Formulation and Evaluation Of Scheduling Techniques For Control Flow Graphs

Maher Rahmouni **Ahmed A. Jerraya**

Laboratoire TIMA/lNPG,
46, Avenue Felix Viallet,
38031 Grenoble Cedex, France
Email:rahmouni@verdon.imag.fr

Abstract

This paper presents a theoretical basis for scheduling approaches based on purely control-flow graphs. This formulation includes a control flow graph model based on a finite discrete-time homogeneous Markov chain suitable to repesent complex control structures. A probabilistic finite state machine is introduced to model the resulting schedule and evalute the effectiveness of the scheduling approaches for control flow graphs. The need of such models is imposed by the nature of real time systems in which the control sequence depends on external conditions.

1 Introduction

High level synthesis means going from a functional specification at the algorithmic level of a digital system to a register transfer level structure. The first step in high level synthesis is to derive an internal graph-based representation equivalent to the algorithmic description for both data flow and control flow. We then perform high level synthesis tasks such as scheduling and allocation. The scheduling problem can be described as determining in witch control step each operation or set of operations is going to take place. Most Scheduling algorithms use data flow graphs[7], this type of representation is well understood and the corresponding scheduling called Data Flow Based Scheduling(DFBS) is well formulated. The most popular DFBS techniques are List Scheduling[9] and Force Directed Scheduling[6]. Data flow graphs are the most convenient represenations for behavioural descriptions representing systems repeatedly performs a series of operations on an infinite data stream, but it's not sufficient to represent designs in which the control sequence is based on external conditions. One of the main interesting models which supports such properties is the Constraint Graph proposed by DeMicheli[1], it consists of a polar hierarchical acyclic graph where the vertices represent operations and the edges represent the dependencies among the operations. The hierarchy supports procedure calls, conditional branching and loop iterartions. The sheduling algorithm used within is the relative scheduling, the strengh of this approach is its capacity to treat operations with fixed as well as unbounded delays. Its weakness is the hierarchy which makes it inefficient to schedule accross loops and conditional constructs. In these cases, purely control flow graphs may be more efficient. Thus, we can have algorithms formed on (Control Flow Based Scheduling(CFBS))[2, 3, 4, 5].

In this paper, we build a theorical basis to model all CFBS algorithms. This basis contains a formulation of these scheduling approaches and uses a probabilistic model to compute a cost function allowing an evaluation and comparison between the different algorithms.

This paper is organized as follows. Section 2 describes the control flow graph model and present the basic concepts for CFBS. In Section 3, the formulation of path based approaches according to these concepts is done. In section 4, we use the control flow model based on finite discrete-time homogeneous markov chains and the probabilistic FSM model to evaluate the CFBS approaches. Conclusions and perspectives are given in section 5.

2 Basic Concepts and Definitions

For the sake of clarity, we will take as an example the algorithmic description of the computation of the function **ab mod n**[11], given $0 \leq a, b \leq n$, and $\lg(n) \leq 15$. This is shown in figure 1(a).

2.1 Control flow Graph

Control flow graphs are the most suited representation to model control design, which contain many(possibly nested) loops, global exceptions, multiple wait on events and procedure calls; in other words, features that reflect the inherent properties of controllers.

Definition 2-1:(Control flow Graph)
A control flow graph CFG is a graph $G = (V, E)$, where :

(i)$V = \{v_1, \ldots, v_n\}$ is a finite set whose elements are nodes, and
(ii) $E \subset V \times V$ is a control flow relation, whose elements are directed sequence edges.

The CFG corresponding to the function **ab mod n** is shown in figure 1(b).

Definition 2-2:(Graph node)
Nodes are of two classes:

- operation nodes such as assignments, logic, arithmetic operations and procedure calls, represented by the subset of V, Vo. For the example of figure 1,
$Vo = \{v_1, v_2, v_5, v_7, v_8, v_9, v_{10}, v_{12}, v_{13}\}$.
- branch nodes modelling conditional statements such as If, Case and Conditional loops, represented by Vb. For the example of figure 1, $Vb = \{v_3, v_4, v_6, v_{11}\}$.

