References

[1] D.D. Gajski, N.D. Dutt, and B.M. Pangrle, "Silicon compilation (tutorial)," in Proc. IEEE 1986 Custom Integrated Conf. (Rochester NY), May 1986, pp. 102-110.

[2] P.G.Paulin and J.P.Knight, "Force-directed scheduling in automatic data path synthesis," in Proc. 24th Design Automation Conf. (Miami Beach, Fl), July 1987, pp. 263-270.

[3] C. Tseng and D.P. Siewoirek, "Automated Synthesis of data paths in digital systems," IEEE Trans. Computer Aided Design, Vol CAD-5, pp. 379-295, July 1986.

[4] C.H. Genotys and M.I. Elmasry, "A VLSI methodology with testability constraints," in Proc 1987 Canadian Conf. VLSI (Winnipeg), Oct. 1987.

[5] P.Marwedel, "A new synthesis algorithm for the MIMOLA software system," in Proc. 23rd Design Automat. Conf. (Las Vegas, NV), July 1986, pp. 271-277.

[6] S.Y. Kung, H.J. Whitehouse and T. Kailath, VLSI and Modern Signal Processing. Englewood Cliffsm NJ: Prentice Hall, 1985, pp. 258-264.

[7] Pierre G. Paulin and John P. Knight, "Force Directed Scheduling for th behavioral synthesis of ASIC's," IEEE Trans. Computer Aided Design, Vol.8, pp. 661-679, June 1989.

[8] J. Lee, Y. Hsu, and Y. Lin, "A new Integer Linear Programming Formulation for the Scheduling Problem in Data-Path Synthesis," Proc. of the Int. conf. on Computer-Aided Design, pp. 20-23, 1989.

[9] W.F.J. Verhaegh, E.H.L Aarts, J.H.M. Korst and P.E.R Lippens, "Improved Force Directed Scheduling," Proc. of European Design Automation Conf., pp. 430-435, 1991.

[10] I-C. Park and C-M. Kyung, "Fast and Near Optimal Scheduling in Automatic Data Path Synthesis," Proc. of the 28th DAC, pp. 680-685, 1991.

[11] K.H. Kernighan and S.Lin, "An Efficient Heuristic procedure for partitioning graphs," Bell system technical journal, vol. 49, pp. 291-307, February 1970.

[12] S. Davidson et. al., "Some experiments in local microcode compaction for horizontal machines," IEEE Trans. on Computers, pp. 460-477, July 1981.

[13] B.M. Pangrle and D.D. Gajski, "Slicer: A state synthesizer for intelligent silicon compilation," in Proc. of IEEE Int. Conf. on Computer Design, Oct. 1987.

[14] R. Jain, A. Mujumdar, A. Sharma and H. Wang, "Empirical evaluation of some high-level synthesis scheduling heuristics," Proc. of 28th DAC, pp. 210-215, 1991.

[15] S. Devadas and A.R. Newton, "Algorithms for allocation in datapath synthesis," IEEE Trans. on CAD of Integ. Cir. and Systems, vol. 8, pp. 768-781, July 1989.

[16] R. Camposano, "Path-Based Scheduling for Synthesis," IEEE Trans. on CAD of Integ. Cir. and Systems, vol. 10, no. 1, pp. 85-93, Jan 1991.

9 Summary and Open Problems

In this paper several algorithms have been described for scheduling operations into control steps. The broad areas that were focussed are: time constrained and resource constrained (using list techniques).

The ILP approach solves the problem optimally but had large execution times. On the other hand FDS is quicker but the optimality of the solution cannot be guaranteed. The IR algorithm can improve the design quality of an initial schedule generated by any algorithm. The list based resource constrained scheduling algorithms belong to a more complex class of algorithms than the time constrained algorithms. Among other algorithms discussed simulated annealing is easy to implement but has extremely large execution times. Path-based scheduling has high computational complexity but can produce good schedules.

