An Efficient List-Based Scheduling Algorithm for High-Level Synthesis

Azeddien M. Sllame, Vladimir Drabek
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno, Czech Republic
Sllame@fit.vutbr.cz

Abstract

Scheduling is considered as the most important task in high-level synthesis process. This paper presents a novel list-based scheduling algorithm based on incorporating some information extracted from data flow graph (DFG) structure to guide the scheduler to find near-optimal/optimal schedules quickly. We have developed a novel approach based on DFG analysis that is totally done as preparation phase. This DFG analysis information includes: every node knows its successor and its predecessor, total number of successors, and the tree which it belongs to, where trees are constructed from every output operation from the constructed DFG. Incorporating this knowledge in the priority functions of the scheduler guided the scheduler to make the correct choice of the perfect operation to be scheduled next.

1. Introduction

High-level synthesis (HLS) is the design task of mapping an abstract behavioral description of a digital system onto a register-transfer-level (RTL) design to implement that behavior. By using HLS tools in the current state-of-the-art CAD flow systems the designers are able to explore the design space more efficiently. The main procedure of HLS can be divided into four subtasks. The first subtask is to describe the behavior of the digital system using a hardware description language (e.g. VHDL). This step is usually followed by a translation of the description into a graph-based representation called the control data flow graph (CDFG), which represents the control flow and data dependence relations between the operations composing the system behavior. The *scheduling* is the next subtask, where each operation in the CDFG is assigned to one control step (c-step), that is, a clock cycle during which it will be executed, while satisfying the precedence relations between operations and preserving the design specified constraints. For that reason, scheduling directly controls the throughput rate of the produced RTL design. Likewise, scheduling determines the cost-speed tradeoffs of the given design. In speed constrained case the scheduler will execute many operations in parallel in order to satisfy the specified speed, while in case of limited chip area the scheduler will serialize the design execution in order to meet the chip area constraints. The third subtask is *allocation*, in which the desired number of functional units, storage elements, and required interconnections for the scheduled digital system are allocated. The forth subtask is *binding*, in this phase the operations are assigned to functional units, values to storage elements, and a complete data path is produced by interconnecting all components together. Finally, a control unit is synthesized to synchronize the executions of the produced datapath.

In this paper, we are concentrated on the scheduling process because the main design decisions such as the number of hardware resources, clock cycle time, and implementation styles (pipeline, multi-cycle operation, etc.) are made during the scheduling process [4]. In addition, these decisions have strong influence on the following data-path allocation and binding subtasks. We propose a new scheduling algorithm, which is very effective for data-driven algorithms such as those found in digital signal processing algorithms.

This paper is organized as follows: Section 2 reviews previous work and related research. Section 3 defines the problem and the cases handled by the proposed scheduling algorithm. Section 4 illustrates the main procedure of the new proposed scheduling algorithm. Extension to realistic scheduling features is given in Section 5. Various experiments and results are shown in Section 6. In Section 7, some points about design space exploration are illustrated. Finally, concluding remarks closing the paper results are given in Section 8.

2. Basic scheduling techniques

In this section, we are going to survey commonly used scheduling algorithms in state-of–the-art HLS systems, but

Proceedings of the Euromicro Symposium on Digital System Design (DSD'02)
0-7695-1790-0/02 $17.00 © 2002 IEEE

the intention is only to give an overview of the different approaches, not to be exhaustive. These important scheduling techniques include *list scheduling* [3], *force-directed scheduling* [8], and *path-based scheduling* [9]. A brief description of *integer linear programming* (ILP) [1] approach is also provided. The simplest scheduling technique is as soon as possible (ASAP) scheduling, ASAP schedules each operation, one at a time, into the earliest possible control step. Another simple technique is called as late as possible (ALAP) scheduling. In contrast to ASAP, ALAP scheduling schedules the operations from the last control step toward the first. Both ASAP and ALAP algorithms are constructive heuristic algorithms and are used in HLS to solve the *unconstrained scheduling problem* (UCS) [10]. The difference between the ASAP time and ALAP time of the operation defines the *mobility* range of that operation. Mobility range of the operation is very useful and commonly used to aid many other scheduling algorithms such as list-based scheduling and force-directed scheduling.