Thus $V = Vo \cup Vb$
The operation nodes have only one successor, while branch nodes have more than one successor.

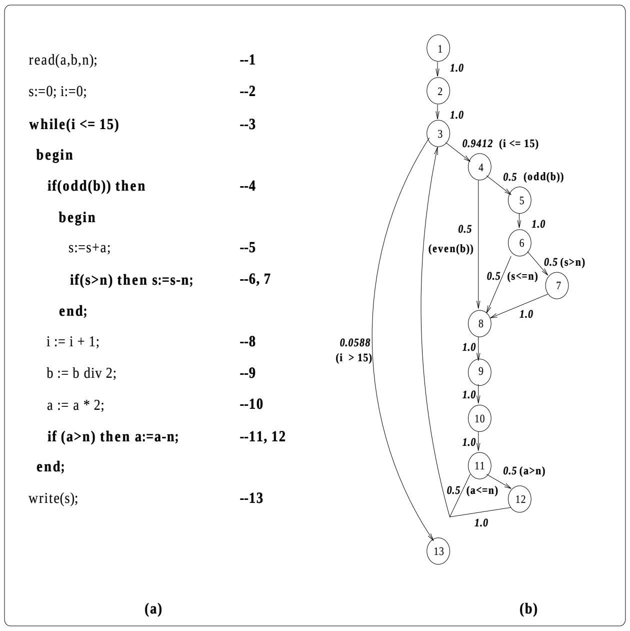
Definition 2-3:(Graph edge)
An edge represents the precedence relation between two nodes v_i and v_j. An edge e_{ij} is weighted by a condition $Cond_{ij}$ and a probability p_{ij}, meaning that the operation represented by v_j will be executed with a probability p_{ij} if v_i is executed and $Cond_{ij}$ is evaluted to True. These assumptions mean that the Control Flow Graph is a finite, discrete-time, homogeneous Markov chain[10]. For this reason the nodes have to satisfy the following two properties:

Property 1: if v_i is an operation node and v_j is an immediate successor of v_i, i.e $(v_i, v_j) \in E$ then: $Cond_{ij} = True$ and $p_{ij} = 1$.
This property assume that the probability of going from the operation represented by v_i, o_i, to that represented by v_j is independent from the operations executed before o_i.

Property 2: if v_i is a branch node, and $(v_{i1}, \ldots, v_{ik})$ are the k immediate successors of v_i then:

(i) $Cond_{i,im} \wedge Cond_{i,ih} = 0$, for all $m, h \in \{1, \ldots, k\}$ and $m \neq h$, and $\sum_{l=1}^{k} Cond_{i,il} = 1$.

(ii) $\sum_{m=1}^{k} p_{im} = 1$.



```
read(a,b,n);                          --1
s:=0; i:=0;                           --2
while(i <= 15)                        --3
 begin
   if(odd(b)) then                    --4
     begin
       s:=s+a;                        --5
       if(s>n) then s:=s-n;           --6, 7
     end;
   i := i + 1;                        --8
   b := b div 2;                      --9
   a := a * 2;                        --10
   if (a>n) then a:=a-n;              --11, 12
 end;
write(s);                             --13
```

Figure 1: (a) Algoritmic description of *ab mod n* function. (b) CFG of the *ab mod n* function.

