A Design Space Exploration Scheme for Data-Path Synthesis

Chittaranjan A. Mandal, Partha Pratim Chakrabarti, *Member, IEEE,* and Sujoy Ghose, *Member, IEEE*

***Abstract*— In this paper, we examine the multicriteria optimization involved in scheduling for data-path synthesis (DPS). The criteria we examine are the area cost of the components and schedule time. Scheduling for DPS is a well-known NP-complete problem. We present a method to find nondominated schedules using a combination of restricted search and heuristic scheduling techniques. Our method supports design with architectural constraints such as the total number of functional units, buses, etc. The schedules produced have been taken to completion using GABIND as written by Mandal *et al.*, and the results are promising.**

***Index Terms*— Data-path synthesis, design space exploration, scheduling, VLSI.**

I. Introduction

DATA-PATH synthesis (DPS) first involves scheduling of operations and then allocation and binding of abstract design entities to their physical counterparts. At the end of DPS, we are required to find one or more "optimized" implementations for a given behavioral specification. In general, the objectives of optimization are multifold, making the DPS a multicriteria optimization problem. In this paper, we consider the scheduling problem along with the criteria of area (estimated through the cost of individual components) and performance of the final design measured as a function of the length of the schedule. These two criteria are noncommensurate and we represent the cost of a design as a tuple of costs of the individual objectives, similar to the approach taken in Stewart *et al.* [2]. One cost tuple is said to be better than another distinct cost tuple if the cost of each criterion of the first tuple is no worse than the corresponding costs of the other tuple. A design whose cost tuple is better than that of another design is said to dominate that design. The global problem of optimization is to find the set of designs that are not dominated by any other design. The set of feasible designs satisfying the design parameters constitute the design space. Each design point in the design space corresponds to an estimate of hardware requirement and performance, computed as a function of the schedule time. Thus, an algorithm for DPS needs to consider techniques not only for scheduling and allocation, but also for a systematic exploration of the design space to locate these nondominated designs. The starting point of design space exploration (DSE) often revolves around the basic scheduling problem. We perform DSE using a combination of controlled search and heuristic scheduling techniques.

Conventional scheduling algorithms require a time constraint or specification of the available hardware operators or functional units (FU's). In a practical DPS situation, neither the appropriate time constraint, nor the appropriate FU requirement will be known in advance. Through DSE, we systematically explore several combinations of time constraints and hardware resource configurations that are feasible. In our scheme, we have a state space generation mechanism coupled with an estimator for obtaining various (hardware cost, performance) estimates. A controlled depth first branch and bound is used to determine the hardware cost estimate and produce a partial schedule for a given time constraint. The search is controlled by a few design parameters.

At the heart of the DSE mechanism is the controlled-search-based resource estimation and partial scheduling (REPS) algorithm. The basic DSE technique makes use of the REPS algorithm to estimate the hardware requirement, as tightly as possible, so that the design parameters are also satisfied. REPS also returns a partial or complete schedule depending on the situation. Scheduling, however, is an NP-hard problem [3] and, for large problem instances, it may be necessary to settle for a restricted search. With restricted search, the design points obtained are approximate [lower bounds (lb's)] and the schedules may be partial in the sense that the degree of freedom of some operation may still be more than one. To meet this situation, a local exploration mechanism has been developed to explore the neighborhood of such a design point to obtain one or more nondominated design points for which feasible schedules will exist. The local exploration mechanism also produces a feasible schedule for each design point that it returns. Such schedules are obtained using existing scheduling techniques, which perform scheduling using the precedence constraints and sometimes the available hardware resources.

In the following, we present details of our solution to the problem of DSE to generate a set of schedules that will represent nondominated designs. The inputs for DSE are explained in Section II. The estimates used by REPS for hardware cost and schedule time are discussed in Section III. REPS itself is presented in Section IV. The overall DSE mechanism (which uses REPS) is then explained in Section V. Finally, the experimental results and conclusions are presented in Sections VI and VII, respectively.

Manuscript received September 12, 1996; revised October 14, 1998.

C. A. Mandal is with the Department of Computer Science and Engineering, Jadavpur University, Calcutta 700032, India (e-mail: crmandal@hotmail.com).

P. P. Chakrabarti and S. Ghose are with the Department of Computer Science and Engineering, Indian Institute of Technology, Kharagpur WB 721302, India (e-mail: ppchak@cse.iitkgp.ernet.in; sujoy@cse.iitkgp.ernet.in).

Publisher Item Identifier S 1063-8210(99)04563-1.

II. Inputs to DSE

Operation precedences are the most important input to the DSE algorithm. For practical design examples there will be a number of basic blocks (bb's) and for operations in each bb there will be precedence constraints in the form of a partial order. Each type of operation is also assigned an execution time, which indicates the number of time steps over which the operation will execute. The execution time of an operation is determined by the speed of the hardware implementation of that type of operation.

