Performance and Interface Buffer Size Driven Behavioral Partitioning for Embedded Systems

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

Sadiq M. Sait
Computer Engineering Dep.
King Fahd University of Petroleum and Minerals
Dhahran-31261, Saudi Arabia
Sadiq@ccse.kfupm.edu.sa

Walling R. Cyre
Electrical Engineering Dep.
Virginia Tech
Blacksburg,VA 24061

Abstract

One of the major differences in partitioning for co-design is in the way the communication cost is evaluated. Generally the size of the edge cut-set is used. 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. Experimental results indicate that, according to a proposed quality metric, our approach can attain an average 87% of the optimum for two-way partitioning.

1. Introduction

For hardware/software co-design, one of the important and difficult problems is to partition the given system specifications into software, which is to be executed by microprocessors, and in hardware, which is to be executed by a co-processor (custom designed ASICs or synthesized FPGAs). One major difference in various partitioning approaches is in the way the communication cost is evaluated. The most common way is taking the size of edge cut-set between partitions as the communication cost [8]. However, the communication between two components is normally through buffered channels [2]. The size of the edge cut-set is not accurate enough to estimate the buffer size. As a matter of fact, the buffer size is in general smaller than the values (i.e., edges) transferred among the blocks of a partition. Another important factor to measure the quality of the partitioning result is the system delay. Most of the partitioning approaches [5] set up the number of the nodes/functions in each partition as the constraint for partitioning, and then try to minimize the communication cost. The effects of the data dependency among these nodes/functions, the delays of each functions, and the sequences of executing the functions are not considered. These effects have a great impact on the system performance. For example, in Figure 1, 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 dependency among the functions and the blocks of the functions assigned to.

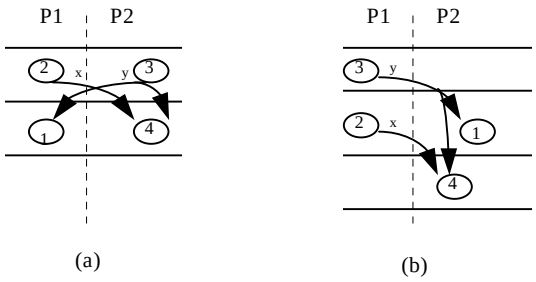


Figure 1. Partition results with different system delays

In this paper, behavioral-level system specification is described using hardware description language, such as VHDL. The VHDL code is then parsed and translated into a graphic representation, which is called Control Data Flow Graph (CDFG). The CDFG is used as the input for our partition algorithm. We propose two optimization objectives to measure the quality of the partition. One is the buffer size, and the other is the system delay. The buffer size is estimated by analyzing

the data flow patterns of the CDFG, which is obtained by tracing the paths in the CDFG. The buffer size estimation is transformed into a Clique Partition problem [13]. The system delay is estimated by using List Scheduling [5]. These two quality measures are formulated as a weighted-sum cost function, and we try to find a partition that requires a minimal cost by using a type of the genetic algorithms called Problem Space Genetic Algorithm (PSGA) [4,7]

2. Previous Related Work

The min-cut partitioning algorithm [8] 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 measurement, 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 a NP-complete problem [5]. Approaches that have been proposed by previous researchers are usually based on constructive heuristics such as the min-cut algorithm [8], or some non-deterministic hill-climbing algorithms such as the simulated annealing [9]. 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 [5]. Another non-deterministic optimization algorithm called the Genetic Algorithm has been applied successfully in numerous research areas [6].

Problem Space Genetic Algorithm was first proposed by Storer *et al.* [12]. 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 use to map the problem space into the solution space, which guarantees that the solutions are always feasible. This approach was first adopted by Dhodhi *et al.* [4] to optimize the datapath in a high-level synthesis environment.

3. Problem Formulation

The Control Data Flow Graph G = (V, E), which is used to describe the system behaviors, 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 an input for 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 $Vi \cap Vj = \emptyset$ if $i \neq j$, 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 a compatibility graph. The compatibility graph is then used to find the upper bound on the buffer size.

