A Low Power Scheduler Using Game Theory

N. Ranganathan and Ashok K. Murugavel
Dept. of Computer Science and Engineering
Nanomaterials and Nanomanufacturing Research Center
University of South Florida
Tampa, Florida 33620

ABSTRACT

In this paper, we describe a new methodology based on game theory for minimizing the average power of a circuit during scheduling in behavioral synthesis. The problem of scheduling in data-path synthesis is formulated as an auction based non-cooperative finite game, for which solutions are developed based on the Nash equilibrium function. Each operation in the data-path is modeled as a player bidding for executing an operation in the given control cycle, with the estimated power consumption as the bid. Also, a combined scheduling and binding algorithm is developed using a similar approach in which the two tasks are modeled together such that the Nash equilibrium function needs to be applied only once to accomplish both the scheduling and binding tasks together. The combined algorithm yields further power reduction due to additional savings during binding. The proposed algorithms yield better power reduction than ILP-based methods with comparable run times and no increase in area overhead.

Categories and Subject Descriptors

B.5.1 [**Design**]: Data-path design

General Terms

Algorithms, Low Power Design, High-level Synthesis, Game Theory, Auction Theory

1. INTRODUCTION

The advent of portable and wireless computing has increased the need for low power high performance compute intensive VLSI circuits. Increased power consumption, reduces the battery life in portable systems as well as the reliability and increases the packaging costs [12]. The various steps of behavioral synthesis are: scheduling, allocation and binding [12]. Scheduling is the process of determining when a resource is to be executed, allocation is the process of determining the number of instances of a required resource, and binding is the process of attaching a resource to an operation that needs to be performed [13]. A detailed treatment on low power behavioral synthesis can be found in [2]. Numerous algorithms have been proposed in the literature for low power scheduling. In [15], a force directed low power scheduling algorithm based on heuristics is described. Minimizing the number of times the input operands to a functional unit change reduces the power consumed by the functional units, which is explored in [7]. In [9], a module assignment algorithm for pipelined datapath's is formulated as a max-cost multi-commodity flow problem. A combined register and module assignment algorithm for low power data-path synthesis is presented in [18].

Power reduction through combined scheduling and binding has been investigated in the following works. A scheduling and binding algorithm that minimizes switching activity with simulated annealing is investigated in [1]. In [2], an iterative mechanism based on an initial solution with efficient pruning techniques is used for low power synthesis. A novel approach involving the use of constrained logic programming to minimize the switched capacitance, during scheduling and binding is described in [8]. A heuristics based algorithm under a resource constraint can be found in [13]. In [5], a stepwise approximation algorithm for combined scheduling and binding is described. In [3], a scheduling and binding algorithm, that reduces glitches through clock gating is investigated. Most recently, an integer linear programming (ILP) method for scheduling and a linear programming (LP) method for binding are given in [19].

In this work, we model the problems of low power scheduling and binding as auction based game theoretic problems and solve them using the Nash equilibrium function. The scheduling and binding algorithms achieve power reduction through the use of neighborhood operations, path balancing, functional unit sharing to reduce switched capacitance and glitch reduction. The proposed algorithms do not increase the area overhead, since there is no addition of modules or multiplexors. The application of economic models and game theory as well as the Nash equilibrium for power optimization for the scheduling task in high-level synthesis is being attempted for the first time in this work.

2. GAME THEORY

Game theory is a tool for analyzing the interaction of decision makers with conflicting objectives. Economists have used game theory as a tool to understand the action of economic agents. The basic building blocks of game theory are based on theories proposed by von Neumann in 1928 [10] and Nash in 1950 [11]. A game, in which the rules can be previously stated or agreed upon by the players for use in deducing common strategies, is called a cooperative game. A game in which such agreements cannot be made is called a non-cooperative game [11]. Non-cooperative games are played with fully rational players who know the complete details

CODES+ISSS'03, October 1–3, 2003, Newport Beach, California, USA.

of the game, including each others preferences and outcomes. One of the first works in applying game theory to VLSI CAD problems was attempted in [17], a 2-person zero sum game theoretic formulation has been used to solve the system-level power management policy problem. A game theoretic solution to the problem of binding is described in [4].

The salient features of game theory that aid in the formulation of the behavioral synthesis problem are: *(i) Rationality:*Each player is always selfishly trying to optimize its gain. *(ii) Coalition:* Coalition formation is significant when a subset of players have the same agenda in terms of strategies for optimization can pool in their resources during resource allocation. *(iii) Competition:* In game theory, multiple decision makers who control a specified set of system variables and seek to optimize their conflicting objectives. *(iv) Equilibrium:* A solution is in an equilibrium state when all the players' objectives have been optimized with respect to one another.