In order to explore the designs that are possible for a given behavioral specification in reasonable time and structured manner, it is desirable to guide the design process with some simple user-specified parameters. Our DSE scheme uses the following parameters:

1) NFUS;
2) NBUS;
3) NVREF.

NFUS indicates the number of sites where hardware operators will be clustered. However, FU's need not be formed during scheduling. No two hardware operators at the same site may receive inputs or deliver outputs in the same time step. NFUS controls the maximum operation parallelism in the implementation. NBUS is the maximum number of logically distinct buses in the system and determines the maximum number of concurrent data transfers. NVREF is the maximum number of distinct variable references permitted in any time step. Reading and writing to a variable are considered distinct accesses. This parameter is used to have a check on the storage bandwidth.

Though the above parameters are independent, they are well corelated. It may be expected that $\mathrm{NBUS} \approx 3\mathrm{NFUS}$ and $\mathrm{NVREF} \approx 3\mathrm{NFUS}$.

III. Measures for DSE

In general, we shall have to resort to scheduling to find design points. Since exact scheduling is computationally intensive, we rely on heuristic measures to aid scheduling and compute the tuple costs. The measures we compute are as follows.

A. Estimates of Hardware Requirement

We would, like to estimate the costs of the following:

1) hardware operators;
2) storage elements;
3) buses;
4) switching elements.

At this early stage of design, it is difficult to have reliable estimates of these register transfer level (RTL) components mentioned above. Among these, it is easiest to estimate the requirement of hardware operators. It is also possible to estimate the storage requirement before scheduling has been done [5], but this estimate is relatively less reliable. It is most difficult to estimate the switch requirement before scheduling has been performed, and this cost has been excluded in this paper. All the above estimates are more accurately computed for a scheduled design. For a scheduled straight-line code, the minimum storage cost can be obtained using the *left edge algorithm* [6]. For a bus-based interconnection scheme, a reasonable estimate of switch requirement can be obtained after transfers have been mapped to buses [7]. In Section III-B, we indicate the computation of lb estimates for specific hardware operators, total number of operations in any time step, and the bus requirement.

B. Estimators for DSE

1) Estimation of Resources for Specific Operations: A method similar to [8] and [9] is used to compute a lb estimate of each hardware operator. We first introduce the notion of a *window*, which we shall use to compute the estimates. A contiguous sequences of time steps is referred to as a window. Given a directed acyclic graph (DAG) to be scheduled in n time steps, there can be n windows of size one, $n-2$ windows of size two, etc., and one window of size n. Thus, there can be a total of $n(n+1)/2$ windows with n time steps. For determining the estimate, it is necessary to determine the earliest and latest times at which each operation in the DAG (of a bb) may be scheduled. These are most conveniently determined from the as soon as possible (ASAP) and as long as possible (ALAP) schedules.

The construction of the lb is now explained. First consider any window in the given DAG of j, $j \leq n$, steps and starting at time step i, $1 \leq i \leq n$. Consider any operation o in the DAG, let the earliest time step where it can be scheduled be $t_{a,o}$ and the latest time step where it can be scheduled be $t_{l,o}$. If $t_{a,o} \geq i$ and $t_{l,o} < i+j$ then, in each and every possible schedule of the DAG, o must lie in the aforesaid window. Let the operation o be of type x. Let there be a total of m operations of type x restricted in the same manner to lie in this window, then at least $l^r_{x,i,j} = \lceil m/j \rceil$ hardware operators are required to realize the operations of type x in the DAG. Let $l^r_x = \max_{i,j} l^r_{x,i,j}$, $1 \leq i \leq n$, $1 \leq n-i-j+1 \leq n$. Then, l^r_x is also a lb on the number of hardware operators for operations of type x. Similarly, let L^r_x be the maximum of l^r_x over all the DAG's. This too is a lb. This is the principle that has been used to derive the lb's on the number of operation units of type x for the design. If C_x is the cost per unit for an operator of type x then the estimate of the resource cost is defined as $\sum_x L^r_x C_x$.

2) Estimation on the Total Number of Operations per Time Steps: This metric is required to ensure that the parameter NFUS is not violated. It is computed in a manner similar to the method explained above, for the previous metric. Therefore, this estimate is also obtained as an lb. Only in this case, no distinction is made between the different types of operations, and all the operations occurring in a window are counted.

3) Estimation for Buses: The bus requirement is estimated by examining the transfers that take place in various windows. Each operand of an operation contributes to a transfer. Transfers also arise due to variable assignments. We consider the transfers that will be restricted within the window under consideration and then compute the lb on the number of concurrent transfers. Common variables that form inputs to

operations need to be handled carefully. For the purpose of computing an lb, transfers arising from the same variable to operations that are neither ancestors, nor descendents of one another may be counted only once, otherwise they may be considered distinct.