Research in high-level synthesis and scheduling has been going on for more than a decade and yet there are many unsolved or partially touched issues, some of which are listed below:

- Pipelined scheduling : Optimized scheduling of pipelined behavioral descriptions is a recent development in the field of behavioral synthesis. Two simple type of pipelining (structural and functional) have only been attempted.
- Controller cost : Most scheduling algorithms do not consider the controller costs with is directly dependent on the controller style used during scheduling.
- Area constraints : The resource constrained algorithms could have better interaction between scheduling and floorplanning.
- Realism : A lot scope scope is there for efficient ways scheduling realistic design descriptions that contain several special language constructs. Work is also needed on using more realistic libraries and cost functions. Scheduling algorithms must also be expanded to incorporate different target architectures.

execution only one branch gets executed. Therefore an effective scheduling algorithm should share resources among mutually exclusive operations. A behavioral description may also contain loop constructs that exhibit parallelism between different iterations of the loop. There are three ways of scheduling a loop:

1. Sequential execution : A simple approach is to directly schedule the loop body into control steps.
2. Loop unrolling : Here certain number of loop iterations are unrolled which results in a loop that has a larger body but fewer number of iterations. This can be done only when the length of the loop is already known.
3. Loop folding : Here intraloop parallelism is exploited by folding the loop. Successive iterations of the loop are overlapped in a pipelined fashion and executed.

Both loop unrolling and folding would increase the control costs. There is always a tradeoff between the execution speed and the control cost. Also the problem size is an important factor that would affect algorithms that do these optimizations.

path. Such intervals from different are combined using the same graph technique for the final CDFG. Introducing new control steps to this final CDFG would produce a unique schedule.

8 Some Important Scheduling Issues

In this paper so far scheduling has been discussed without actually considering realistic design models that would have FUs with varying delays, multifunctional units, behavioral description that have more than just straight line code. These issues have been discussed in short in the following sections.

8.1 Functional Units with varying delays

Each functional unit will have a different delay and therefore assuming that an operation assigned to a control step would take the same time as another operation. This assumption would lead to a clock cycle that is unusually lengthened by the slowest unit in the design. The following three approaches are followed to solve this problem:

1. Multicycling : If the clock cycle is shortened to allow fast operations to execute then the slower operation would take multiple clocks and hence are called multicycle operations. However input latches are needed in front of the multicycle functional units to hold its operands units its result is available. This would in turn increase the size of the control logic.
2. Chaining : Two or more operations could be allowed to perform sequentially in a single control step. Since the output of one FU has to be fed to another FU, they should be directly connected.
3. Pipelining : A FU may have stages in it separated by latches. This makes it possible to execute two operations in the same FU since they operate in two different stages. Pipelining is a simple yet efficient technique to increase parallelism.

8.2 Multi-functional Units

So far its been assumed that a FU can perform only one operation but in practice there are several cost effective multi-functional being used. For this purpose the scheduling algorithms could be technology based so that they can explore the library of components. Operations in the critical path could be assigned faster functional units than those not in the critical path. Also the scheduling algorithm could try to use the same multi-functional for two data independent operations which are in two different control steps.

8.3 Realistic Design Descriptions

Behavioral description usually contain conditional and loop constructs. A conditional construct (similar to an "case" statement) results in several branches that are mutually exclusive. During

6.3 Comparison of List Scheduling Algorithms

List scheduling is one of the most popular methods for scheduling operations under resource constraints. A table comparing these two approaches is given below.

A COMPARISON OF THE LIST SCHEDULING ALGORITHMS		
	List-Based	Static List
Computational Complexity	High	Not very high
Quality of schedule	Mostly Optimal	Mostly Optimal
Space Complexity	Very High	Not High
Input Problem Size	Any size	Any size
Execution Speed	Slower than FDS	Faster than List-Based
Technique Used	Many priority lists	One priority list

7 Other Scheduling Algorithms

In addition to the scheduling techniques discussed above several other solution exist to the scheduling problem. Two important approaches: Simulated Annealing and Path-Based scheduling have been briefly discussed in following sections.

7.1 Simulated Annealing

A simulated annealing based scheduling is described in [15]. Schedules can be represented as a two dimensional table of control steps versus available functional units. Scheduling can now be viewed as a placement problem where the table entries are filled by operations. Since a FU can perform only a single operation in a given control step an entry cannot be occupied by more than operation. Beginning with an initial schedule the algorithm iteratively modifies the table, each time displacing an entry and determining the cost of the displacement. A modification is accepted with a probability eventhough the resulting schedule is not better, so that it is possible to search the solution space by climbing out of local minimums in search of a globally optimal solution. Although simulated annealing is robust, it requires long execution times.

7.2 Path-Based Scheduling

Path-based scheduling algorithm [16] minimizes the number of control steps needed to execute the critical path in the CDFG. All possible execution paths are extracted and scheduled independently. The schedules of different paths are combined to generate the final schedule. The algorithm generates constraints between nodes that have to be assigned in different control steps. The problem of introducing minimum control step constraint is transformed into a clique-partitioning graph problem. Nodes represent constraint interval and edges represent constraint interval overlapping. A clique partitioning solution would then indicate the minimum overlapping of intervals in a given

6.1 List-Based Scheduling

Scheduling techniques described in [12] belong to list-based scheduling algorithms. List based scheduling is a generalization of the ASAP algorithm with the inclusion of resource constraints. A list based algorithm maintains a priority list of ready nodes, i.e., nodes whose predecessors have already been scheduled. The priority list for each operation is sorted with a priority function that resolves any resource contentions. In each iteration, operations with higher priority are scheduled first and lower priority operations are deferred to later control steps. Scheduling an operator to a control step makes other successor operations ready, which will be added to the priority list.

