *path vectors (PVs)* and then transforming the edges into compatibility graphs. The compatibility graphs are then used to find the upper bound on the buffer size.

4.1 Path Vectors

Labeling the edges with path vectors is used to detect the variables with non-overlapping lifetimes. The variables with non-overlapping lifetimes can share the same register in a buffer, which leads to buffer size reduction. For example, in Figure 1(b), Variables **a**, **c,** and **e** are non-overlapping because the data dependencies among Functions **1**, **2**, **3, 4, 5** and **6**. Variables **a**, **c,** and **e** can share a register in Buffer $\mathbf{b_{12}}$.

A path vector is a bit vector. The dimension of a path vector is the number of paths in the CDFG. The paths in the CDFG can be found by using entry nodes, which are the nodes without predecessors, as the roots, and then performing the Depth-First Traversal [9]. For example, in Figure 3(a), the roots for this CDFG are Nodes 1, 2, and 3. After using the roots to find all the paths, which are shown in Figure 3(b), each path is represented by a one-hot bit vector, e.g., [001] is used to represent Path 1. All the edges that belong to that path are labeled with the same path vector. If an edge is traversed by more than one path, the edge is labeled with the bit-wise OR of the path vectors. For example, in Figure 3(a), edges **c** and **d** are traversed by paths **1** and **2**; therefore, edges **c** and **d** are labeled with a path vector [011]. The formal notation of the path vector for an edge is denoted as $\mathbf{PV(e_{ij})}$, where $\mathbf{e_{ij} \in E}$.

The path vectors are then used to determine whether two variables are potentially lifetime overlapped or not. Two variables are lifetime non-overlapping if the bit-wise AND of their path vectors is not a zero vector. If the result of bit-wise AND of two path vectors is a zero vector, then the two variables are potentially lifetime overlapped.

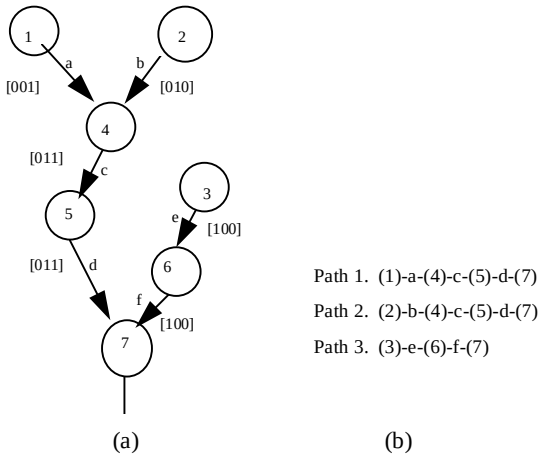

Figure 3. CDFG labeled path vectors.

4.2 Buffer Size Upper Bound

After all the edges are labeled with path vectors, compatibility graphs can be constructed to estimate the upper bound of the buffer sizes. For two partition blocks $\mathbf{P_i}$ and $\mathbf{P_j}$, the edge cut set, which is used to represent the variables generated by the functions in $\mathbf{P_i}$ and consumed by the functions in $\mathbf{P_j}$, is denoted as $\mathbf{C_{ij} = \{ c_{ij} \mid c_{ij} = (v_i, v_j), v_i \in V_i, v_j \in V_j \}}$. The compatibility graph for $\mathbf{C_{ij}}$ is constructed as follows:

- for every edge pair **(c_{ij}, c_{ij}')**, where c_{ij}, c_{ij}' $\in$ C_{ij} and $c_{ij} \neq c_{ij}$',
- if **PV(c_{ij})** AND **PV(c_{ij}') ≠ 0**, then introduce an edge between c_{ij} and c_{ij}'.

For example, if Figure (3) is partitioned into two blocks, P_1={1,3,5} and P_2={2,4,6,7}, the edge cut-set C_{12} and the path vectors are shown in Figure 4(a). The compatibility graph is shown in Figure 4(b).

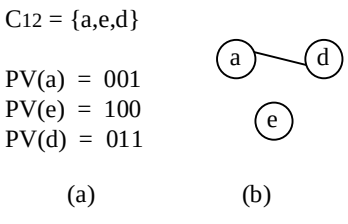

Figure 4. (a) Path vectors and (b)Compatibility graph for C_{12}.