2.2 Control-flow Based Scheduling

These type of approaches have appeared for the problem of handling loops and conditional branches. The DFBS approaches for treating this problem is to schedule the innermost loop body and the corresponding test expression first, and then to proceed outwards treating the loop as a single node. Similarly, conditional branches cause optional program paths. So, the number of control steps may vary from one branch to another, however it's fixed for each branch. The DFBS approaches assume that all branches have the same length as the longest path. Path-based scheduling [2], remedies to all these problems considering all program paths and exploiting optimization potential concerning state assignement especially in conditional branches. To ease path extraction and analysis, PBS uses a CFG structure. As in the case of DFBS, CFBS may be restricted by constraints. Some are inherent to the implementation like the data dependencies, where registers can be written only once per control step, or I/O ports can be read or written only once per control step. Others are user constraints which are specified explicitly often to indicate area and delay characteristics of the target architecture.

Since CFBS are mainly Path-based approaches, we have to define scheduled paths from the CFG, and construct the FSM corresponding to the resulting schedule. Scheduled Paths represent implicitly a function to be executed in one clock cycle at a given time from the whole execution time of the description modelled by the CFG. In other words, the CFG will be partioned in a sequence of Scheduled Paths.

Definition 2-4:(Scheduled Path)

Let $G = (V, E)$ be a CFG, let $(C_1, \ldots, C_r)$ be a set of constraints. A Scheduled Path SP is a sequence of nodes $(v_1, \ldots, v_n)$, extracted from the CFG and satisfying all the constraints. If $(C_1, \ldots, C_r)$, the set of constraints, $\prec C_{k(k=1,\ldots,r)}(v_i, v_j) = 1$ this means that the constraint C_k is not satisfied between nodes v_i and v_j. Therefore, a scheduled Path has to satisfy the following property:

Property 3: for all $v_i, v_j \in SP$, for all $C_{k(k=1,\ldots,r)}$, $\prec C_k(v_i, v_j) = 0$.

We define for each Scheduled path four parameters:

- a header which is the first node of that path v_1, we note $Header(SP)$.
- a successor, which is the immediate successor of the last node in SP, and we note $Succ(P)$.
- a condition $Cond_{SP}$
- a probabiliy $Prob_{SP}$

Since the CFG is a finite discrete-time homgeneous Markov chain, the condition (probability) enabling the execution of SP, $Cond_P$ $(Prob_P)$ is derived by logically ANDing (multiplying) the conditions (probabilities) on the control-flow edges leading from Header(SP) to Succ(SP). Then the following property:

Property 4: Let $SP = (v_i, \ldots, v_j)$, then the condition $Cond_{SP}$ enabling the execution of SP with a probability $Prob_{SP}$ are derived as:

- $Cond_P(SP) = \wedge_{(i=1,\ldots,n-1),(j=i+1)} Cond(v_i, v_j)$ $\wedge\, Cond(v_n, Succ(SP))$
- $Prob_P(SP) = \prod_{i=1}^{n-1} p(v_i, v_{i+1}) \times p(v_n, Succ(SP))$

Definition 2-5:(CFG Headers)

For each CFG, we define the set of Headers $H(G)$, which represent the set of all the headers of scheduled Paths.

$$H(G) = \{v_i \in V \,|\, \exists SP \subset V, v_i = Header(SP)\}$$

Scheduling produces a finite state machine FSM, the FSM is modelled either by a Moore or a Meally automata.

Definition 2-6:(Control-flow Scheduling)

Let $\Psi = (S, I, O, f, g)$ be a transition-based FSM, a schedule of a CFG $G = (V, E)$ is a mapping:

$$\begin{array}{rccc} \sigma : & \mathrm{H(G)} & \longrightarrow & \mathrm{S} \\ & v_i & \longmapsto & s(t_i) \end{array} \quad \text{Where:}$$

- $I(t_i) = \{i_1(t_i), \ldots, i_M(t_i)\}$ $= \{Cond_P(SP_{ik})_{(k=1,\ldots,M)}$ such that $Header(SP_{ik}) = v_i\}$.
- $O(t_i) = \{o_1(t_i), \ldots, o_R(t_i)\} =$ $\{SP$ such that $Header(SP) = v_i\}$.
- $f(s(t_i), ik(t_i)) = \sigma(v_j)$ such that $v_j = Succ(SP_{ik})_{(k=1,\ldots,M)}$.
- $g(s(t_i), ik(t_n)) = SP_{ik}, (k = 1, \ldots, R)$.