The first step is to transform the behavioral synthesis problem into a performance model considering power, delay and area as resources. The second step is to transform the performance model into an economic optimization model based on auction theory, with a set of buyers and a seller, all selfishly trying to optimize their individual resources. The goal is to achieve a stable solution which is good for everyone including the seller and the buyers. The Nash equilibrium proposed in [11], is an elegant solution to achieve the stable point. Auction theory is the science of studying the selling of objects among different buyers, such that the seller and the buyers optimize their gain [6]. The cost that each buyer is willing to pay is called a bid. Among the different types of models available in auction theory, the first-bid sealed auction model is most suitable for modeling power optimization in behavioral synthesis. In the first bid sealed auction, the buyer places sealed bids to the seller that cannot be changed and the seller decides, to which buyer the seller is willing to sell his item based on the bids from the potential buyers.

A finite game can be generalized to consist of n players who can choose from a set of strategies S_i where, $i = 1, ..., n$, and a set of payoff functions p_i where, $i = 1, ..., n : S_1 \times ... \times S_n \rightarrow \Re$, where, $\Re$ is the set of all real numbers and a payoff value is assigned to each pair of strategies chosen by the players. The rationality or the equilibrium point is a set of strategies that maximizes the payoff of the player assuming that all other players strategies are held fixed. The game is played until each players strategy is optimal with respect to the strategies of others. The following example is useful in understanding the concepts.

$$\begin{array}{cc} & P2 \\ P1 & \begin{array}{|c|c|c|c|} \hline 3 & -2 & 1 & 3 \\ \hline 2 & 1 & 0 & -4 \\ \hline -1 & 1 & -2 & 2 \\ \hline \end{array} \end{array}$$

The game consists of two players, **P1** and **P2**, and the matrix given above. The entries in the matrix are the possible outcomes of the game, corresponding to the selections made by the two players. The various possible alternatives for player **P1** are the rows of the matrix and for player **P2** are the columns of the matrix. The alternatives are the strategies that each of the players can choose from. In this game, for **P2**, column 1 is the best since at most he loses 1 and for **P1**, rows 2 and 3 are the best since at most he loses 2. These are the secure strategies of the two players. If **P2** plays the game first then his strategy will be to choose column 1, and the unique strategy of **P1** is row 3, with an outcome gain of 1 by **P1**. But, if **P1** plays first, he can choose either row 2 or row 3. If **P1** chooses row 2, then the best strategy for **P2** is column 1, but, if **P1** chooses row 3, then the best strategy for **P2** will be column 4. In this game, we could assume that the players do not make their decisions independently, and there is a predetermined ordering for the players. If **P1** decides first and passes the choice of his row to **P2** then **P2** has an advantage over **P1**. If the best choice for **P1** (i^*), and the optimal response of **P2** (j^*), is given as,

$$a_{i^*j^*} = \max_j a_{i^*j} = \bar{V} = \min_i \{\max_j a_{ij}\} \tag{1}$$

where, the $\min$ and $\max$ operations are specified in the order in which they are to be performed. The outcome of the game when **P1** follows **P2** is a gain of 1 by player **P1**. When **P2** follows **P1**, the outcome of the game is a gain of 2 by **P2**. If the two players choose independent of each other which is normally the case, player **P2** gains 2, which is the equilibrium solution for this game. Determining this equilibrium solution is complex for games with multiple players. Hence, we apply the rational Nash equilibrium to find such a solution.

Let S_i be the set of various possible strategies of player i, where, $i = 1, ..., N$, and p^i be the payoff for player i. Nash equilibrium can be stated as: a set of strategies $(n_1^* \in S_1), ..., (n_N^* \in S_N)$ such that $p^i_{(n_1^*,...,n_i^*,...,n_N^*)} \geq p^i_{(n_1^*,...,n_i,...,n_N^*)}$ for all i, $n_i \in S_i$. In words, this translates to, a player's payoff does not increase if any of the players unilaterally deviate from the Nash equilibrium strategy. The Nash equilibrium NE, defines the payoff function for all the players in the game. The result that we are interested in, is the global payoff $\bar{p}$ of all the players in the game, or the average payoff of each player due to the combined strategies, $NE = \{n_1^*, ..., n_N^*\}$. Mathematically, this can be given as,