4) Estimation for Variable Accesses: The number of distinct variable accesses is determined by examining the variable accesses that take place in various windows. Each input and output operand of an operation contributes to a variable access. As usual, we consider the accesses that will be restricted within the window under consideration to compute the lb. Input operands named by the same variable need careful handling. Like the lb determination for buses, variable accesses by operations that are neither ancestors or descendents are counted only once, otherwise they may be considered distinct.

The above estimation methods are applicable to individual DAG's of bb's. For multiple bb's, these estimators need to be applied to each of those bb's. The global lb is obtained by merging the individual lb's.

5) Schedule Time: The schedule time is dependent on the duration of the clock cycle and the number of time steps in the schedule. We will, in general, only be concerned with the number of time steps. However, when the intermediate representation of the BS consists of multiple bb's, the effective schedule time of the design needs to be suitably defined, as a function of the time steps for each bb.

IV. Search Algorithm for REPS

The REPS algorithm uses the estimators described in the previous section to determine the resource cost for a given schedule time. It also returns a complete schedule if the window size W is set to one. If $W > 1$, then it returns a partial schedule in the sense that the degrees of freedoms (DOF) of all operations are suitably reduced. Some operations may still have nonzero DOF, but no more than W. We have experimentally found that these estimators work better for smaller DAG's. Thus, the REPS algorithm partitions the DAG, if necessary, into smaller DAG's, applies the estimator to these partitions, and combines the estimates for the different partitions to arrive at the final estimate. The schedules of partitions are combined to return the partial schedule of the entire DAG. The REPS algorithm does a systematic search of the problem space, using DAG partitioning as the state space decomposition procedure.

A. DAG Partitioning

The partitioning scheme involves splitting the n time steps, in which to schedule (a partition of) the DAG, if $n > W$, into $\lceil n/w \rceil$ bands, each of at most W time steps. Each operation of the DAG is restricted to lie in only one of these bands. For operations whose ASAP and ALAP times t_a and t_o lie within a band, nothing needs to be done. For other operations, it is necessary to take a decision regarding the band where it should be restricted to be scheduled. The process of decomposition continues recursively until the size of none of the partitions of the current DAG are more than W. The resource requirement of a particular type of resource in the design is the maximum requirement of that resource over all the partitions of a particular DAG.

B. The Search Scheme

The memory requirement for storing the partial solutions is high. Thus, we have chosen the depth first branch and bound (DFBB) search, whose memory requirement is minimal. In the search scheme, the partitioned DAG's are treated like separate DAG's. If the number of time steps within which the DAG needs to be scheduled does not exceed W, then no more repartitioning is done and the current estimates are accepted. Otherwise, it is split into two smaller DAG's. The splitting is done near about the middle so that the two subproblems generated are of similar size. If there are one or more operations crossing the boundary, then all the possibilities of distributing these operations need to be tested out. This is where the search comes in. We perform the search by explicit backtracking. In order to keep track of the moves, a stack (stack1) is used. For an operation that crosses the partition boundary, there are three moves to be made, which are as follows.

1) It has to be scheduled in the top half.
2) It has to be scheduled in the lower half.
3) Its original freedom has to be restored.

The first two moves are forward moves, while the third move is there to perform backtracking. The first move is performed right away, while the other two are put into the stack (stack1) in that order. After making a move, the ASAP and ALAP schedules are recomputed. When the move made is of the forward type, the resource estimate is computed. This is an lb estimate and may increase as the depth of the search increases. If this estimate exceeds the estimate of the best design found thus far, the current move is then rejected and backtracking is initiated. Move rejection followed by backtracking also takes place if the resource estimate after the move is found to be infeasible with respect to the design parameters. Initially, there is no solution and, thus, at the beginning, a dummy solution of very high cost is assumed. This solution is replaced by the first (partial) feasible solution that is found.

When partitioning is done, it becomes necessary to handle multiple bb's. A list is used to handle these bb's. To start with, the initial bb is entered in the list. The list is then passed to REPS to estimate the resource requirement for a single bb. The bb at the head of the list is examined. In case the bb is a small one, then it is temporarily removed from the head and placed in another stack (stack2). Otherwise, it is partitioned into two smaller blocks and entered at the end of the list. By placing the partitioned bb's at the end of the list, it is ensured that the sizes of the bb's in the list are near about the same. Therefore, the resource estimate obtained is, in some sense, a useful one.