The control flow based scheduling consists of merging all the scheduled paths with he same header into the same state, and a transition is made between two states S_i represented by v_i and S_j represented by v_j if and only if there is a scheduled path having as Header, v_i, and as successor, $Succ(SP) = v_j$. Then if the machine is currently in cycle step $s(t_i)$ and it's presented an input condition $Cond_P(SP)$, then it will change its control step to $f(s(t_n), I(t_n))$, and output the result of execution of all the operations in SP.

2.3 cost function

Data dependent loops in control flow graphs introduce a major problem for the cost function of the scheduling result, this is due to the unknown number of iterations for each loop. The number of states or transitions generated by the schedule do not reflect the real total execution time of an algorithm. So, we will use a probabilistic finite state machine to represent the resulting schedule.

Definition 2-7:(Probabilistic Finite State Machine)

a probabilistic FSM $\Psi = (S, I, O, f, g, \lambda)$ is an FSM with a probabilistic state transition function λ defined as:

$\lambda(s(t_k), s(t_l)) = \sum_{m=i}^{j} Prob(SP_m)$

such that $\{SP_i, \ldots, SP_j\}$ is the set of scheduled paths having as entry state $s(t_k)$ and as destination $s(t_l)$.

Definition 2-8: (Expected number of Clock Cycles Of aSchedule)

Let $\Psi = (S, I, O, f, g, \lambda)$ be the probabilistic FSM resulting from the Schedule. The expected number of

clock cycles needed to execute the correponding input behavioural description is: $[X_{sch} = \sum_{i=1}^{n} X_i$ where X_i is the random variable representing the expected number of times the state $s(t_i)$ is executed during an execution of the behavioural description. Computing X_i, $\forall i \in (1,\ldots,n)$ is equivalent to resolve the following set of linear equations:
$X_1 = 1$
$X_i = \sum X_j.\lambda(s(t_j), s(t_i))$
$\forall j$ such that a transition exist between $s(t_j)$ and $s(t_i)$

3 Formulation of CFBS Approaches

Around Path-based Scheduling[2], many heuristics are derived, to consitute a new generation of scheduling techniques oriented toward control-flow. The most well-known are Dynamic Loop Scheduling [3], Loop Directed Scheduling[5] and Pipeline Path-based Scheduling[4]. These approaches comes to resolve problems related to the initial path-based approach, such as its exponential complexity, or to add more efficiency to the way that loops are handled.

3.1 Path Based Scheduling

In this approach, the CFG is made acyclic by removing the feedback edges in loops, and saving the first and the last nodes of a loop. The second step is the computation of all paths, starting from the first node of the Acyclic CFG and then all first nodes of loops. Then each path is scheduled independently with respect to the constraints using a clique covering method. This step consists of partitionning each path on a set of scheduled paths. Finally, the finite state machine is built.
Definition 3-1: (Path) Let $G = (V, E)$ an Acyclic CFG, A Path P is a sequence of nodes $(v_1, \ldots, v_n)$, where:

(i) v_1 is either the first node of the graph or the first node of a loop.
(ii) v_n is a node with no successor.