A simple priority function can be inversely proportional to the mobility, i.e., the greater the mobility the lesser the priority and vice-versa. This would ensure that operations with large mobility are deferred to later control steps because the number of control steps into which they could go is more. Basically this is same as pruning one control step from its mobility range and deferring the decision later. The SLICER system [13] developed at the University of Illinois uses this type of priority function.

As it can be seen the success of a list scheduler depends mainly on the priority function used. Mobility is a good priority function because smaller the mobility higher the urgency for scheduling. An alternative mobility function could use the length of the longest path from the operation node to a node with no immediate successor. There are many other priority functions that have been proposed. The time and space complexity for this approach is slightly more because several lists have to be maintained dynamically.

6.2 Static List Scheduling

This approach [14] starts by creating a single large list before starting scheduling. This algorithm differs from ordinary list-based scheduling in both the assignment of control steps as well as in maintaining the priority list. The algorithm first uses the ASAP and ALAP algorithm to obtain the least and the greatest possible control step assignments (LCS and GCS respectively) for each operation. The algorithm then sorts all the operations in ascending order using the GCS labels as the primary key and then sorts each set of operations with same GCS labels, in descending order with the LCS labels as the secondary key.

Once the priority list is created the operations are scheduled sequentially starting with the last operation in the priority list (i.e., with the highest priority). In each iteration when the limit for the number of resources has been reached the rest of the operations are deferred to later control steps. the type of the operation matters because eventhough an addition operation is of lower priority them multiply operations but has the highest priority among addition operations then it might be scheduled before the multiply operations if an adder unit is available. This scheduling technique has an advantage over the ordinary list based approach: A list is constructed statically only once and not grown dynamically.

constraints. For example the five possible moves in the schedule in figure-5c is shown by the arrow marks. A random move is picked, moved and locked temporarily in that position. Similarly all other moves are made until all operations are locked. The costs for all these moves are calculated and the move that produced the maximum gain (or maximum reduction the cost) is chosen and all moves in the sequence until this move are moved permanently. Then all operations are unlocked and the whole procedure is repeated with this new schedule. Very much like the KL method, the quality of the result produced by this algorithm depends on the initial solution.

There have been two enhancements made to this algorithm: (i) Since the algorithm is computationally efficient it can be run many times with different initial solution and the best solution can be picked. (ii) A better look-ahead scheme that uses a more sophisticated strategy of move selection as in [kris84] can be used.

5.4 Comparison of Time Constrained Scheduling Algorithms

All the time constrained scheduling algorithms use the ASAP and ASAP algorithms in their initial phases. A comparison table of these approaches is given in the table below. Look at them it can be said that FDS is the best on an overall basis and hence the most popularly used.

A COMPARISON OF THE TIME-CONSTRAINED SCHEDULING ALGORITHMS			
	ILP	FDS	IR
Computational Complexity	Very High	Low	High
Quality of schedule	Optimal	Opt./Sub-Opt.	Near Opt.
Space Complexity	Medium	Low	Very High
Input Problem Size	Small	Any size	Cannot be large
Execution Speed	Low	High	Medium
Technique Used	Math. Prog.	Constructive	Refinement
Popularity	Least used	Most used	Medium

6 Resource Constrained Scheduling

In applications where the design is restricted by the silicon area, resource constrained scheduling algorithms will be useful. The goal of these algorithms is to produce a design with the best possible performance but still meet the given resource constraints. The schedule is gradually constructed, one operation at a time, so that the resource constraints and data dependencies are not violated. In each control step the number operations scheduled in any control step does not exceed the number of FUs available. Also the algorithm ensures that all the predecessors are scheduled before the before that operation.

The following sections describe two popular scheduling algorithms in this category.