4.1 Path Vectors

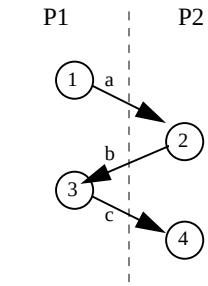
Labeling the edges with path vectors is used to detect the variables with non-overlapping lifetimes. The variables with non-overlapping lifetimes can share the same register in a buffer, which leads to buffer size reduction. For example, in Figure 2, Functions **1** and **3** are assigned to **P1** and Functions **2** and **4** are assigned to **P2**, Variables **a** and **c** are non-overlapping because the data dependency among Functions **1**, **2**, **3,** and **4**. Variables **a** and **c** 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 each entry nodes, which are the nodes without predecessors, as the roots, and then perform Depth-First Search. For example, in Figure 3 (a), the roots for this CDFG are Nodes 1, 2, 3, 4, and 5. 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., [00001] is used to represent Path1. 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 **f** and **h** are traversed by Paths **1** and **2** so that Edges **f** and **h** are labeled with a path vector [00011]. The formal notaiton 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.

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 **Pi** and **Pj,** the edge cut set, which is used to represent the variables generated by the functions in **Pi** and consumed by the functions in **Pj,** 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:



Edge Cut-Set $C_{12} = \{a,c\}$
$C_{21} = \{b\}$

Figure 2. Edge Cut-sets with non-overlapping lifetimes

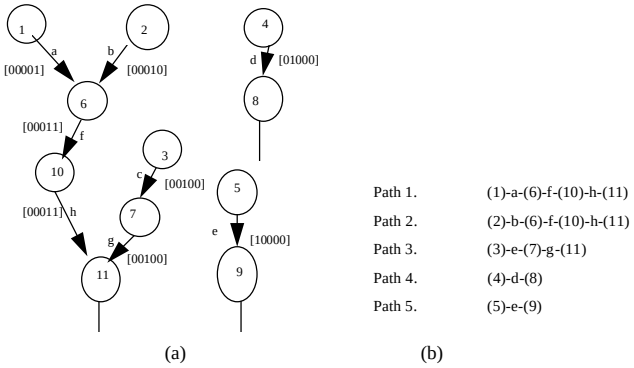


Figure 3. CDFG labeled path vectors

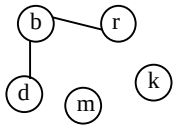
- For every edge pair $\mathbf{(c_{ij}, c_{ij}')}$, where $\mathbf{c_{ij}, c_{ij}' \in C_{ij}}$ and $\mathbf{c_{ij} \neq c_{ij}'}$,
- If $\mathbf{PV(c_{ij})}$ AND $\mathbf{PV(c_{ij}') \neq 0}$, then introduce an edge between $\mathbf{c_{ij}}$ and $\mathbf{c_{ij}'}$.

For example, Figure 4(a) shows the edge cut-set and the path vectors for $\mathbf{C_{ij}}$. The compatibility graph for $\mathbf{C_{ij}}$ is shown in Figure 4(b).

$C_{ij} = \{ b,d,m,r,k\}$

PV(b) = 0000 0110
PV(d) = 0000 0100
PV(m) = 0100 1000
PV(r) = 0000 0010
PV(k) = 1001 0000

(a)



(b)

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

The buffer size upper bound for $\mathbf{b_{ij}}$ 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 [5]. An effective heuristic algorithm proposed by Tseng and Siewiorek [13] is used to find the minimal number of cliques for our buffer size estimation algorithm. Analogously, the buffer size for $\mathbf{b_{ji}}$ can be obtained by applying the above procedure, and the buffer size upper bound for Partition **i** and **j** is equal to $\mathbf{b_{ij}+b_{ji}}$. 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 (2).

5. Problem Space Genetic Algorithm for Behavioral Partitioning

5.2. Problem Space Genetic Algorithm (PSGA)

Genetic algorithms (GAs) [6] are powerful domain-independent search algorithms for optimization problems. In prior genetic algorithm research [10, 11], 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. For example, in the high-level synthesis scheduling problem [5], a function cannot be scheduled before its predecessors are scheduled. If the chromosomes carry directly the control steps in which the functions are scheduled for execution, infeasible solutions will be generated after crossover and/or mutation if a function is scheduled before its predecessors were scheduled. The infeasible solutions have to be fixed or discarded.

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 advantage of PSGA includes (1) Crossover operator can be constructed easily without providing the technique of fixing the infeasible solutions. (2) An existing fast heuristic algorithm can be selected to map the problem space to the solution. The existing problem specific heuristic algorithm has already gained-insight knowledge of the problem. Embedding this heuristic algorithm within GA is equivalent to incorporating problem specific knowledge into the search for the optimal solution.