$$\bar{p} = \sum_{i=1}^{N} p^i_{(n_1^*,...,n_N^*)} \tag{2}$$

The Nash equilibrium for an N-player finite game is a N-tuple set of strategies $\{n_1^*, n_2^*, ..., n_N^*\}$, given by N inequalities such that, no single player can gain by individually changing his strategy. The inputs needed to calculate the Nash equilibrium are: (i) the strategies available for each player, (ii) the number of players in the game, and (iii) the number of alternatives a_i that are available for each player i. The steps for calculating the Nash equilibrium are: (i) determine all possible outcomes for the given game were the number of possible outcomes for the given game is $(\sum_{i=1}^{N} a_i^N)$, where a_i denotes the number of alternatives for player i, (ii) determine the inequalities for each player i such that his departure from the Nash equilibrium will lead to no increase in his gain which is stated formally as $(p^{i^*} \triangleq p^i_{(n_1^*,...,n_i^*,...,n_N^*)} \geq p^i_{(n_1^*,...,n_i,...,n_N^*)})$, and (iii) the outcome that satisfies all the inequalities defined in step (ii) forms the Nash equilibrium solution of the game.

An elegant proof for the existence of equilibrium points for a finite game is given in [11]. An interesting fact about the Nash equilibrium is that, *there is a guaranteed existence of a solution* if we allow n_i to be of mixed strategies [11].

3. SCHEDULING

We model the scheduling problem as an auction based first-bid sealed auction and then describe a game theoretic solution.The auctioning of items by a seller through bidding can be extended to the auctioning of operations in the DFG, G_f, among the modules of the architecture, G_a as, with an operation being bought by only one module and the sale entity being power consumption. In the presence of multiple operations and modules, an equilibrium point needs to be achieved. An auction consists of a set of available resources M, and a set of interested buyers O, where M is not equal to O and a buyer can only buy a single item at a time. The buyer $(b \in O)$ may also have some preference for some resources over the others, and this can be specified in terms of a cost function

f_{sch}, These functions are mappings from the buyers domain to the resource domain. Let f_{sch}, be the cost function that maps buyer b in the buyers domain to the resources x, y in the resource domain. If the buyer prefers $x \succ y$, then, $f_{sch}(x, b) < f_{sch}(y, b)$.

input : Data flow graph (DFG) G_f, Architecture G_a
output : Scheduled data-flow graph (S-DFG) G_{sf}
$i \leftarrow 0$;
% initially level i is set to top level;
$CG \leftarrow DFG$;
**: Perform breadth first search of current data-flow graph CG;
$O_{cg} \leftarrow$ Operations in the top level of the CG;
for *each set of operations* $O_{ij} \in O_{cg}$ **do**
% O_{ij} is the set of operations of same type j at level i that can be assigned to the same module;
$CMP_{sch}, CMD_{sch} \leftarrow$ Payoff matrix (O_{ij}, M_{ij}, i);
$NE_{sch}(i) \leftarrow$Nash sol. ($CMP_{sch}, CMD_{sch}, M_{ij}, S$);
% Nash equilibrium solution at control step i - see Alg 2;
$O_{sf} \leftarrow$ Set of operations that form the Nash equi.;
$CG \leftarrow \{CG - O_{sf}(i)\}$;
% Remove operations that are scheduled and their edges;
$O_{sf} \leftarrow O_{sf} \cup O_{sf}(i)$;
% Assign operations that form the Nash equilibrium to step i;
if CG != $EMPTY$ **then**
$i \leftarrow i + 1$;
goto **;
end
else
return;
end
end

Algorithm 1: Game theoretic scheduling algorithm

Formally, we can look at the preference mapping as: A buyer has a preference to an allocation, such that $x \succeq y$, i.e., resource x is preferred over resource y, where $x, y \in M$, where M is the set of all available resources at the time instant and $M \in \Re^b$. If a buyer has no preference between resources x and y, then it is denoted as $x \sim y$. We assume that the preferences are transitive, i.e., if $x \succeq y$ and $y \succeq z$, then it implies $x \succeq z$, where $x, y, z \in M$. The scheduling of the feasible resources under a given resource constraint for power optimization can be given as,

$$FS(\vec{m}) = \{x : \vec{m} \cdot x \leq w\} \tag{3}$$

where, $FS(\vec{m})$ is the *feasibility set* for all the operations and their resources and $\vec{m}$ is the cost vector corresponding to the resources. If there exists a preference for an operation towards a particular resource, the *preference set* for the operation is a set of allocations with that preferred module, operation pair. The preference set is a subset of the *feasibility set*. The preference set $PS(\vec{m})$, is defined mathematically as,

$$PS(\vec{m}) = \{x : x \in FS(\vec{m}), f_{sch}(x) \geq f_{sch}(x'), \forall x' \in FS(\vec{m})\} \tag{4}$$