The buffer size upper bound for b_{ij} is then transformed into a clique partition problem, which is to find the minimal number of cliques to cover the compatibility graph. Finding a clique partition is an NP-complete problem [4,10, 11]. An effective heuristic algorithm proposed by Tseng and Siewiorek [12] is used to find the minimal number of cliques for our buffer size estimation algorithm. Analogously, the buffer size for b_{ji} can be obtained by applying the above procedure and the buffer size upper bound for Partition **i** and **j** is equal to $b_{ij}+b_{ji}$. The system buffer size can be attained by computing the buffer sizes of each partition pair and then adding them up, which is expressed in Equation (1). The pseudo-code of the buffer size estimation algorithm is given in Figure 5.

```
Algorithm Buffer_Size_Estimation
// initialize data
Ne = | E |;  // number of edges in the CDFG
// set of nodes that don't have predecessors in the CDFG
So =  set of entry nodes;
// set of nodes that don't have successors in the CDFG
Si =  set of exit nodes;
Path  = 1;
N = number of  blocks;              // N_way partitioning
For each edge i, i ∈ E;
        PV(i) = [0];              // [0] is denoted as a 0 bit vector
End For;
// label the edges with path vectors
For each node i , i ∈ So;
        use i as root to perform Depth-First Traversal;
        if node j is visited ; j ∈ Si; then
        // a path from Node i to Node j is found
                Path = Path + 1;
        Label each traversed edge e,
          with PV(e) = PV(e) | To_Bit_Vector (Path) ;
        // '|' is denoted as a bit-wise OR operation
        // To_Bit_Vector() is a function to convert
        // an integer to a bit vector
End For;
// construct compatibility graphs for each
// block pairs and then do clique partitioning
For each block pair Pi and Pj
//  Cij is the edge cut-set between Block Pi and Block Pj
        Cij = Get_Cut_Set(Pi,Pj);
        For each edge pair cij and cij' ; cij,cij' ∈ Cij and cij ≠ cij'
          if  ( PV(cij) & PV(cij') ≠ [ 0] ) then
// '&' is denoted as a bit-wise AND operation
                introduce an edge between Node cij and Node cij'
        End For;
// Get_Clique() is a clique partition algorithm proposed by
// Tseng and Siewiorek, and it returns the number of cliques
        Nq = Get_Clique(Cij);
        Buffer_Size = Buffer_Size + Nq;
// compute the buffer size for Cut-Set Cji
        repeat the above steps for Cji;
End For;
        Return Buffer_Size;
End Buffer_Size_Estimation.
```

Figure 5. Algorithm for Buffer Size Estimation.

4.3 Complexity Analysis

The buffer size estimation algorithm contains two major sub-algorithms. The first sub-algorithm is the Depth-First Traversal algorithm. The complexity of the Depth-First Traversal depends on the data representation of the CDFG. If the CDFG is represented as an adjacency list, which is adopted in our program implementation, the complexity is **O(n + e)**, where **n** is the number of nodes and **e** is the number of edges. Because **e** is usually greater than **n**, the complexity of Depth-First Traversal is often considered as **O(e)** [9].

The second sub-algorithm is the heuristic clique-partitioning algorithm proposed by Tseng and Siewiorek. Their algorithm comprises processes of selections of a pair of nodes with the maximum number of common neighbors and then merging the nodes as a super node. All the incident edges of the merged nodes are deleted, and new edges are added to their common neighbors. The algorithm repeats the processes until all the edges in the compatibility graph are deleted. For the worst case, in each iteration, there are **$C(n,2) = (n^2/2 - n/2)$** ways to choose a pair of nodes to merge, which happens when there are edges between every two nodes, i.e., a complete graph. Therefore, in each iteration, the complexity is **$O(n^2/2 - n/2)$,** which can be simplified to **$O(n^2/2)$, n** is the number of nodes. **n-1** iterations are needed for the worst case, when the compatibility graph is a complete graph, so that the complexity of the heuristic partitioning algorithm is **$O((n-1)n^2/2)$**. For **N**_way partitioning, there are **2C(N,2)** compatibility graphs needed to be established therefore **$2C(N,2) = (N^2 - N)$** is the number of times the clique partitioning algorithm is to be applied. The complexity becomes **$O((n-1)\ n^2/2(N^2 - N))$,** which can be simplified to **$O(n^3N^2/2)$.**

From the above complexity analyses, the complexity

for buffer size estimation algorithm can be expressed as $\mathbf{O(e + n^3N^2/2)}$, which allows us to estimate the buffer size in polynomial time.