5.3. Problem-Space Genetic Behavioral Partitioning (PSGBP)

The implementation of the behavioral partitioning algorithm, which is based on the PSGA, is presented in this section. For illustration purposes, a CDFG used for modeling the superposition of reflected and incident uniform plane electromagnetic waves is shown in Figure 5 [3].

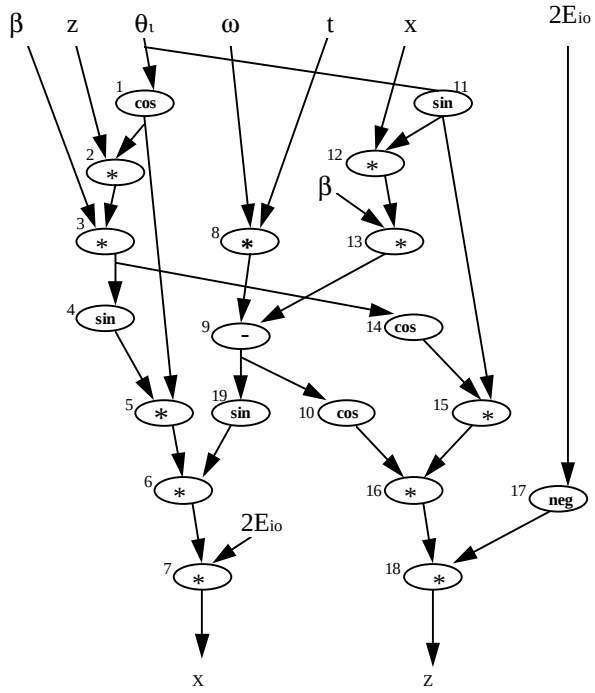


Figure 5. DFG for the electromagnetic wave superposition model

5.3.1 Chromosome Encoding and Initial Population

For optimization of the behavioral partitioning problem, a chromosome consists of two parts (see Figure 6): (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) [12] for each function in the CDFG.

Suppose a two-way partitioning is to be performed, the initial chromosome in the population is constructed as follows. Each function is randomly assigned to one of the two blocks. For example, in Figure 6 the first part of the , 13, 14, 15, and 17 are assigned block 0 and Functions 5, 8, 10, 16, 18, and 19 are assigned to block 1. Suppose the execution delays for floating point addition is 2 clock cycles, multiplication is 1 clock cycle, cosine is 24 clock cycles, and sine is 26 clock cycles

(these numbers are taken from Intel arithmetic processor unit 8231 and are normalized against multiplication), then the second part of the chromosome shows the *work remaining* of each function. *Work remaining* is a term borrowed from operation research and is used to indicate the minimum time required to complete all the jobs [12].

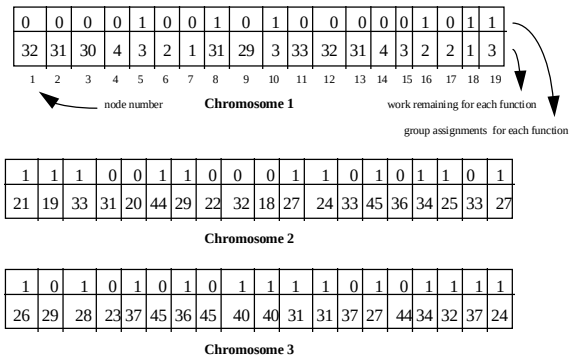


Figure 6. Chromosome representation

For the rest of the chromosomes, the block parts are generated using the same technique as that applied to the first chromosome. The work remaining parts are generated using the following formulae [4]:

$$\mathbf{WR_i = WR_{max} + \delta_i} \qquad (4a)$$

$$\mathbf{\delta_i = [-a,a]\ ,} \qquad (4b)$$

where $\mathbf{WR_i}$ is the work remaining for Function **i** in a chromosome, $\mathbf{WR_{max}}$ is the maximum work remaining in the initial chromosome, and **a** is equal to $\mathbf{WR_{max} / 2}$. $\mathbf{\delta_i}$ is a randomly generated integer number that is between **-a** and **a**. Figure 6 shows examples of the second and the third of the chromosomes produced for the CDFG shown in Figure 5. The work remaining is used by the list scheduling algorithm as the priority function [5]. List scheduling is a fast heuristic scheduling algorithm with computational complexity of **O(n),** where **n** is the number of functions in the CDFG. After assigning the functions into different blocks and then applying list scheduling, the costs for each chromosome can be obtained by using Equation 3.