When the list becomes empty, it is assured that the sizes of all the (partitioned) bb's is less than or equal to W. When this condition is satisfied, no more partitioning needs to be done, and the set of stacked (stack2) bb's constitute the partial schedule. If $W = 1$, then this is also the complete schedule and corresponds to a feasible design point. Otherwise, the design point found is an approximate one. If this point corresponds to

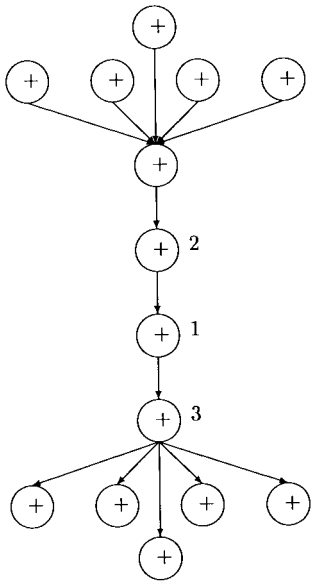

Fig. 1. DAG for example 1.

a design with a better (lower) resource estimate than that of the best stored design, then it replaces that design, otherwise it is rejected and backtracking is initiated. The algorithm terminates with a failure if there exists a partition where the design parameters cannot be possibly satisfied.

The requirement for each resource is generated in the form of the tuple $\langle m, w, j \rangle$, where m is the number of units of that entity occurring in a window of size w in the bb j. Such tuples are generated for the maximum number of operations per time step, bus requirement, storage access point requirement, and requirement for hardware operator for each type of operation. $\lceil m/w \rceil$ is the lb on that resource. Tuples, instead of the resource requirement, are generated because this information is needed by the exploration heuristic used in the DSE tool (Section V-A).

The search mechanism explained above has one anomaly. The problem is that when REPS is being done with a relaxed time constraint, then the search space turns out to be far larger than when REPS is being done with a tighter time constraint. This situation is addressed by running an approximate scheduling algorithm on the current bb before going ahead to partition it into smaller bb's. Basically, a time- and resource-constrained scheduling-type approximate scheduling algorithm is required. If a time-constrained scheduling-type algorithm is used, then the final resource requirement after scheduling should not exceed the lb estimate of the resource requirement while satisfying the design parameters. If a resource-constrained (hardware operators) scheduling-type algorithm is used, then the time steps required should not exceed the available number of time steps. If the approximate scheduling algorithm terminates successfully, then the current bb may be assumed to satisfy the lb on the resource estimate and the remaining bb's may be examined.

Example 1: We consider the DAG, shown in Fig. 1, for scheduling in ten time steps. We first note that DAG's of this type pose a difficulty for the hardware lb estimator described earlier. If we compute the lb for scheduling in nine time steps, then the estimator will report a requirement of two adders, whereas three adders will be actually needed.

For illustrating the working of REPS, we consider a schedule in ten time steps for the aforementioned DAG, with $W = 1$. An inspection of Fig. 1 reveals that two adders will be required. The estimator initially determines the correct lb in the start step (state 0), but the approximate scheduling algorithm fails to schedule using two adders in ten time steps. Hence, partitioning of the DAG is required.

We choose the fourth time step for partitioning. The time steps are assumed to increase in a direction opposite to the direction of the dependencies. Thus, in Fig. 1, operation 1 would be scheduled before operation 2. The time frame of the operation marked "1" in Fig. 1 spans across this time step. We restrict it to be scheduled on or before time-step 4 (state 1). This decision does not complete the partitioning of the DAG. Partitioning is completed after the operation marked "2" in Fig. 1 is scheduled after time-step 4 (state 2). The lb continues to be three and, this time, the approximate scheduling algorithm succeeds in finding a schedule without violating the lb This case is recorded as the current best solution. Backtracking is initiated. Since the lb of the parent state (state 1) matches with the current cost, the other child of state 1 is not generated. Backtracking is continued to state 0.

The other option of scheduling the operation marked "1," above or at time-step 4 is exercised, leading to state 3. The lb continues to be two and the partitioning is completed by restricting the node marked 3 in Fig. 1 on or before time-step 4 (state 4). The approximate scheduling algorithm succeeds and a new and better solution is recorded. Backtracking is initiated and continues to the start state and the algorithm terminates. After partitioning, in states 2 and 4, the two smaller DAG's are entered into the queue. The schedule of the approximate algorithm that obtained the best solution (for this problem) is accepted as the schedule. However, it must be noted that, in some cases, partial schedules may be returned when $W > 1$ and lb's and upper bounds (ub's) from solutions obtained by approximate algorithms do not match. □

C. Multicycling and Pipelining

Special care has to be taken for operations whose execution does not get completed within a single time step, as for multicycle and pipelined operations. We consider only simple arithmetic pipelined implementations because these are the ones that are most commonly used in practice in DPS. When multicycle or pipelined operations are present, they may sometimes cross-partition boundaries. In such situations, the computation of the lb is more involved.