The power cost function of a module m for operation o at control step i is mathematically given as,

$$CMP_{sch}[m, o] = P_{pow_i} + P_{com_i} + \sum_{k=0}^{i-1}(P_{pow_k} + P_{com_k}) \tag{5}$$

where, P_{pow_i} is the power consumed by module m for operation o at level i, P_{com_i} is power consumed by communication between the modules with m being the destination at level i and the last term

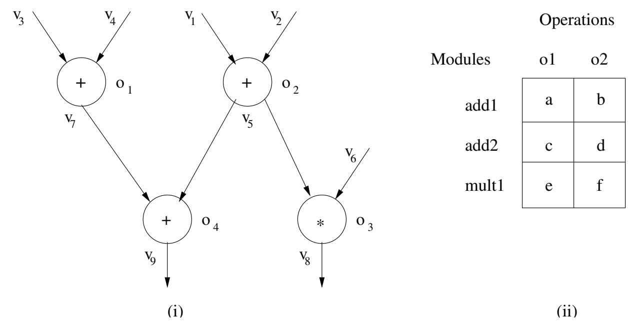

Figure 1: (i) Data-flow graph (ii) Cost matrix

is the average power consumed by the given module from level 0. The delay cost function of a module m for operation o at control step i is mathematically given as,

$$CMD_{sch}[m, o] = D_{pow_i} + D_{com_i} + \sum_{k=0}^{i-1}(D_{pow_k} + D_{com_k}) \tag{6}$$

input : Cost Matrix CM, Number of player N, Set of various possible strategies S
output : Nash equilibrium solution NE_{sch}
for *each player i, where i* $\leftarrow 1$ ***to*** N **do**
for *each set of strategies* ($n_1 \in S_1$),...,($n_N \in S_N$) **do**
Calculate $p^i_{(n_1,...,n_N)} = CM[i, n_i]$;
end
% The payoff for strategy set $(n_1, ..., n_N)$ is calculated;
for *each strategy* ($n_i \in S_i$) **do**
$n_i^* \leftarrow$ Nash equi. strategy for player i given by the inequality $p^i_{(n_1^*,...,n_i^*,...,n_N)} \geq p^i_{(n_1^*,...,n_i,...,n_N)}$;
end
$NE \leftarrow NE \cup n_i^*$;
end
% $NE = \{n_1^*, ..., n_i^*, ..., n_N^*\}$, set of optimized strategies for all N players;

Algorithm 2: Algorithm to calculate the Nash equilibrium

where, D_{pow_i} is the delay of module m for executing operation o at level i, D_{com_i} is communication delay between the modules m at level i and the modules to which it was connected in the previous cycles and the last term is the average delay for the given module from level 0. In our formulation, the two optimizing parameters, namely delay and power are considered as a single optimizing parameter. This helps to simplify the formulation and solve problem as a simple resource allocation game.

Consider an architecture with two adders and a multiplier for the data-flow graph given in Figure 1. From the example given in Figure 1-(1), operations *o1, o2* can be scheduled in control cycle 1. We will discuss the game for control cycle 1 to help in understanding the problem, Figure 1-(ii) gives the power cost matrix for cycle 1. Let, a, c, e be the cost associated with scheduling operation *o1* to the modules *add1, add2, mult* respectively and b, d, f the cost associated with scheduling operation *o2* to the modules *add1, add2, mult* respectively. Simply, $co(m, o)$ is the cost of scheduling operation o to module m. The bidding strategy, is the bids by the operation for the modules *add1, add2, mult*. The Nash Equilibrium is the set of bids for the modules, such that no operation can gain by deviating unilaterally (i.e. operations changing their bids).

The complete set of module and operation pairs that are possible for a given cycle is referred to as MO. In the examples, if adder 1 is combined with operation 1 and adder 2 with operation 2, the MO will be {(1,1),(2,2)}. For the set MO, the total cost $CO = co(1,1) + co(2,2)$. Another such combination is adder 1 with operation 2 and adder 2 with operation 1, the corresponding MO is {(1,2),(2,1)}. Now, for each control step, the problem is to find the minimum of all CO's (CO_{min}),

$$CO_{min} = min_{l,k=1,2,3 l \neq k}[co(m_l, 1) + co(m_k, 2)] \quad (7)$$

This work is based on the assumption that Nash equilibrium strategies exist for the scheduling of operations such that the total power for the set MO is minimized. In other words the competition drives the operations to choose the modules such that power minimization is achieved. Normally, in a bid, the bidder has a profit but in the case of the modules, they don't have a profit of their own and so the bid is a value from the cost matrix. The bid by module 1 for operation 1 will be given as, $b(1,1) = co(1,1)$. The generalized cost matrix CM of size x_j x y_j is given as,