5.3.2 Crossover

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:

$$f(i) = \frac{(Cmax - Ci + 1)^P}{\sum_{j=1,N} (Cmax - Ci + 1)^P}, \qquad (5)$$

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 [12]. 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|).**

5.3.3 Mutation

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. Equation 4 is used to compute values for the new work remainings.

6. Experimental Results

PSGBP algorithm is coded using C++ and compiled using GNU g++. The run-times are in the order of minutes on SUN UltraSparc Workstations for all of the test cases used. Two CDFGs that are derived from electromagnetic field and digital signal processing are used to evaluate PSGBP. Eight artificially generated CDFGs, whose optimal solutions are known, are also used to determine the best solutions PSGBP can achieve for a given memory constraint, i.e., the population size, and a time constraint (that is, the number of generations). The complexities of these CDFGs used range from 19 nodes to 110 nodes.

The experimental results for Two- and Four-way partitioning are shown in Table 1. The population size, Np, is set to 30, and the number of generations, Ng, is

set to 1000. The mutation probability is set to 5%. The weights for α and β are set to 1 and 5 respectively.

CDFG	Number of Partitions	Optimal				PSGBP				
		Number of Nodes	Communication (buffer size)	System Delay	Cost	Communication (buffer size)	System Delay	Cost	Quality
wave superposition	2	19	-	-	-	2	84	94	-
elliptic digital filter	2	33	-	-	-	6	301	331	-
gh_1	2	24	0	121	121	0	121	121	100 %
gh_2	2	48	0	241	241	5	241	266	89.6%
gh_3	2	96	0	481	481	12	481	541	87.5%
ph_1	2	22	0	275	275	2	275	285	96.3%
ph_2	2	66	0	823	823	11	825	880	93.0%
ph_3	2	110	0	1371	1371	12	1375	1435	95.6%
ph_4	4	44	0	275	275	9	275	320	83.6%
ph_5	4	88	0	549	549	31	553	728	71.0%

Table 1. Comparing the results of PSGBP with optimal solutions

Because the optimal solutions for the first two CDFGs are unknown, the costs and Q do not apply (indicated by "-"). CDFG gh_1 to gh_3 and ph_1 to ph_5 are generated manually, and optimal solutions for them are known. In Table 1, *buffer size* is used to indicate the buffer size required for the interface communication, *system delay* is used to indicate the execution time required for the given CDFG, and *cost* is calculated using Equation 3. From the results shown in Table 1, For Two-way partitioning, all the test cases can reach about 87% of the optima.

Table 2 shows comparison of the buffer size for the final partition by using the size of edge cut-set, or, the estimated buffer size, as part of the cost function. For each test case, the initial populations are set to be the same for comparison. The population size is set to 30, and the number of generations to 500.

CDFG	Total buffer cost of finial partition	
	use **size of edge cut-set** as part of cost function	use **estimated buffer size** as part of cost function
wave superposition	4	2
elliptic digital filter	7	6
gh_2	9	7
ph_2	9	9

Table 2. Buffer size comparisons

The results show that using estimated buffer size as part of the cost function can achieve smaller buffer sizes in the final partitions in three test cases and the same buffer size in one test case, which implies using estimated buffer size instead of the size of edge cut-set as part of the cost function can attain lower communication costs and better partition results.

6. Conclusions

We have presented a new partitioning algorithm that is based on the Genetic Algorithm which is an efficient way to search optimum solutions for NP-complete problems. The partitioning algorithm also 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 of the buffer size. The experimental results show that our partition algorithm can achieve solution qualities ranging from 83 to 100% of optimal for Two-way partitioning. Using buffer size estimation as part of the cost function can also obtain lower communication costs than using the size of edge cut-set in the finial partitions.

References

[1] Agrawal, S. and Gupta R. K., "Data-flow Assisted Behavioral Partitioning for Embedded Syatems," *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] Cheng, D.K., *Field and Wave Electromagnetics*, Addison-Wesley,Reading,MA 1989,390-396.

[4].Dhodhi, M.K et al, "Datapath Synthesis Using a Problem-Space Genetic Algorithm," *IEEE trans. on CAD, 14(8)*, 1995, pp.934-943.

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

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

[9] S.Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing," *Science 220(4598)*, 1983, pp. 671-680.

[10] Ravikumar ,C.P. and Gupta A.K., "Genetic algorithm for mapping tasks onto a reconfigurable parallel processor," *IEE Proc-Comput. Digit. Tech, 142(2)*, 1995, pp.81-86.

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

[12] 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.

[13] 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.