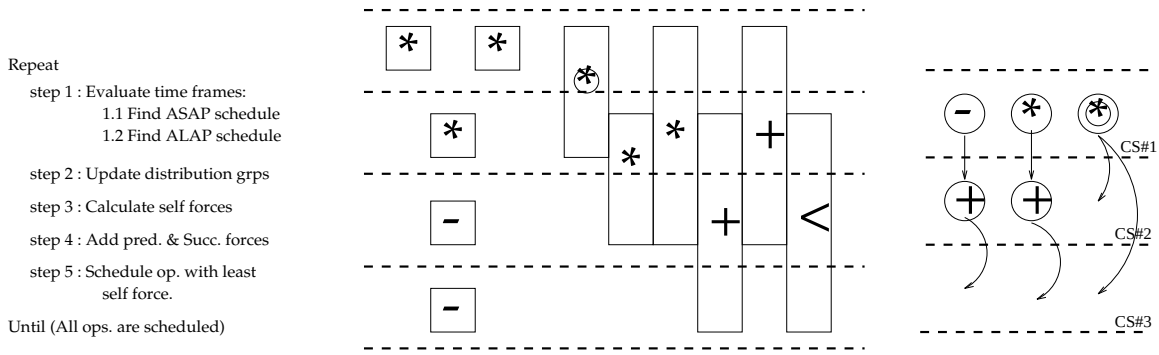

Figure 5: (a) FDS outline (b) Distribution graph (c) Sub-opt schedule example

in figure-4a is shown in figure-5b. For each operator the force is calculated using the formulae: DG(i) = $\sum_{OpnType}$ Prob(Opn,i), where Prob is the probability of an operation in control step i. The probability of each vertical bar in the distribution calculated using the formula: Prob = (1 / Mobility Range). The force for each operation is calculated using the formula: Force(i) = DG(i) * x(i), where x(i) is the probability of an operation in control step i, and is negative if it is being removed from control step i and positive if it is being added. The total force for an operation is calculated by adding the forces of the predecessor and successor operations with its self force.

For example the force for the circled multiply operation is calculated as follows: Probability = 1/2 = 0.5, DG(1) = 1 + 1 + 0.5 + 0.33 = 2.83, DG(2) = 2.33. Therefore Force(1) = (DG(1) * x(1)) + (DG(2) * x(2)) = (2.833 * +0.5) + (2.33 * -0.5) = +0.25. Similarly Force(2) = -0.25. When the total force for all the operations are calculated, operation with the least force is scheduled in that control step. the algorithm stops when all the operations in the DFG have been scheduled.

The complexity of the FDS algorithm is O(cn^2) where c is the number of control steps and n is the number of operations in the DFG. The FDS algorithm not always produces an optimal solution. For example the DFG shown in figure-5b has a distribution graph where the forces for scheduling the circled operation into control steps 2 or 3 are the same. In this case the algorithm cannot estimate the best choice accurately. The best choice here would be to assign it in control step 3 so that each control step has only one type of FU. If it happens to take the other choice then it would result in a sub-optimal schedule where there are two FUs of a type in control step 2 or 3.

Alternatively the strategy could be changed by pruning one control step from its mobility range and postponing the decision to a later stage as in [9]. The FDS algorithm never backtracks on its decisions and hence is classified under constructive algorithms.

5.3 Iterative Rescheduling

Due to the lack of a look ahead scheme the FDS algorithm is likely to produce a sub-optimal solution this weakness can be coped by rescheduling some operations in each partial schedule. The iterative rescheduling (IR) method [10] based on the previously proposed graph-bisection problem by Kernighan and Lin [11] proceeds by rescheduling one operation at a time. Any initial schedule is taken each operation is scheduled into an earlier or later step keeping in mind the data dependency

2. Constructive heuristics : Force directed scheduling method described in section 5.2 is an example of a constructive heuristic.

3. Iterative Refinement : An example of this type, iterative rescheduling, is given section 5.3.

The following sections summarize each of these algorithms and finally compare them.

5.1 Integer Linear Programming

The integer linear programming (ILP) method [8] tries to find an optimal schedule using a branch-and-bound search algorithm. It also involves some amount of backtracking, i.e., decisions made earlier are changed later on. A simplified formulation of the ILP method is given below.

First it calculates the mobility range for each operation $M = \{S_j \mid E_k \leq j \leq L_k\}$, where E_k and L_k are the ASAP and ALAP values respectively. The scheduling problem in ILP is defined by the following equations:

$$Minimize\ (\sum_{k=1}^{n}(C_k * N_k))\ \ and \sum_{E_i \leq j \leq L_i} x_{i,j} = 1,\ \forall i,\ 1 \leq i \leq n,\ number\ of\ operations,$$

where $1 \leq k \leq m$ operation types are available, and N_k is the number of FUs of operation type k and C_k is the cost of each FU. Each $x_{i,j}$ is 1 if the operation i is assigned in control step j and 0 otherwise. Two more equations that enforce the resource and data dependency constraints are:

$$\sum_{i=1}^{n} x_{i,j}\ \leq\ N_i\ and\ ((q\ *\ x_{j,q}) - (p\ *\ x_{i,p})) \leq -1,\ p < q,$$

where p and q are the control steps assigned to the operations x_i and x_j respectively.