The other type of scheduling algorithms is force-directed scheduling (FDS)[7, 8]. It was developed as a part of Carleton University's HAL system. It is a popular constructive algorithm commonly used to solve the *time-constrained scheduling* (TCS) problem. The main policy of this algorithm is to distribute the execution of similar operations in different control steps in order to achieve a high utilization of functional units, which in turn, reduces the number of functional units composing the final datapath. This goal is reached by balancing the concurrency of the operations among the time frame specified to complete the desired design. This balancing is done in three steps: determination of the time frame of each operation (i.e. mobility) using ASAP/ALAP algorithms mentioned above, creation of a distribution graph for each operation, and calculating the *force* associated with each assignment [8]. A force is a value associated with each operation in the DFG. Precisely, the force associated with the tentative assignment of an operation to c-step j is equal to the difference between the distribution value in that c-step and the average of the distribution values for the c-steps bounded by the operation's initial time frame [7]. The algorithm is iterative, and uses a global selection scheme to choose the proper operation to be scheduled next. This scheme makes the algorithm computationally expensive compared to list-based scheduling or to ASAP/ALAP techniques. Another modification of FDS is a force-directed list (FDLS) scheduling [7], which is used to solve the *resource-constrained scheduling* (RCS) problem as a standard list-based scheduling algorithm, but by using *force* as a priority function.

List-based scheduling techniques [3] are adopted to HLS systems to solve resource-constrained scheduling (RCS) problem. In this kind of scheduling problem, we are restricting the number of functional units of each type. List scheduling processes each control step sequentially, and tries at each control step to choose the best operation from all candidate operations to place into the current control step, subject to resources constraints. List scheduling uses a *ready-list*, which keeps all nodes that have all predecessors already scheduled and the ready-list is always sorted with respect to a *priority function*. The priority function always resolves the resource contention among operations, i.e. operations with lower priority will be deferred to the next or later control steps. Mobility of the operation is commonly used as a priority function in many HLS systems. The quality of the results produced by a list-based scheduler depends predominantly on its priority function. The time and space complexity for this approach is slightly more than ASAP/ALAP techniques because several lists have to be maintained dynamically.

Static list scheduling [3] is another approach to list-based scheduling techniques. In this algorithm, we are using only one single large sorted static list prepared before the start of scheduling. Then the operations are scheduled sequentially starting with the operation that has highest priority value and ending with lowest one.

Path-based scheduling algorithm is another important algorithm developed by Camposano and implemented in IBM Yorktown Silicon Compiler [9]. The basic strategy of the algorithm is to minimize the number of control steps needed to execute the critical paths that exist in the given CDFG. To do that, the algorithm gives emphasis to conditional branching i.e. it starts by extracting all possible execution paths from the given CDFG and schedules them independently. Then the schedules for the different paths are combined to generate the final schedule for the whole design [9]. However, the path-based approach restricts the execution order of the operations before scheduling.

Integer linear programming (ILP) is another kind of scheduling algorithms that is used to solve both resource-constrained scheduling problem and time-constrained scheduling problem. ILP is an exact algorithm based on solving an ILP formulation that guarantees to find the globally optimal schedule, but at the cost of more processing time. This method produces an optimal schedule for all operations simultaneously. An example of ILP approach is the ALPS system [1]. Though ILP approach is able to find the optimal solution, it is not adequate for large examples because ILP is an exponential time algorithm by nature.

3. Problem description

Most existing scheduling methods such as FDS [8], FDLS [7], FAMOS [4], SALSA [6], and SEHWA [5] use

various heuristics to find good designs quickly. Although they find optimal solutions for some benchmarks, they may generate sub-optimal results for others such as those illustrated in Figure 1 [11] and Figure 2. On the other hand, even though ILP based scheduling techniques guarantee to produce an optimal schedule, but they are exponential time techniques, consequently they are impractical for large problems. Figures 1 and 2, describe some cases where both list-based scheduling (with a mobility alone as a priority function) and FDS algorithms are unable to find the optimal schedule. These deficiencies in several existing scheduling algorithms have led us to examine the underlying methodology used to select the candidate operation to be scheduled next, in such well-structured DFGs, and come up with a new approach that is simple and yet effective.