REPS handles multicycle operations in the following manner. Suppose that the time frame of a multicycle operation of k time steps crosses the partition boundary set at time t. Up to k possibilities need to be examined. These are, initiating the operation at times earlier than time step $t - k$, initiating the operation at times $t - k, \cdots, t$, and at times, later than t. Initiation of the operation at specific times is the additional overhead for handling multicycle operations. When more than

a single operation crosses the partition boundary, partitioning is initiated with the operation requiring the least number of cycles for its execution. Handling of pipelined operations is as follows. Consider a p-stage pipelined implementation with a stage delay of d of an operation of type x. The result of such an operation will be obtained $p-1$ time steps after initiation. Therefore, while scheduling, the number of time steps to complete operations of type x should be taken as p. The lb is obtained as for a multicycle operation, the stage delay d being used in place of the number of time steps k of the multicycle operation.

V. Scheme for DSE

We now describe the overall scheme for DSE. At the heart of the DSE technique is the REPS algorithm, which is repeatedly invoked with varying time constraints. The time constraints with which REPS is invoked is determined by the exploration heuristic in Section V-A. With each invocation REPS either indicates that the time constraint is not feasible or it returns a design point and a schedule. The design points thus obtained are used to obtain the design space. When a new design point is obtained, one of the three conditions will be true. *The point is dominated by existing design points.* In this case, this design point has to be discarded. *The point dominates a set of the existing design points.* All the dominated points have to be discarded and the new point has to be incorporated in the design space. *It neither dominates, nor is it dominated by other design points.* The point is simply incorporated in the design space. With $W > 1$, then REPS will generally return a partial schedule and an approximate hardware requirement. In this case, it is desirable to obtain the complete feasible schedules, which will be needed for performing subsequent allocation and binding. These schedules will have to be obtained using approximate scheduling algorithms. The detailed scheme of obtaining complete schedules from approximate design points is explained in Sections V-B and V-C.

A. Exploration Heuristic

The resource cost estimation scheme described above requires the number of time steps for each bb to be specified. To start with, the number of time steps for each DAG is set to its critical length, and then REPS is invoked. The resulting resource requirements are computed from the tuples, as explained above, and examined. It was mentioned in Section IV-B that the requirement for each hardware resource or the requirement of FU's, buses, etc., are generated in the form of the tuple $\langle m, w, j\rangle$, where m is the number of units of that entity occurring in a window of size w in the bb j. In case any of the design parameters is violated, a corrective action is taken as follows. Suppose that a design parameter X having the value v_X is violated, i.e., $\lceil m_X/w_X\rceil > v_X$. Consider the effect of adding i, $i > 0$, time steps to the DAG of the bb j_X. Now, the earliest time of each operation o, $t_{a,o}$ remains unaltered, but $t_{l,o}$ goes up by i. Therefore, each operation previously restricted to lie in a window of size w will now lie in a window of size $w+i$. A minimal number of time steps $t_X > 0$ is added to w_X so that, i.e., $\lceil m_X/w_X + t_X\rceil \leq v_X$. REPS is invoked after making the correction.

The DSE retains the set of mutually nondominating design points that have been found. When a design point, characterized by the time constraints and the resource cost estimate, is found to be feasible, it is compared with the stored design points. If it is dominated by any point, then it is not included in the set. If it dominates any point of the set, then it replaces that point. Exploration continues with a new set of constraints, generated as follows. For each operation O whose requirement exceeds unity, we identify the DAG's where it is required maximally. In each of these DAG's, we determine the time t by which the time constraint of that DAG should be relaxed so that the new requirement of the operator will be one less, i.e., $\lceil m_O/w_O + t\rceil = \lceil m_O/w_O\rceil - 1$. Let t_O be the maximum of all the times computed above. Let D_O be the DAG where this relaxation may be effected. Let O^* be the operation for which t_O has the minimum (nonzero) value of all the t_O's. Let D_{O^*} be the corresponding DAG.

We now relax the time constraint on the bb for D_{O^*} by t_{O^*} so that $\lceil m_{O^*}/(w_{O^*} + t_{O^*})\rceil = \lceil m_{O^*}/w_{O^*}\rceil - 1$. This is the heuristic use to conduct the exploration of the design space. Exploration is terminated when the resource requirements of all the operations becomes unity.

B. Scheduling Schemes for Use with DSE

We have noted that the REPS generates (hardware cost, performance) estimates and corresponding schedules for a given design input. When the grain of partitioning is a single time step, the schedule obtained is necessarily complete. For a larger grain of partitioning the schedules obtained may be partial. For subsequent allocation and binding complete schedules are needed. Most of the existing scheduling algorithms, like force-directed list scheduling (FDLS) [10], can be adapted to work with the partially scheduled DAG's generated by REPS. However, the performance of such modified heuristic algorithms may not match the performance of the original algorithm. The solutions obtained from this completion gives us ub estimates. If these match with the lb estimates obtained through DSE, we can terminate with accurate design points and schedules.