The ILP formulation increases rapidly with the number of control steps. For unit increase in the number of control steps we will have n additional x variables. Therefore the time of execution of the algorithm also increases rapidly. In practice the ILP approach is applicable only to very small problems.

If it is possible to eliminate the backtracking involved in the ILP method considerable amount of computation time could be saved. Heuristic methods do the job by scheduling one operation at a time based on some criterion. The following section describes one such method.

5.2 Force Directed Scheduling

The Force directed scheduling (FDS) is a heuristic method [7] that is a very popular scheduling technique for time constrained scheduling. The main goal of this algorithm is to reduce the total number of FUs used. This algorithm achieves its goal by uniformly distributing the operations of the same type over the available control steps. This algorithm is briefly explained below.

A simple outline of the FDS algorithm is given in figure-5a. Using the ASAP and the ALAP values the distribution graphs for the operators are obtained. A distribution graph for the DFG

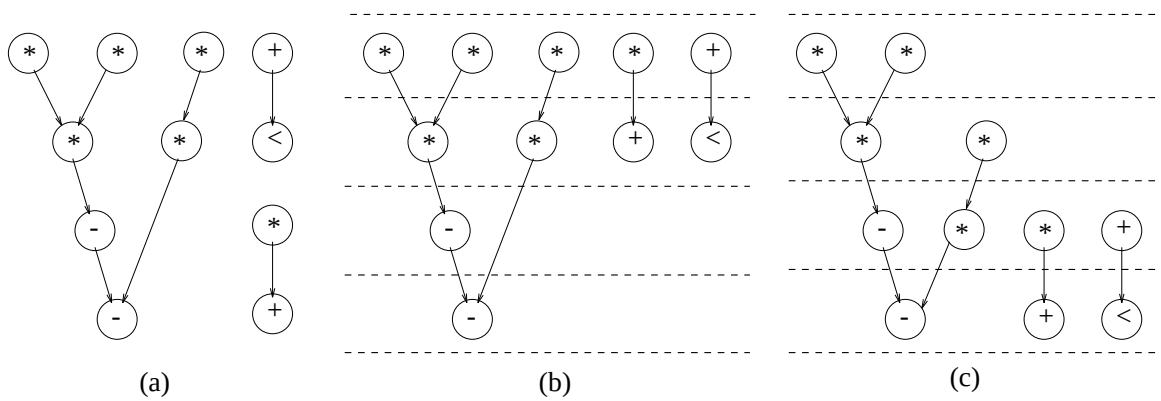

Figure 4. (a) Data Flow Graph (b) ASAP schedule (c) ALAP Schedule

The ASAP Algorithm starts with the highest nodes (that have no parents) in the DFG and assigns time steps in increasing order as it proceeds downwards. It follows the simple rule that a successor node can execute only after its parent has executed. This algorithm clearly gives the fastest schedule possible. In other words, it schedules in least number of control steps but never takes into account the resource constraints. It has also been stated [7] that this technique has proved useful for near-optimal microcode compaction.

4.2 ALAP Algorithm

This approach is a refinement of the ASAP scheduling concept with conditional postponement of operations. In the MIMOLA system [5], postponement occurs whenever the operation concurrency is higher than the number of available functional units. Kung et. al. use a similar scheme [6] assigning "as late as possible" (ALAP) levels for the operations. Figure-3c shows an ALAP schedule for the DFG in figure-3a.

The ALAP algorithm works exactly in the same way as the ASAP algorithm expect that it starts at the bottom of the DFG and proceeds upwards. This algorithm gives the slowest possible schedule that takes the maximum number of control steps. However this doesn't necessarily reduce the number of functional units used.

5 Time Constrained Scheduling

Time constrained scheduling is also called as fixed-control-step approach. Time constrained scheduling is important for designs targeted towards applications in real-time systems like digital signal processing systems where the main objective is to minimize the cost of the hardware.

Time constrained scheduling algorithms usually use three different techniques:

1. Mathematical Programming : One of the most popular techniques is the integer linear programming method described in section 5.1.