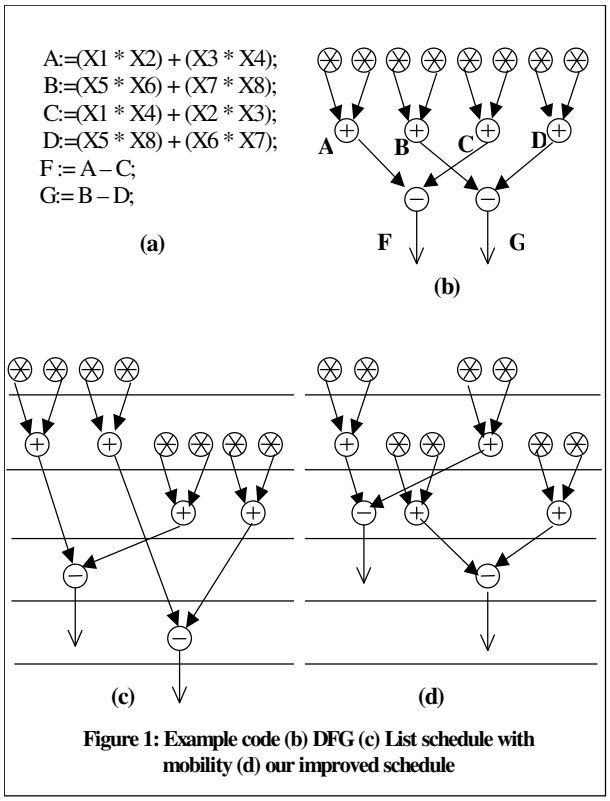

Figure 1: Example code (b) DFG (c) List schedule with mobility (d) our improved schedule

The scheduling method presented in this paper emphasizes a novel list-based scheduling algorithm. The proposed algorithm aims to determine a schedule with near-minimal number of c-steps, given fixed hardware constraints. The algorithm exploits some inherent features of data-driven digital systems (i.e. signal and image processing systems). Since this kind of algorithms, enclose in their DFGs the features of regularity and symmetry, such as a butterfly computational structure, which is found in FFT and DCT algorithms, as well as those regular structures that are essential computational cores found in wide variety of filtering algorithms. The regularity can guide the scheduler to find very efficient schedule as seen in Figures 1 and 2, while the symmetry of operations can lead to very efficient module selection algorithm.

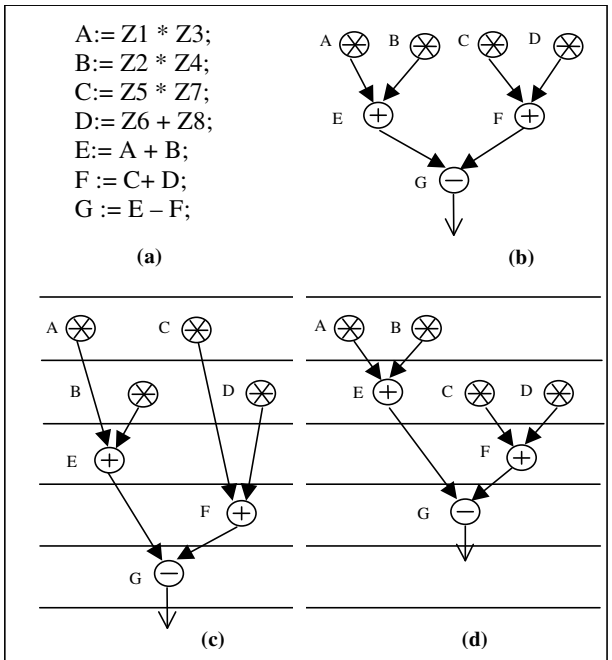

Figure 2: (a) example code (b) DFG (c) list schedule with mobility (d) our improved schedule

The algorithm gathers all the useful information about the structure of the design behavior during the preprocessing phase before the start of the main scheduling process. Then, during the execution of the main scheduling procedure the algorithm efficiently uses that information to reorder those operations found in the *ready-list* with an equal main priority value in such a way that it guides the scheduler to correctly select the proper operation for scheduling in the current c-step. A tree-like structure is constructed in the preprocessing phase in such a way that the underlying DFG structure is well captured and provided to the algorithm in order to exploit the available resources efficiently, as well as to guide the scheduler to produce well-organized schedule in efficient computational time.