5. Partition Using Problem Space Genetic Algorithm (PSGA)

Genetic algorithms (GAs) are powerful domain-independent search algorithms for solving optimization problems. In the GA approach, the solutions of the optimization problems are encoded as string of symbols called *chromosomes*. A number of possible solutions (*population)* co-exist. During each iteration or generation, a selection algorithm is used to choose *parents* (solutions) from the population to produce *offsprings*. Two operators are involved in the reproduction process, which are termed as *crossover* and *mutation*. A fitness function is used to evaluate the *fitness value* of each chromosome. The fitness values are used to choose the chromosomes from the population and the offsprings to form a new population for the next (generation) iteration. The more *fit* the chromosome, the higher the probability of it being selected. The chromosomes that are not chosen are discarded. The iteration process stops when a satisfactory or optimum solution is reached.

In prior genetic algorithm research [13], the chromosomes in the population are encoded directly as the solutions of the combinatorial optimization problem. One major disadvantage for this type of chromosome encoding is that after crossover and/or mutation operations, the generated solutions may not be feasible. PSGA takes an alternate approach by encoding the problem data not the solution data. The problem space information is used by a fast heuristic algorithm to map the problem information into solutions. The major advantage of PSGA is that crossover operator can be constructed easily to always produce feasible solutions.

5.1. Modeling and Implementation Technique

For optimization of the behavioral partitioning problem, a chromosome consists of two parts: (1) a list of integers representing the block to which each function in the CDFG is assigned, and (2) a list of integers representing the *work remaining* (WR) [8] for each function in the CDFG. A procedure is then provided to generate the initial population. The detailed chromosome representation and the initial population generation procedure can be found in Ref. [14].

For each chromosome, the buffer size is estimated by the Buffer_Size_Estimation algorithm. The system delay is estimated by using the List Scheduling algorithm that uses WRs (work remainings) as the priority function [4]. After the buffer size and the system delays are estimated, the costs for the chromosome is computed using Equation 3.

Crossover is an operation that selects two parent chromosomes from the population and produces two offsprings. The selection of parents from the population is based on the fitness values of the chromosomes. The higher the fitness value, the higher the probability of a chromosome being chosen for reproduction. The fitness values for each chromosome are calculated as follows:

$$f(i) = \frac{(Cmax - Ci + 1)^p}{\sum_{j=1,N} (Cmax - Ci + 1)^p} \qquad (4)$$

where $\mathbf{C_{max}}$ is the maximum cost in the population, $\mathbf{C_i}$ is the cost for chromosome **i** , **N** is the population size, and **p** is a parameter used to determine the selectivity of the fitness function [8]. The plus 1 in the denominator is needed for preventing a division by 0 error if all the members in the population converge to an identical chromosome.

After all the fitness values in the population are computed, the Roulette Wheel Algorithm [6] is used to select the chromosomes for crossover operation. The crossover operator selects two chromosomes, **Mm** and **Mf**, and randomly generates a cut point **i**. The first offspring is the concatenation of **Mm(1:i)** and **Mf(i+1,|V|),** and the second offspring is the concatenation of **Mf(1:i)** and **Mm(i+1,|V|).**

The **mutation** operator selects a small percentage, usually less than 5% of the offsprings and changes their block mappings and *work remainings.* The block change is re-mapping the randomly selected functions into another block. The work remaining change is re-assigning randomly selected functions new work remainings.

The partitioning procedure using PSGA technique can be summarized as: **a)** generate initial population (parents), **b)** apply crossover and mutation operators to generate offsprings, **c)** construct a new population by selecting the chromosomes from parents and offsprings, **d)** iterate steps **b** and **c** until the stop criteria is met, **e**) return the solution by selecting the chromosome with the highest fitness value.

6. Experimental Results

The buffer size driven partitioning algorithm is implemented on a SUN SPARC workstation using C++ programming language. To assess the results obtained using the PSGA algorithm, the simulated annealing (SA) version of the buffer driven partitioning algorithm is also implemented. The test cases used to evaluate the algorithm performance are derived from Electromagnetic

Field Theory, Digital Signal Processing, and Image Processing. The number of functions/nodes range from 19 to 241 and the number of edges range from 32 to 240. For PSGA, the population size is set to 50, and the mutation rate is set to 5%. For SA, the starting temperature is computed using the algorithm proposed by [15], the cooling parameter is set to 0.95, with 50 iterations at each temperature.