Each path $P = (v_1, \ldots, v_n)$ will be scheduled independentlty, and a set of constraints $\{C_1, \ldots, C_r\}$ are computed for each path. A constraint C_k correponds to an Interval $I_{C_k} = \{v_i, v_{i+1}, \ldots, v_j\}$, Such that $1 < i < j \le n$, this means that constraint C_k is violated between nodes v_{i-1} and v_j.
Figure 2 shows the acyclic CFG and the set of paths from the CFG example of figure 1. to simplify, the only constraint considered is data dependency.
Definition 3-2: (Scheduled Path)

Let $P = (v_1, \ldots, v_n)$ a path, Let $C_1, \ldots, C_r$ a set of constraints. A Scheduled path SP (in Path-based Scheduling) is a sequence of nodes $(v_i, v_{i+1}, \ldots, v_j)$ where:

(i) $1 \le i < j \le n$.
(ii) $\exists \{C_m, \ldots, C_k\}$ such that
$$Succ(SP) \in I_{C_m} \cap I_{C_{m+1}} \cap \ldots \cap I_{C_k}$$

Since feedback edges are removed, Scheduled paths satisfying the following conditions will be added:

•$\forall$ $SP = (v_i, v_{i+1}, \ldots, v_j)$ such as $\exists$ $v_l \in SP$, $i \le l \le j$ and v_l is the last node of a loop.
A scheduled path $SP' = (v_i, \ldots, v_l)$ is added to the set of scheduled paths.
•$Succ(SP') = v_f$ such as v_f is the first node of the loop.

The scheduled paths and the FSM for path based scheduling are shown in figure 2(c) and figure 2(d).

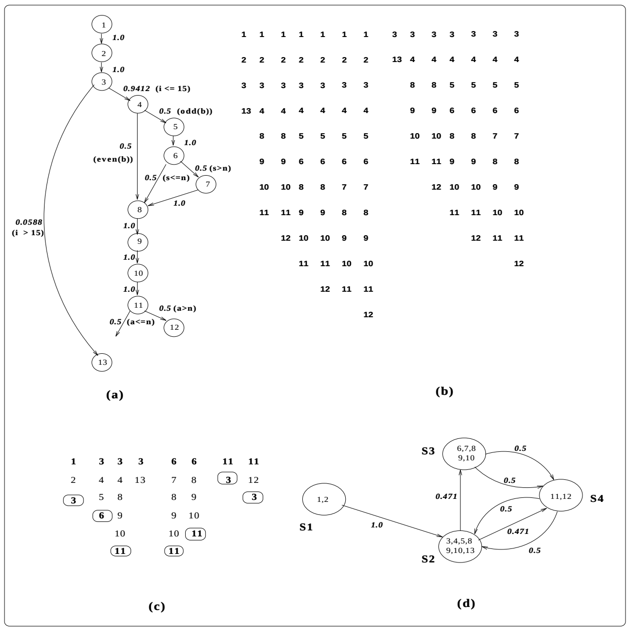


Figure 2: (a) Acyclic CFG, (b) Paths, (c) Scheduled Paths, (d) FSM for Path Based Scheduling

3.2 Dynamic Loop Scheduling

DLS uses a CFG, by keeping all loop feedback edges in the control-flow graph, thereby allowing the parallel execution of parts of successive loop iterations. DLS interrupts the generation of paths *on the fly* , in other words, the generation of a current path is stopped if a constraint is violated. This reduce the complexity of the number of generated paths.

Definition 3-3: (Scheduled Path)
Let G=(V,E) a CFG and $C_1, \ldots, C_r$ a set of constraints. A scheduled path SP (In Dynamic Loop Scheduling) is a sequence of nodes $(v_i, v_{i+1}, \ldots, v_j)$ where:
•$\exists\ v_l$, C_m such that $i \leq l \leq j$ and $C_m(v_l, v_{j+1}) = 1$.
This approach schedule paths As Late As Possible. Figure 3 shows the scheduled paths and the FSM for DLS.