input : Nodes O_{ij}, modules M_{ij}, control step i
output : Power pay off matrix CMP_{sch}, delay pay off matrix CMD_{sch}

for *each module* $m \in M_{ij}$ **do**
  for *each operation* $o \in O_{ij}$ **do**
    P_{pow_i} ←power to execute oper. o on module m in i;
    D_{pow_i} ←delay for oper. o on module m at step i;
    P_{com_i} ←power to communicate data to module m at step i from a previous step;
    D_{com_i} ←communication delay for module m at step i from a previous step;
    $P_{sum} \leftarrow \sum_{k=0}^{i-1}(P_{pow_k} + P_{com_k})$;
    $D_{sum} \leftarrow \sum_{k=0}^{i-1}(D_{pow_k} + D_{com_k})$;
    $CMP_{sch}[m,o] \leftarrow P_{pow_i} + P_{com_i} + P_{sum}$;
    $CMD_{sch}[m,o] \leftarrow D_{pow_i} + D_{com_i} + D_{sum}$;
  end
end

Algorithm 3: Payoff matrix calculator for scheduling

$$CM = \begin{bmatrix} co(1,1) & ... & co(1,y_j) \\ ... & ... & ... \\ co(x_j,1) & ... & co(x_j,y_j) \end{bmatrix}$$

The set of module-operation pairs $A \in S$ is given as $A = (m_1, o_1), ..., (m_{y_j}, o_{y_j})$, where $m_1, ..., m_{y_j} \in M_{ij}$ are the modules, $o_1, ..., o_{y_j} \in O_{ij}$ are the operations, and S is the set of all possible module-operation pairs. A is a feasible set of module-operation pairs, there could be several such possible "A" sets within S. The module set corresponding to A is, $MS_A = \{(m_A(1), ..., m_A(y_j)\}$. The Nash equilibrium solution A^*, represents the set of module-operation pairs satisfying the Nash equilibrium inequality. The total power (CO) due to all the modules in a given set A is,

$$CO(A) = \sum_{o=1,\dots,y_j} co(m_A(o), o) \quad (8)$$

The proposed algorithm is aimed at minimizing the above equation. Thus, the total power $CO(A^*)$ corresponding to the power optimal set A^* is,

$$CO(A^*) = CO_{min} = \min_{A \in S} CO(A) \quad (9)$$

where, $A^* = \{(m_{A^*}(o), o), o = 1, ..., y_j)\}$ is the Nash equilibrium set of module-operation pairs and MS_{A^*} is the set of modules for A^*. The pseudo-code for scheduling is given in Algorithm 1.

Notations and Definitions

G_f	directed data-flow graph
G_{sf}	scheduled data-flow graph
$O_{sf}(i)$	operations scheduled during control step i
G_a	architecture
x_j	number of modules of type j in the architecture
y_j	number of operations of type j in the architecture
M_{ij}	set of all modules of type j in control step i
O_{ij}	set of all operations of type j in control step i
$NE_{sch}(i)$	Nash equilibrium solution at control step i
CG	current data-flow graph
O_{cg}	operations in the top level of the levelized DFG
$co(m,o)$	power for executing operation o on module m
CMP_{sch}	power cost matrix, $CMP_{sch} = [co(m,o)]$
CMD_{sch}	delay cost matrix, $CMD_{sch} = [co(m,o)]$
S	set of all module-operation pairs available
A	$A \subset S$ is any feasible set of module-operation pairs
MS_A	module set of A, MS_A = {m}, where $m = 1, ..., y_j$
CO	total cost for a specific $A \subset S$
CO_{min}	the minimum total cost of any $A \subset S$

The inputs to the scheduling algorithm are the data-flow graph G_f and the architecture G_a, and the output from the algorithm is the scheduled data-flow graph G_{sf}. A breadth first search of the data-flow graph is performed and the input nodes to the DFG form the nodes in the top level of the DFG. For each set of compatible operations in the top level nodes of the DFG, the payoff matrixes CMP_{sch}, CMD_{sch} are compiled Nash equilibrium solution NE for the game is determined. The Nash equilibrium solution gives, the power optimal schedule for the given DFG at each control step. The payoff matrix equations 5 and 6 are used in Algorithm 3. The nodes that have been scheduled are removed from the data-flow graph. For the current data-flow graph the process is repeated. If all the nodes in the DFG are scheduled, the S-DFG is given as the union of all the Nash equilibrium solutions.