In this experiment, the weights α and β are set to 1 and 10 respectively. We also set up computation time limits for PSGA and SA in each case. The results are shown in Table 1, in which **n** denotes the number of functions/nodes in the CDFG, **N** the number of partitions, and **m** denotes the size of edge cut-sets. The delay indicates system delays and the numbers are derived from Intel Arithmetic Processor Unit 8231. The results show that PSGA finds better solutions in most of the cases with the same time limits. The table also shows that the buffer sizes are smaller or equal to the edge cut-sets in all the test cases, and some cases show that the buffer sizes are as low as 50% of the sizes of the edge cut-sets.

			PSGA			SA			
CDFG	n	N	m	buffer	delay	m	buffer	delay	Time
E-field superposition	19	2	4	3	86	4	4	85	10 (sec)
		4	6	5	59	13	8	62	10(sec)
Elliptical Wave Filter	34	2	19	8	311	13	6	311	20(sec)
		4	23	10	251	25	12	251	20(sec)
Fresnel Transition Function	20	2	3	3	101	3	3	107	10(sec)
		4	5	5	81	11	10	81	10(sec)
Wedge Diffraction Coefficient	150	2	90	45	664	81	45	711	12(min)
		4	140	81	426	141	83	445	27(min)
Edge Detector	241	4	163	69	184	165	79	192	5(hr)
		8	187	87	150	196	100	160	8(hr)

Table 1. Comaprison of PSGA and SA for partition results.

7. Conclusions

We have presented a new partitioning algorithm that is based on the Problem Space Genetic Algorithm which is an efficient way to search optimum solutions for NP-hard problems. The partitioning algorithm uses buffer size estimation and system delay to evaluate the qualities of the solutions. The proposed buffer size estimation algorithm starts from searching the paths in the CDFG. The edges of CDFG are then labeled with path vectors, and the path vectors are used to construct compatibility graphs. The compatibility graph is then input to a fast heuristic partition algorithm to find the upper bound on the buffer size. The experimental results show that using buffer size estimation as part of the cost function can obtain lower communication costs than using the size of edge cut-set in the final partitions if the communication is through buffer channels and the cost is estimated by the buffer size. PSGA results are compared with those produced by simulated annealing.

References

[1]Agrawal, S. and Gupta R. K., "Data-flow Assisted Behavioral Partitioning for Embedded Systems," *34th DAC*, 1997, 709-712.

[2] R.K. Gupta, C. Coelho, and G.D. Micheli, "Synthesis and Simulation of Digital Systems Containing Interacting Hardware and Software Components," *29th DAC*, 1992, 225-230.

[3] T. Amon and G. Borriello, "Sizing Synchronization Queues: A Case Study in Higher Level Synthesis", *28th DAC*, 1991,690-693.

[4] Gajski, D. D., *High-Level Synthesis: Introduction to Chip and System Design*, Norwell, MA: Kluwer Academic Publishers, 1992, 213-294.

[5] Kernighan, B.W.and Lin, S. "An efficient heuristic procedure for partitioning graphs," *Bell System Technical Journal, vol. 49*, pp. 291-307, Feb. 1970.

[6] Goldberg, D.E. *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Wesley, USA, 1989

[7] Holland, J.H., *Adaptation in Natural and Artificial systems*. Ann Arbor, MI: University of Michigan, 1975.

[8] Storer, R.H., Wu, D.S., and Vaccari, R., "New search spaces for sequencing problems with application to job shop scheduling," *Management Science, 38(10)*, 1992, pp. 1495-1509.

[9] A.M. Tanenbaum, Y. Langsam, and M.J. Augenstein, *Data Structures Using C*, Prentice-Hall, Englewood Cliff, N.J., 559-560.

[10] C.J. Tseng and D.P. Siewiorek, "Automated Synthesis of Data Path on Digital Systems," *IEEE trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. CAD-5, no 3,379-395,July 1986.

[11] Micheli,G., *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, New York, NY , 93, 63-67.

[12] Garey, M.R. and Johnson, D.S. *Computers and Intractability: A guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.

[13] Sait, S.M. et al "Scheduling and allocation in high-level synthesis using stochastic techniques," *Microelectronics Journal, vol.27*,1996, pp 693-712.

[14] Lin, T-C,. Sait, S.M., and Cyre, W.R, "Performance and Interface Buffer Size Driven Behavioral Partitioning for Embedded Systems," *IEEE 9th RSP*,1998, pp 116-121.

[15] Wong, D.F and Liu, C.L., "A new algorithm for floorplan design," *23rd DAC*, 1986,101-107.

Acknowledgement

The second author acknowledges King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for all support.