The ub's obtained for the partial schedules may differ from the lb estimates found by REPS. It is quite possible that for a time constraint and a set of hardware operators, as indicated by a design point, a feasible solution might not exist. Even if such a solution does exist, it might be missed out by the approximate scheduling algorithm. However, feasible solutions will be present in the neighborhood of a design point. We, therefore, resort to a systematic generation of nondominated (performance, FU requirement) design points and corresponding feasible schedules in the neighborhood of a design point reported by REPS and retained by the DSE mechanism. We refer to this as local exploration. We rely on existing scheduling algorithms, such as FDLS [10] or the scheduling method proposed in [9], and use them in an appropriate framework. Such a local exploration scheme should be capable of examining the neighborhood of a design point for

feasible nondominated solutions using approximate scheduling algorithms. The choice of polynomial time techniques here is emphasized because an exact method could be used otherwise to obtain the schedule in the first place.

Thus, after the first phase of DSE, we have a set of design points. With each design point, we also have the set of partitioned DAG's, which had led to its FU estimate component. At this juncture, we complete the schedules of these partitioned DAG's using standard algorithms like FDLS [10] or the scheduling method proposed in [9]. The solutions obtained from this completion gives us ub estimates. If these match with the lb estimates obtained through DSE, we can terminate with accurate design points and schedules. On the other hand, if the ub's and the lb's differ, we explore around the estimated design point for feasible schedules leading to nondominated ⟨performance, FU requirement⟩ design points. That is, we make limited search (in polynomial time) around the estimated design points obtained earlier. Our study of some list scheduling algorithms shows that these algorithms usually terminated with optimal solutions for small DAG's. Therefore, in our state space generation, we performed decomposition in a balanced manner to ensure that the subproblems generated after DSE are small and more suitable for existing scheduling algorithms. In Section V-C, we examine the local exploration scheme.

C. Local Exploration

Given a design point for which only a partial schedule is available, we first try to schedule using the available time and resource constraints to check for the existence of a solution. If such a schedule is found, then we are done. If case scheduling fails for the time and resource constraint, as indicated by the design point, then the time constraint as well as the resource constraint can be relaxed. The relaxation of these constraints also constitutes a search space. We have adopted a heuristic relaxation scheme. The algorithm works in polynomial time. This relaxation scheme effects both resource and time constraint relaxation. Otherwise, it initiates time relaxation on a copy of the design point, keeping the original one for resource relaxation. The time is relaxed in steps and for each new constraint, a schedule is found. The actual schedule found may not satisfy the constraint; anyway, the schedule along with the actual design point corresponding to the schedule is incorporated in the set of nondominated designs. It is necessary to incorporate a schedule, even if it does not satisfy the design constraint, to accommodate the inadequacy of the approximate scheduling algorithm, and ensure a proper termination of the approximate scheduling scheme. The time constraint is relaxed until the new design point is dominated by one of the designs in the set of nondominated designs. This marks the end of a run of time constraint relaxations.

Now, the resource constraint on the original design is relaxed and the entire process is repeated until the new resource constraint turns out to be dominated by one of the designs in the set of nondominated designs.

The approximate scheduling algorithm that has been used here is the force directed list scheduling algorithm [10].

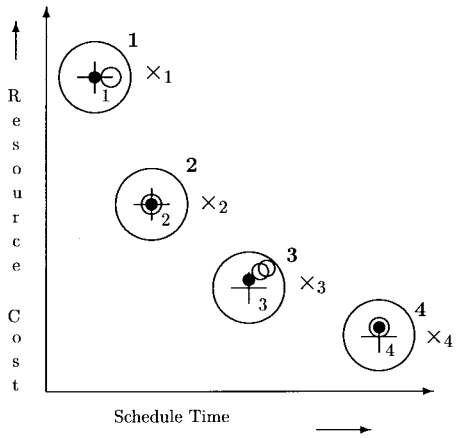

Fig. 2. Illustration of basic DSE scheme.

However, any other approximate scheduling algorithm can also be used. The quality of the design space actually will be governed by the quality of the approximate scheduling algorithm.