4. SCHEDULING AND BINDING

The algorithm for scheduling and binding can be applied in sequence to obtain a power optimal behavioral synthesis solution. However, this involves applying the Nash equilibrium twice. In this section, we propose a combined scheduling and binding algorithm that applies the Nash equilibrium only once and results in better power reduction than applying the separate algorithms in sequence. The most important step in the combined algorithm is the determination of the payoff matrix, such that the power optimization is maximized. The first-bid sealed auction discussed in the previous sections is used here and hence is not explained again. The algorithm to calculate the payoff matrix used in Algorithm 4 is given in Algorithm 5.

The cost function aims at reducing the delay and the switching activity of the circuit. The power consumption for a module operation pair is,

$$P_{sw} = P_{pow_i} + P_{com_i} + \sum_{k=0}^{i-1}(P_{pow_k} + P_{com_k}) \quad (10)$$

where, P_{pow_i}, is the power consumed by the module m for the operation o in cycle i, similarly P_{com_i} is the power consumed due to the communication between the modules with m as the destination at cycle i, the final term is the total average power of the module from level 0 to level $i-1$. For binding we identify neighborhood operations that can be grouped for the same module to reduce switching activity. If the power due to the neighborhood operations is P_{inp}, then the power cost equation is given as,

$$CMP[m,o] = P_{sw} - P_{inp} \quad (11)$$

The combined scheduling and binding problem is formulated as a resource allocation game with two inter-related parameters to be optimized among the N-players. The notations in Sections 3 are also applicable here.

The inputs to the algorithm are the data-flow graph G_f and the architecture G_a, and the output of the algorithm is the binding matrix B. A breadth first search of the data-flow graph is performed and the input nodes to the DFG form the nodes in the top level of the DFG. For each set of compatible operations in the top level nodes of the DFG, the payoff matrix CM is calculated. The payoff matrix is given in Algorithm 5. The Nash equilibrium for the game is calculated using Algorithm 2 to determine the optimal binding of the operations to the modules and also obtain the power optimal schedule. The register binding task determines the optimal binding for the variables to the registers in each control cycle. Each functional unit that was active in the current control cycle is determined if it was active in the previous control cycle. Then, the input variables to the functional unit in the current cycle are bound to the same registers as in the previous cycle. If the functional unit was inactive in the previous control cycle, then variables are assigned to registers that are equidistant from the functional unit in-terms of the interconnect length.

```
input   : Data flow graph (DFG) G_f, Architecture G_a
output  : Binding matrix B
  i ← 0;
% initially level i is set to top level;
  CG ← DFG;
**: Perform breadth first search of current data-flow graph
  CG;
  O_cg ← Operations in the top level of the CG;
  for each set of operations O_ij ∈ O_cg do
      CMP, CMD ← Payoff matrix (O_ij, M_ij, i);
      NE ← Nash equi. solution (CMP, CMD, M_ij, S);
% Nash equilibrium solution at control step i - see Alg 2;
      Represent A* as a binary binding matrix B;
      CG ← {CG − O_sf};
% Remove edges and operations that have been scheduled;
      if CG != EMPTY then
          for each active functional unit m ∈ MS_A* in control step i do
              for each control step k, where k ← 1 to i − 1
              do
                  if Functional unit m was active in step k
                  then
                      Assign the input variable to the same
                      registers as in step k;
                  end
                  else
                      Assign the variables to new registers
                      with balanced paths;
                  end
              end
          end
          i ← i + 1;
          goto **;
      end
      else
          return;
      end
  end
```

Algorithm 4: Combined scheduling and binding algorithm

```
input   : Nodes O, modules M, level i
output  : Power pay off matrix CMP, delay payoff matrix
          CMD
for each module m ∈ M do
    for each operation o ∈ O do
        P_pow_i ← power to execute oper. o on module m in i;
        D_pow_i ← delay for operation o in module m at step i;
        P_com_i ← power to communicate data to module m at
        step i from a previous step;
        D_com_i ← communication delay for module m at step
        i from a previous step;
        P_sum ← Σ_{k=0}^{i-1} (P_pow_k + P_com_k);
        D_sum ← Σ_{k=0}^{i-1} (D_pow_k + D_com_k);
        P_sw ← P_pow_i + P_com_i + P_sum for scheduling to
        module m at step i;
        Identify power reducing neighborhood operations;
        Determine the # of inputs to the oper. that change;
        P_inp ← power reduction due to neighborhood oper.;
        CMP[m, o] ← P_sw − P_inp;
        CMD[m, o] ← D_pow_i + D_com_i + D_sum;
    end
end
```