3 Classification Of Scheduling Algorithms

Over the years researchers have tried to come up with various kinds of solutions [2, 7, 8, 10, 12, 14, 15, 16] to the scheduling problem. Several algorithms have be put forth and each one has it own advantages and disadvantages. Scheduling algorithms can be broadly classified into time constrained and resource constrained scheduling, based on the goal of the scheduling problem. In time constrained scheduling the number of FUs are minimized for a fixed number of control steps. On the other hand, in resource constrained scheduling the number of control steps are minimized for a given design cost (number of functional and storage units). I have tried to classify the well known scheduling algorithms into categories as shown in figure-3.

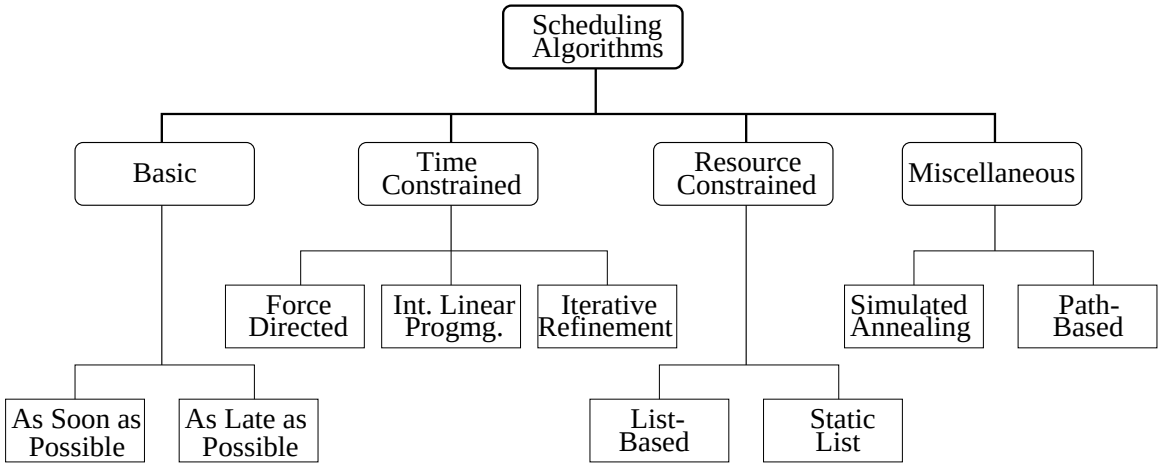

Figure 3: Classification of Scheduling Algorithms

There have been other classifications [7] of the scheduling approaches also. The following sections give a brief summary of each of these algorithms and compare them.

4 The Basic Scheduling Algorithms

As we have seen DFGs expose parallelism in the design. Consequently each node has a range of control steps in which it can be assigned. Most of the algorithms that will be described later require the earliest and the latest bounds within which operations in the DFG can be scheduled. The first and simplest schemes that are used to determine these bounds are called the As Soon As Possible (ASAP) and the As Late As Possible (ALAP) algorithms.

4.1 ASAP Algorithm

A simple scheme is to schedule operations "as soon as possible" (ASAP), as is done in Carnegie Mellon University's Emerald/Facet system [3] and CATREE system [4] system from the University of Waterloo. Figure-3b shows an ASAP schedule for the DFG in figure-3a.