We summarize the working of the overall DSE technique as follows, referring to Fig. 2. In this figure, the filled circles (•) indicate the actual nondominated design points that we would like to uncover through DSE. For small problems where REPS can be run with window size $W = 1$, these points are directly obtained. We consider a scenario where REPS is invoked with $W > 1$. The design points returned by REPS are indicated by $+$ and $\times$. These are approximate design points. We do not consider the points indicated by $\times$ because these are dominated by the points marked $+$. The feasible schedules in the neighborhood of these points (indicated by the large circles) are now found by local exploration. These points are marked by empty circles It may be noted that some of these points coincide with the design points found by REPS (marked $+$). These are optimal schedules because the lb and the ub costs are identical. In other cases, these are distinct.

VI. Experimentation

The techniques proposed in this paper have been implemented and tested. The implementation has been done in C in a UNIX environment. We have performed DSE on some common examples, such as Facet [11], differential equation solver [5], and elliptic wave filter [12]. We now describe our experimentation.

Table I indicates the design points obtained after DSE of facet, diffeq., and elliptic wave filter, respectively. All these designs are for single-cycle implementations of the operations. The first two columns indicate design parameters. The design points obtained after DSE for an *example* are indicated in the block entitled "DSE results for *example*," while the actual results obtained after allocation and binding are indicated alongside, within round brackets, if they happen to be different from the corresponding DSE results. While computing the costs of the FU's, the cost of each hardware operator is taken as follows: $\text{cost}(/) = 160$, $\text{cost}(*) = 160$, $\text{cost}(+) = 20$, $\text{cost}(-) = 20$, $\text{cost}(<) = 10$, $\text{cost}(|) =$

TABLE I
DSE RESULTS

Num. of F.U.s	Num. of bus	Hardware opr. req. (values after Synthesis with GABIND, if different)	F.U. cost (cost after synthesis, if different)	Num. of time steps
DSE results for Facet				
2	6	1⋆, 1/, 1+, 1-, 1&, 1—	380	5
3	9	1⋆, 1/, 2+, 1-, 1&, 1—	400	4
		1⋆, 1/, 1+, 1-, 1&, 1—	380	5
DSE results for Diffeq.				
2	6	2⋆, 1+, 1-, 1<	370	6
		1⋆, 1(2)+, 1-, 1<	210 (230)	7
3	9	2⋆, 1(2)+, 1-, 1<	370 (390)	4
		1⋆, 1+, 1(2)-, 1<	210 (230)	7
DSE results for Elliptic Wave Filter				
3	9	2⋆, 3+	380	18
		2⋆, 2+	360	19
		1⋆, 2+	200	21
		1⋆, 1+	180	27

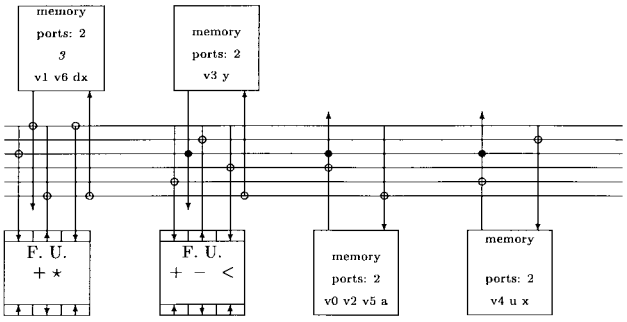

Fig. 3. Diffeq. data paths for two FU's and seven time steps.

10, and $\mathrm{cost}(\&) = 10$. The allocation and binding has been done by the allocation and binding tool GABIND [1]. Each block of rows in the table, under the name of the particular design, indicates the design points obtained for a particular set of parameters for that design. For facet, the design points obtained by DSE match the actual designs obtained after allocation and binding. This is also true for the elliptic wave filter example. For diffeq., the actual implementations of the design points indicated in the last three rows in the blocks under "DSE results for diffeq." of Table I, require an additional adder in each case. For two FU's and seven time steps for diffeq., the operations scheduled in three time steps were as follows: $\langle * + \rangle$,$\langle * - \rangle$, and $\langle + - \rangle$. Therefore, although at most one "+" and one "−" are scheduled in any time step, it is not possible to have an FU configuration using two FU's where at least a "+" or "−" is not repeated. For the case with three FU's and seven time steps, however, such a problem did not exist. Yet an additional "+" was used to keep the switch cost low. The implementation of diffeq. by GABIND using two FU's in seven time steps is especially nice, requiring only three interconnection switches (filled circles in Fig. 3). The design point indicated in the second row of Table I for the diffeq. example is for designing with only two FU's, whereas there are four types of operations distributed over seven time steps. For the design point indicated in the third row for diffeq. in Table I, the number of time steps is four, exactly equal to the length of the critical path in the data flow graph.

VII. CONCLUSION

A given behavioral specification can have a large number of RTL implementations. We can partially characterize RTL implementations by means of design parameters like the number of FU's and the number of buses, when we consider a bus-based data path. Even for a given set of design parameters, a number of designs are possible that differ in their hardware requirement and performance. These designs constitute a design space that needs to be systematically explored to find the nondominated designs. The early part of this DSE problem revolves around the basic scheduling problem, which is NP-hard.