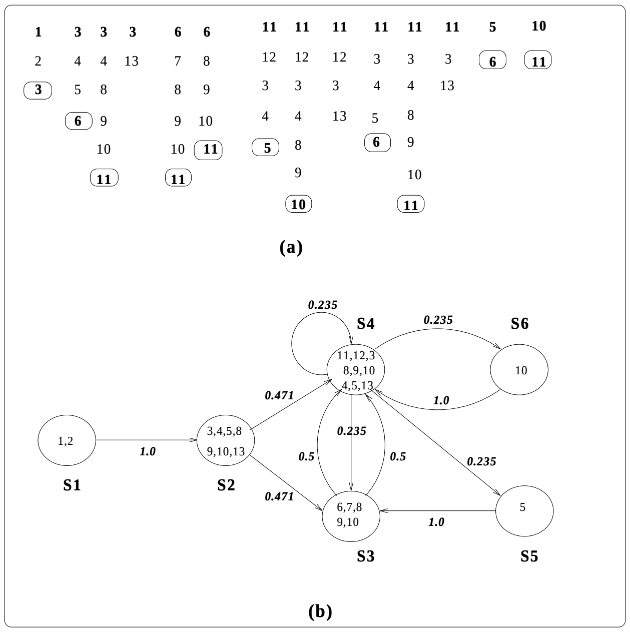


Figure 3: (a) Paths and successors, (b) FSM for the Dynamic Loop Scheduling

3.3 Pipeline Path-based Scheduling

Pipelined Path-based Scheduling (PPS)[4] is another transformed path-based approach. It's main strength is the opimisation of loops, since loops are usually the most time critical part of an application. It considers that a given loop will be executed 0, 1 or 2 or more times. If there are no loops, the paths are generated in accordance with the PBS approach. This algorithm differs from the original path-based approach in the way of computing paths. It starts from the fisrt node in the control flow graph and ended with a node with no successor. In the presence of a loop, it generates a set of paths where the loop is unrolled one time or twice. Figure 4(a) shows one possible path in which the loop is executed twice. The subscripts correspond to the loop iteration from which the node

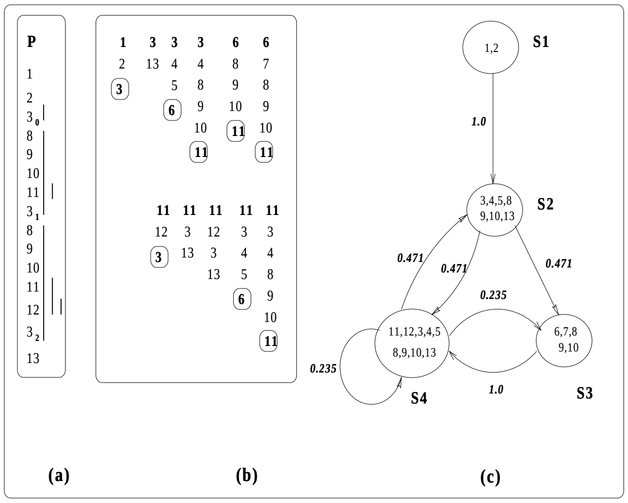


Figure 4: (a) One path and cuts, (b) Paths after cuts, (c)FSM for PPS.

is taken. Thus, 3_0 is node 3 for zero loop iterations (the loop condition is false), 3_1 is node 3 of the first iteration of the loop and node 3_2 is node 3 of the second iteration. The scheduled paths and the FSM for PPS are shown in figure 4. More details about this approach can be found in[4].

4 Evaluation and Comparaison Of CFBS Algorithms

PBS generates an As Fast As Possible (AFAP) schedule for a description containing many different possible execution paths. This is achieved through a complex clique covering technique that identifies the minimum number of cuts necessary for all paths in order to satisfy the constraints (user-imposed or data-dependent). This approach however, tends to be sub-optimal when the input description contains many loops. The problem is related to the fact that, in Camposano's approach, all loop feedback edges are broken and thus no advantage can be taken of the fact that different loop iterations can be pipelined. DLS attempt to overcome this problem by leaving loop feedback edges intact. However,their approach is rather simplistic as they only consider one iteration. In addition, they do not cut the generated paths in an optimal way as they use an As Late As Possible (ALAP) scheduling technique to do this. Nevertheless, results published for these algorithms show that by treating loops more efficiently, improvements on the original path-based approach can be made, even taking into consideration the fact that the path cuts are not optimal. PPS benefit from the advantages offered by these two approaches. It uses a clique covering technique to cut the paths in an optimal fashion while at the same time, using a new technique for pipelining loop iterations in order to identify any parallelism that may exist beyond loop boundaries. Figure 5 shows the