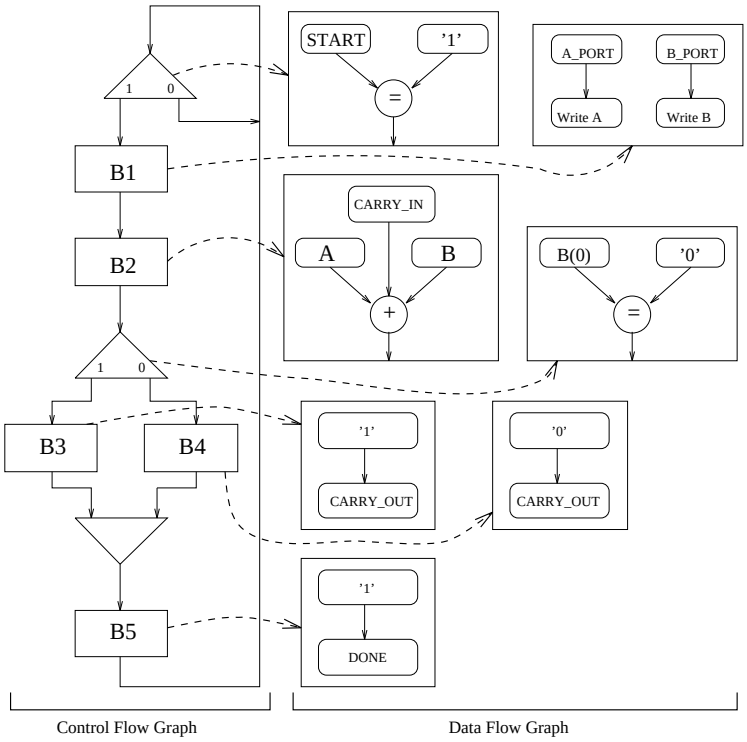

Figure 1: CDFG for the ADDER example

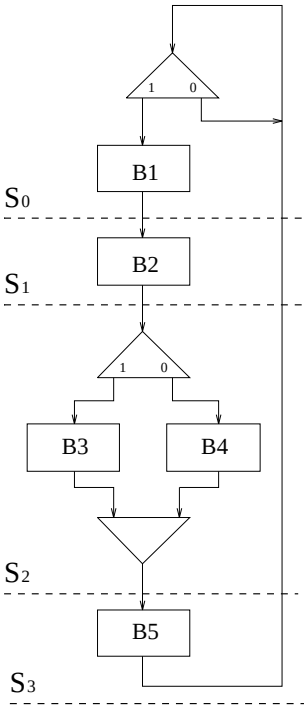

Figure 2: The scheduled CDFG

2.2 A Scheduled CDFG Example

Figure-2 shows the same CDFG (in figure-1) scheduled into four synchronous time steps labeled S_0, S_1, S_2, S_3. In state S_0 the adder busy-waits for the signal START to rise. When START is asserted, the adder is initialized with the values in A_PORT and B_PORT. The addition operation takes place in state S_1. In the next state S_2, the CARRY_OUT port is assigned the carry of addition, depending on the value of the most significant bit of RESULT. Finally, in state S_3 the addition is completed by asserting the DONE signal. The CDFG after scheduling is annotated with state labels, thus maintaining a link between the abstract behavior and the states in the design.

After partitioning the CDFG representation of the behavioral description into control steps, a separate functional unit (FU) is assigned to execute each operation in that control step. The total number of FU's needed for the design is proportional to the maximum number of FU's in any control step. Thus, if we can schedule more operations in a control step greater will be the FU's but fewer number of control steps would be necessary for the design. On the other hand if fewer operations are scheduled in a control step, fewer will be the FU's required but the number of control steps needed to finish all the operations would be more. *Scheduling is an important task in high-level synthesis because it impacts the tradeoff between design cost and performance.*

```
entity ADDER is
     port( A_PORT, B_PORT,
           CARRY_IN, START : in bit;
           SUM,
           CARRY_OUT, DONE : out bit;
     );
end ADDER;

architecture BEHAVIORAL of ADDER is
begin
    process
     variable A, B : bit;
     variable RESULT : bit_vector(1 downto 0);
    begin
     wait until (start = 1);
     A := A_PORT;
     B := B_PORT;

     RESULT := A + B + CARRY_IN;

     if (RESULT(1) = 0) then
       CARRY_OUT <= 0;
     else
       CARRY_OUT <= 1;
     endif;

     DONE <= 1;
    end process;
end BEHAVIORAL;
```

The process statements in VHDL are executed sequentially but the statements without any data dependencies can be executed concurrently. For example, the assignment statements for the variables 'A' and 'B' can be executed concurrently. This implicit parallelism and control flow in the input HDLs can be nicely captured in a CDFG as shown in figure-1.

The CDFG has two parts: the control flow graph and the data flow graph. Each entry in the control flow graph has a corresponding data flow graph. The order of execution of the process statements are captured in the control flow graph. For example, the assignment statements for the variables 'A' and 'B' can be executed only when control comes out of the 'wait statement'. This has been clearly captured in the CDFG shown in figure-1.

The following section describes the CDFG of the adder after it has been partitioned into control steps.

Scheduling is one of the most important and primary tasks in high-level synthesis. The following section gives a brief description of what scheduling is and why it is necessary.

1.2 Scheduling

A Finite State Machine with Datapath (FSMD) model is the most popular one that is used to describe digital systems at the register transfer level. It consists of an FSM called the control unit and a datapath. The datapath consists of the storage and functional units necessary for the system. The FSM consists of a set of states, a set of transitions between states, and a set of actions (involving the datapath) associated with each transition.

Scheduling can be described as the process of dividing the intermediate representation into states and control steps, in such a way that it can directly synthesized into an FSMD model. In other words scheduling does a temporal mapping of the given representation. A behavioral description and hence the intermediate representation consists of a sequence of operations to be performed by the synthesized hardware. The task of scheduling, partitions these operations into time steps such that each operation is executed in one time step. Each time/control step corresponds to one state of the controlling finite state machine in the FSMD model.