Algorithm 5: Payoff matrix calculator for the combined algorithm

5. RESULTS AND CONCLUSIONS

The experimental results on benchmark circuits for scheduling and their resource constraints are given in Table 1. The power values for the circuits and the individual modules were obtained by simulation with 100,000 input vectors using the Synopsys RTL power estimation tool "Power Arc". The input vectors are random, 16-bit, two's complement integers. A library for all the functional units and the registers has been developed based on the TSMC $0.25\mu m$ technology. This library is used by the algorithms to provide a realistic comparison of the various algorithms. The proposed scheduling algorithm (S_{GT}) is compared with an integer linear programming (ILP) based methodology (S_{ILP}) proposed in [19] and a latency based scheduler (S_L) [14]. It should be noted that the latency based scheduler is not optimized for power. Column 2 in Table 1 specifies the resource constraint for the algorithms as the # of adders (A) and the # of multipliers (M). The results for the combined scheduling and binding algorithm are tabulated in Table 2. The ILP-based methodology was obtained by applying in sequence the ILP-based scheduling algorithm and the LP-based binding algorithm proposed in [19]. The Nash equilibrium is computed using *Gambit: Software tools for game theory* [16]. The algorithm has a % power savings of about 11.8 % and 41.3 % on an average over the ILP-based methodology and the algorithm with a random binding and latency based schedule respectively.

Since, the Nash equilibrium algorithm is NP-complete, the proposed game theoretic algorithms are of exponential time complexity. However, it should be noted from the Tables 1 and 2, that the run-times are small, which is due to the fact that the time complexity of the algorithm is a function of the number of players in the game and the set of strategies for each player of the game. In our algorithm, we have restricted the number of players in each game to be less than 4, and there are multiple games for each control step in the behavioral synthesis process. Hence, the run-times are much smaller than one would expect if all the games in a single level are combined into a single game. The time complexity of the Nash equilibrium for an N player game with S strategies for each player is given as $O(N^S NS)$ [16]. Assuming this to be the

Table 1: Experimental results for scheduling

Benchmark Circuits	Resource constraint A, M	Sehwa scheduler [14] (S_L)		ILP approach [19] (S_{ILP})				Game-theoretic approach (S_{GT})			
		Latency cycles	Power mW	Latency cycles	Power mW	**% savings w.r.t. S_L**	Runtime s	Latency cycles	Power mW	**% savings w.r.t. S_L**	Runtime s
Diff. eqn	2, 2	4	26.5	4	24.8	**6.4**	18	5	23.9	**9.8**	22
FIR	3, 3	9	99.7	9	82.8	**16.9**	113	10	78.7	**21.1**	107
IIR	2, 2	8	18.9	8	15.6	**17.4**	48	9	14.8	**21.7**	52
Lattice	3, 3	8	121.0	9	91.3	**24.5**	105	9	82.9	**31.5**	90
EWF	2, 2	16	386.6	17	262.8	**32.0**	182	19	251.0	**35.1**	153
WAVE	2, 2	25	352.1	26	258.2	**26.6**	254	27	232.8	**33.9**	215
NC filter	2, 2	27	507.2	27	389.7	**23.1**	295	27	363.4	**28.4**	256
Average						**20.9**				**25.9**	

Table 2: Experimental results for combined scheduling and binding

Bench-mark circuits	Latency scheduler & random binding ($S_L + B_R$)		ILP-based scheduling & LP-based binding [19] ($S_{ILP} + B_{LP}$)				Latency scheduler & Game-theoretic binding ($S_L + B_{GT}$)				Game-theoretic scheduling & binding ($S_{GT} + B_{GT}$)			
	Latency cycles	Power mW (1)	Latency cycles	Power mW	**% sav w.r.t. (1)**	Run-time (s)	Latency cycles	Power mW	**% sav w.r.t. (1)**	Run-time (s)	Latency cycles	Power mW	**% sav w.r.t. (1)**	Run-time (s)
Diff. eqn	4	26.6	4	22.9	**13.9**	35	4	20.1	**24.4**	59	5	16.0	**40.0**	37
FIR	9	95.7	10	61.7	**35.5**	221	9	57.7	**39.7**	216	10	50.4	**47.3**	165
IIR	8	18.3	8	12.8	**30.0**	100	8	12.3	**32.8**	110	9	11.2	**39.0**	96
Lattice	8	113.0	9	66.3	**41.3**	207	8	64.1	**43.2**	215	10	55.9	**50.5**	161
EWF	16	317.2	17	204.0	**35.6**	259	16	210.0	**33.8**	285	19	191.3	**39.6**	204
WAVE	25	291.9	26	201.2	**31.0**	427	25	196.3	**32.7**	427	27	179.2	**38.6**	332
NC filter	27	438.1	27	324.3	**25.9**	501	27	312.8	**28.6**	528	27	286.7	**34.6**	377
Average					**30.5**				**33.6**				**41.4**	