We have proposed a scheme for performing this exploration using a combination of controlled search and approximate scheduling techniques. The search is based on DFBB. DFBB has the advantage of requiring minimum storage space to run. It is necessary to conserve space because the storage for a single (partial) solution is, itself, considerable. We have used a balanced problem decomposition scheme for the DFBB. This has the advantage of partitioning the original problem into smaller subproblems of nearly equal sizes. We have applied our DSE techniques to some common examples, e.g., facet, differential equation solver, and elliptic wave filter, and constructed data paths from the schedules obtained. We have noted close conformity with the estimates obtained with DSE and the actual hardware used in the data paths.

REFERENCES

[1] C. A. Mandal, P. P. Chakrabarti, and S. Ghose, "Allocation and binding for data path synthesis using a genetic approach," in *Proc. VLSI Design'96*, pp. 122–125.
[2] B. S. Stewart and C. C. White, "Multiobjective A*," *J. Assoc. Comput. Mach.*, vol. 88, no. 4, pp. 775–814, 1991.
[3] C. A. Mandal, P. P. Chakrabarti, and S. Ghose, "Complexity of scheduling in high level synthesis," *VLSI Syst. Des.*, vol. 7, no. 4, pp. 337–346, 1998.
[4] A. V. Aho, R. Sethi, and J. D. Ullman, *COMPILERS Principles, Techniques and Tools.* Reading, MA: Addison-Wesley, 1987.
[5] P. G. Paulin, "High level synthesis of digital circuits using global scheduling and binding algorithms," Ph.D. dissertation, Dept. Electron., Carleton Univ., Ottawa, Ont., Canada, 1988.
[6] F. J. Kurdahi and A. C. Parker, "Real: A program for register allocation," presented at the Proc. 24th Design Automation Conf., 1987.
[7] F.-S. Tsai and Y.-C. Hsu, "Star—An automatic data path allocator," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 1053–1064, Sept. 1992.
[8] C. V. Ramamoorthy, K. M. Chandy, and M. J. Gonzalez, "Optimal scheduling strategies in a multiprocessor system," *IEEE Trans. Comput.*, vol. C-21, pp. 137–146, Feb. 1972.
[9] A. Kumar and M. Balakrishnan, "A novel integrated scheduling and allocation algorithm for data path synthesis," in *Proc. VLSI Design '91*, New Delhi, India, pp. 212–218.
[10] P. G. Paulin and J. P. Knight, "Algorithms for high-level synthesis," *IEEE Design & Test Mag.*, vol. 6, pp. 18–31, Dec. 1989.
[11] C. J. Tseng and D. P. Siewiorek, "Automated synthesis of data paths in digital systems," *IEEE Trans. Computer-Aided Design*, vol. CAD-5, pp. 379–395, July 1986.
[12] S. Y. Kung, H. J. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing*, Englewood Cliffs, NJ: Prentice-Hall, 1984.

Chittaranjan A. Mandal received the B.Tech., M.Tech., and Ph.D. degrees from the Indian Institute of Technology, Kharagpur, India, in 1987, 1990, and 1995, respectively.

He is currently a Reader in the Computer Science and Engineering Department, Jadavpur University, Calcutta, India, and a Visiting Researcher in the Department of Information Systems and Computing, Brunel University, Uxbridge, U.K. His current interests include computer-aided design (CAD) for very large scale integration (VLSI), evolutionary computing, and design of algorithms.

Dr. Mandal was the recipient of a 1995 Commonwealth Scholarship.

Partha Pratim Chakrabarti (M'90) received the B.Tech. and Ph.D. degrees from the Indian Institute of Technology, Kharagpur, India, in 1985 and 1989, respectively.

He is currently a Professor in the Computer Science and Engineering Department, Indian Institute of Technology. His current interests include artificial intelligence, CAD for VLSI, and design of algorithms.

Dr. Chakrabarti received the 1985 President of India Gold Medal, the 1991 INSA Young Scientist Medal, the 1995 Anil K. Bose Award from INSA, and the 1997 INAE Young Engineer Medal. He was also the recipient of the 1997–1998 Swarnajayanti Fellowship.

Sujoy Ghose (M'91) received the B.Tech. degree in electronics and electrical communication engineering from the Indian Institute of Technology, Kharagpur, India, in 1976, the M.S. degree from Rutgers University, Piscataway, NJ, in 1978, and the Ph.D. degree in computer science and engineering from the Indian Institute of Technology, in 1987.

He is currently a Professor in the Department of Computer Science and Engineering, Indian Institute of Technology. His research interests include design of algorithms, artificial intelligence, and computer networks.