4. The proposed scheduling algorithm

Recall that the list-based scheduling algorithm maintains a priority list(s) of operations in a *ready-list* at each time step to choose from it, which operations are to be scheduled in the current c-step until the resources become unavailable. In view of the fact that, list-based

scheduling algorithms depend predominantly on its priority function, where in some cases, especially in DSP like algorithms, produce an equal priority values to some of nodes in the *ready-list*. These equal priority values complicate the scheduler selection process in such a way that it does not guide the scheduler to efficiently select the proper operation to be scheduled first in the current c-step. Such incorrect node ordering forces the scheduler to make decision errors which will be translated to sub-optimal schedule, i.e. long c-steps in our case of resource-constraint scheduling problem.

The proposed scheduling algorithm starts by preprocessing phase in which it reads the hardware description language (HDL) description code (e.g. VHDL) and then constructs the corresponding DFG structure. The data structure used in this phase makes use and stores all valuable data about the given design behavior. This information is included in every node, to help the scheduler to know more information about the node during the selection process (*such as successor, predecessor, number of successors, tree-id(s), depth of the tree, other nodes contributing to the same successor*). Therefore, after construction of DFG data structure every node knows its successor and its predecessor. In addition, a simple function to report operation-similarity of those nodes that contribute to the same successor is included in this preprocessing phase. Then a special function named *constructTree()* will choose those nodes that have no successors (last operations in the DFG) to construct *trees* starting from them as roots. The tree is constructed in such a simple way that each node starting from the last operation (i.e. selected as a root) in the DFG will pass a *tree-id* to its predecessor and the distance (*tree-depth*) is accumulated with every node until we will reach an input operation. This tree-like graph contains all nodes reachable from that root, i.e. some nodes may be included in more than one tree which allow them to be considered as a *critical nodes* or to be given the *tree-id* of the largest tree since we are accumulating distances from each tree root.

Actually, after creation of trees in the given DFG, every operation in the DFG will have one or more *tree-id(s)* and knows its successor and knows all the similar operations which contribute to the same immediate successor if any exists (*this is useful for efficient module selection process).* As seen in Figure 4, the algorithm constructs all the described data structure in the preprocessing phase only once before the starting of the main scheduling procedure. By using this approach, the main algorithm complexity is kept the same as the complexity of standard list-based scheduling algorithm, i.e. O(*cNlogN*) [15], where *N* is the number of operations in the DFG and *c* is the number of control steps.

The scheduler will use other priority function (e.g. mobility in our case) as main priority function to generate priority values to all operations in the *ready-list.* Then *ready-list* will be sorted as follows: for those operations that have equal priority value (the same mobility) the scheduler will select those operations that belong to the same tree, i.e. those contributing to the same path, using *tree-id* value enclosed with each node data structure. Then for those operations that have the same *tree-id* value, choose those operations contributing to the same successor (i.e. *subtree*) (*same-successor).* This simple technique was able to guide the scheduler to select the proper operation by providing the scheduler the correct information that is needed to make the correct choice.

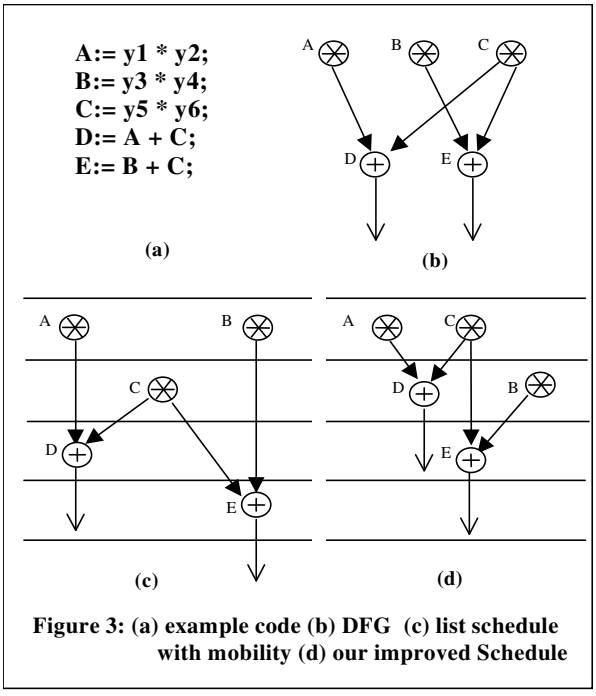

Figure 3: (a) example code (b) DFG (c) list schedule with mobility (d) our improved Schedule