complexity of the Nash equilibrium, the complexity of the scheduling algorithm is given as, $O(lcN^S NS)$, where l is the number of levels for the given DFG and c is the number of sets of compatible operations at each level of the DFG. The space complexity of the algorithms is given as, $O(MO)$, where M is the number of modules and O the number of operations in the game. The algorithm can be further improved by incorporating voltage scaling and dynamic frequency clocking. We plan to incorporate interconnect binding in our game theoretic formulation since, interconnects will play a significant role in synthesis of data-path circuits as we move into deep sub-micron and nano regimes.

6. REFERENCES

[1] A. Dasgupta and R. Karri. Simultaneous scheduling and binding for power minimization during micro-architecture synthesis. In *Proc. Intl. Symp. on Low Power Electronics and Design*, pp 69-74, 1995.

[2] A. Raghunathan and N.K. Jha. SCALP: An iterative improvement based low-power data path synthesis system. *IEEE Trans. on CAD*, 16(11):1260-1277, Nov. 1997.

[3] A. Raghunathan, S. Dey and N.K. Jha. Register transfer level power optimization with emphasis on glitch analysis and reduction. *IEEE Trans. on CAD*, 18(8):1114-1131,Aug.1999.

[4] A.K. Murugavel and N. Ranganathan. A game-theoretic approach for binding in behavioral synthesis. In *Proc. Intl. Conf. on VLSI Design*, pp 452-458, 2003.

[5] C. Park, T. Kim and C.L. Liu. An efficient data path synthesis algorithm for behavioral-level power optimization. In *Proc. Intl. Symp. on Circuits and Systems*, v 1, pp 294-297, 1999.

[6] D.F. Ferguson, C. Nikolaou, J. Sairamesh and Y. Yemini. Economic models for allocating resources in computer systems. In Clearwater S., editor, *A Paradigm for Distributed Resource Allocation*. World Scientific, 1996.

[7] E. Musoll and J. Cortadella. Scheduling and resource binding for low power. In *Proc. Intl. Symp. on System Synthesis*, pp 104-109, 1995.

[8] F. Gruian and K. Kuchcinski. Operation binding and scheduling for low power using constraint logic programming. In *Proc. Euromicro Conf.*, v 1, pp 83-90, 1998.

[9] J.-M. Chang and M. Pedram. Module assignment for low power. In *Proc. European Design Automation Conf.*, pp 376-381, 1996.

[10] J. von Neumann. Zur Theorie der Gesellschaftsspiele. *Zur Theorie der Gesellschaftsspiele*, pp 295-320, 1928.

[11] J.F. Nash. Non-cooperative games. *Annals of Mathematics*, 54(2):286-295, Sep. 1951.

[12] L. Benini and G. Micheli. System-level power optimization: Techniques and tools. *ACM Trans. on Design Automation of Electronic Systems*, 5(2):115-192, Apr. 2000.

[13] L. Kruse, E. Schmidt, G. Jochens, A. Stammermann, A. Schulz, E. Macii and W. Nebel. Estimation of lower and upper bounds on the power consumption from scheduled data flow graphs. *IEEE Trans. on VLSI*, 9(1):3-14, Feb. 2001.

[14] N. Park and A.C. Parker. Sehwa: A software package for synthesis of pipelines from behavioral specifications. *IEEE Trans. on CAD*, 7(3):356-370, Mar. 1988.

[15] P.G. Paulin and J.P. Knight. Scheduling and binding algorithms for high-level synthesis. In *Proc. Design Automation Conf.*, 1989.

[16] R.D. McKelvey, A. McLennan and T. Turocy. *Gambit: Software tools for game theory*. California Inst. of Tech., Uni. of Minnesota and Texas A & M Uni., v 0.97, Sep. 2002.

[17] S.K. Shukla and R.K. Gupta. A model checking approach to evaluating system level dynamic power management policies for embedded systems. In *Proc. Intl. High-level validation and Test Workshop*, pp 53-57, 2001.

[18] V.K. Srikantam, N. Ranganathan and S. Srinivasan. CREAM: combined register and module assignment with floor-planning for low power data-path synthesis. In *Proc. Intl. Conf. on VLSI Design*, pp 228-223, 2000.

[19] W.T. Shiue and C. Chakrabarti. ILP-based scheme for low power scheduling and resource binding. In *Proc. Intl. Symp. on Circuits and Systems*, v 3, pp 279-282, 2000.