The rest of this paper discusses in detail the general scheduling problem with the help of an example, classifies the various scheduling algorithms and summarizes and compares them and finally discusses the open problems in scheduling.

2 The Scheduling Problem

The behavioral description of a design is compiled into a canonical intermediate representation that can preserve the original behavior of the input HDL specification, while allowing addition of synthesis results through various refinements. Scheduling algorithms then partition this canonical intermediate representation so that each partition can be executed in one control step.

2.1 A Canonical Intermediate Representation

The behavioral description of a design is usually given in a hardware description language (HDL) like VHDL. Since there can be a lot of variation between the semantics of the input HDLs and the target architectures a canonical intermediate representation is needed. The most popular canonical intermediate representation used is called a Control Data Flow Graph (CDFG) that captures all the control and dataflow dependencies of an input behavioral description.

Consider the following VHDL description of a pseudo full-adder that has a START and a DONE interface signals.

1 Introduction

VLSI technology has advanced to such a state that it would be extremely complex to design digital systems starting at the transistor level or at the logic level. There has been an ever increasing need for design automation on more abstract levels where the functionality and tradeoffs can be clearly stated. This led to the development of CAD algorithms that could search the design space more thoroughly and find nearly optimal designs. Therefore, automation of the design process from conceptualization to silicon became more important and necessary. Designers provided a top-down methodology, where they describe the intent of the design and let CAD tools add detailed physical structure to it. This method of synthesizing systems from a design description became better suited for the design of complex systems. At this point of evolution, VLSI technology has reached a point where high-level synthesis (HLS) of VLSI chips and electronic systems is becoming more cost effective and less time consuming than being fully hand designed by a group of designers.

In the past decade there has been a lot of activity going on in the area of high-level synthesis [1, 2, 3, 4, 5, 13] and HLS is becoming an increasingly popular research topic. The following section briefly describes the various tasks involved in high-level synthesis.

1.1 High-Level Synthesis

High-level synthesis can be described as the process of translation of a behavioral description into a structural description that consists of a set of connected components called the data-path and a controller that sequences and controls the functioning of these components. High-level synthesis starts at the systems level and proceeds downwards to register transfer (RT) level, logic level and finally circuit level, each time adding some additional information needed at the next level of synthesis. The five major tasks involved in high-level synthesis are described below. The first three steps lead to the data-path formation and the last step leads to the formation of the controller.

1. Compilation : Compilation involves translation of the design description into an intermediate representation that is most suitable for high-level synthesis.
2. Partitioning : Partitioning deals with division of the intermediate representation (i.e, the behavioral description or the design) into sub-representations in order to reduce the problem size.
3. Scheduling : Scheduling partitions the intermediate representation into time steps, thereby generating a finite state machine model.
4. Allocation : Allocation though closely intertwined with scheduling, involves partitioning of intermediate representation with respect to space (hardware resources) which is also known as spatial mapping.
5. Control generation : Finally, this step involves the derivation of the controller that sequences the design and controls the functional and storage units in the datapath.

Scheduling Algorithms For High-Level Synthesis

Term Paper
ECE834
Course: Digital Design Environments
Instructor: Dr. Ranga Vemuri

by
Sriram Govindarajan

Dept. of ECECS
University of Cincinnati
Cincinnati, OH 45221–0030

March 17, 1995

Abstract

Scheduling has long been recognized as a very important step in the high-level synthesis process. A wide variety of algorithms exist in the literature for efficiently performing the task of scheduling in high-level synthesis. The objective of this paper is to present a comprehensive survey of the various scheduling techniques currently known. These algorithms have been classified into four different categories: Basic scheduling algorithms, Time constrained, Resource constrained and Miscellaneous. The Basic algorithms are not used individually for scheduling but are used in the initial phase of other algorithms. In each category a set of algorithms are described briefly with examples. A comparison of the algorithms under each category has also been done. Finally, important issues in scheduling, related to improving performance and exploiting parallelism are discussed.

General Terms: Design, Performance, Parallelism, Algorithm, Constraints
Additional Keywords: VLSI (Very Large Scale Integrated Circuits), CAD(Computer Aided Design), HLS(High-level Synthesis), FSMD(Finite State Machine with Datapath), VHDL(VHSIC Hardware Description Language), CDFG(Control Data Flow Graph), ASAP(As Soon As Possible), ALAP(As Late As Possible), ILP(Integer Linear Programming), FDS(Force Directed Scheduling), IR(Iterative Re-scheduling)