Algorithm	**State**	**Transition**	**Clock_Cycle**
PBS	4	**8**	42.18
DLS	6	14	33.5
PPS	**4**	11	**32.55**

Figure 5: Results for the *ab mod n* algorithm

different results for the three approaches, PBS, DLS and PPS. *State* means the number of states, *Transition* the number of transitions and *Clock_Cycle* the expected number of clock cycles needed to execute the algorithm. When looking at the FSMs produced by PBS and DLS, we see that the one produced by DLS is more complex and contains more states than that of PBS. Nevertheless, using DLS we kept 8 clock cycles for the entire execution of the *ab mod n* algorithm. This due to the fact that PBS will always take at least two states to execute the loop. However, under certain conditions, DLS can take one state(S4).By combining the advantages of both approaches, PPS minimizes cuts the number of states and tansitions. Moreover, by unrolling loops it can minimize the number of clock cycles.

5 Conclusion

In this paper, we have presented a definition of Control Flow Based Scheduling and a formulation of popular approaches. This formulation includes a CFG model based on the finite discrete-time homegeneous markov chain and the probabilistic FSM corresponding to the scheduling result, and a cost function used to compare these algorithms which perform scheduling of data dependent loops. These concepts give us the opportunity to derive other CFBS heuristics, by generating differently scheduled paths. Our perspectives concentrate on the migration of Data flow based optimization techniques applied within Control flow Based approaches.

References

[1] D.C.Ku, G. DeMicheli, *"Relative Scheduling under timing constraints"*, IEEE Trans. on CAD/ICAS May 1992.

[2] R. Camposano, *"Path-Based Scheduling for Synthesis"*, IEEE T. CAD, Vol 10(1), pp85-93, January 1991.

[3] K. O'Brien, M. Rahmouni, A.A.Jerraya, *" A VHDL-Based Scheduling Algorithm for Control-Flow Dominated Design"*, 6th Intl. Workshop On High-Level Synthesis, California, November 1992.

[4] M. Rahmouni, A.A.Jerraya, *"PPS: A Pipeline Path-Based Scheduler"*, Proc. European Design and Test Conference'95, March 1995.

[5] S. Bhatacharya, S. Dey, F. Brglez, *"Performance Analysis and Optimization of Schedules for Conditional and Loop Intensive Specifications"*, 31st Design Automation Conference, pp. 491-496, 1994.

[6] P.G Paulin, J.P Knight, *"Force-Directed Scheduling for te behavioral Synthesis of ASIC's"*, IEEE Trans. on computer-Aided Design, vol. 8, no.6, June 1989.

[7] L. Stock, *"Architectural Synthesis and Optimization of Digital Systems"*, Ph D Thesis, 1991.

[8] P. Michel, U. Lauther, P. Duzy, *"The Synthesis Approach to Digital System Design"*, Kluwer Academic Publishers, 1992.

[9] Pangrle,B.M. and D.D Gasjki, *"Slicer: a state synthesizer for intelligent silicon compilation"*, Digest of the IEEE International Conference on Computer Aided Design 1987, pp. 42-45, 1987.

[10] K.S. Trivedi, *"Probabiliy and Satistics with Reliability, Queuing, and Computer Science Applications"*, Prentice Hall, Englewood Cliffs, N.J., 1982.

[11] Howard Trickey, *"Compiling Pascal Programs into Silicon"*, Ph D Thesis, Department of Computer Science, Stanford University, July 1985.