Govindarajan and Vemuri in [11] proposed an algorithm based on graph clustering techniques, in which the graph is reconstructed in cones and clusters, the cones are made once before scheduling while clusters are made dynamically in each time step of the scheduling process. By using that heuristic, they attempted to schedule full clusters at every time step, when there is a conflict among equal priority operations [11]. While our algorithm can easily produce optimal schedule for the DFG given in Figure 3, their algorithm cannot produce the similar schedule unless it will create cones from intermediate nodes, which will increase more overhead to the algorithm. In contrast with their described heuristic, our approach based on DFG analysis that is totally done as preparation phase during the translating of behavioral description to the corresponding DFG structure phase before starting the scheduling process, we are achieving the same goal without any DFG dynamic construction

overhead. Hence, unlike their method, our heuristic produces correct schedule more quickly and efficiently.

```
NewListBasedSchedule()
Phase I
CreateFormDFG();
-- Construct DFG from the VHDL behavior
FindRelations();
-- Find all predecessors of a node
GetNumofSucc();
-- Count and returns no. of successors to each node;
ConstructTree();
-- Construct trees from nodes that have no successors,
 --Assign tree attributes to nodes,
 --Find also depth of each tree;
Phase II
BEGIN
  Find Mobility using ASAP/ALAP;
  PreparePriorityFunction();
  Prepare Resources List(s);
  C-step=0;
    WHILE DFG ≠ ϕ  do
      Insert_Ready_Operations to ready-list;
      Use main priority function to sort operations in ready-list
            WHILE Resources are sufficient DO
              C-step=C-step+1;
              sort the operations in ready-list as following
                IF mobility is the same THEN
                    Select those operations belong to
                     the same tree-id;
                     IF tree-id is the same THEN
                        Select those operations belong to
                        a common successor;
                    schedule selected operations in  current
                    C-step;
                   END IF;
                END IF;
                Defer other operations;
          END WHILE;
   End WHILE;
END NewListBasedSchedule;
```

Figure 4: Overview of the proposed scheduling algorithm

5. Extensions

Though the algorithm proposed by Govindarajan and Vemuri in [11] does not support pipelining, our algorithm supports variable execution time of functional units (multicycling) and structural pipelining. In multicycling the execution time of the operation takes more than one c-step to execute. The multicycled functional unit is considered empty after the last execution cycle. The pipelined functional unit is considered empty after the first stage execution cycle, thus obtaining temporal parallelism, e.g. a two-stage pipelined multiplier. In this case, the operation instances are executed in an overlapping manner [8]. A simple extension to the proposed algorithm is based on the fact that once the first stage of the pipelined functional unit is empty, the functional unit is considered available for another processing, while preserving data dependence relations with other operations.

6. Experimental results

We have implemented the algorithm using "C" language and we have compared our results with those reported in FDS [8], FAMOS [4], SALSA [6], and ALPS [1]. Experiments of FDS, FAMOS, SALSA, and ALPS are performed on Xerox 1108 lisp machine, SUN 4-280, SUN IPC, VAX 11-8800 respectively, while our experiments were done on Pentium III 700 MHz machine 128MB RAM. However, we have carried out several experiments that indicate the quality and effectiveness of the algorithm. The following experiments are conducted:

Experiment 1 is conducted on the *second-order differential equation* [8] benchmark to demonstrate the capability of the algorithm handling of general DFGs structures. For this experiment, it is assumed that the delay of a multiplier is 2 c-steps and the delay of an adder is 1 c-step. Results are summarized in Table 1, which shows that the proposed algorithm produces the same optimal results that have been reported with other scheduling techniques.

Experiment 2 is carried out with *fifth-order elliptic wave filter* [12] benchmark to demonstrate the quality of the algorithm. For this experiment, it is assumed that the delay of a multiplier is 2 c-steps and the delay of an adder is 1 c-step. Results are presented in Table 2, which clearly demonstrate that the proposed algorithm is able to produce similar results as those reported by FDS [7] and ALPS [1]. However, the proposed schedule is less in computational complexity than both FDS and ALPS. In addition, this experiment shows that the proposed algorithm produces the same results of using *mobility* alone as a priority function, in the worst case.

Experiment 3 is carried out with the DFG given in Figure 1 (above) to demonstrate the effectiveness of the algorithm in such class of DFGs. We derive the experimental results of this example assuming the delay of one c-step for a multiplier, an adder and the subtractor, for the simplicity purposes only, but our algorithm is able to handle multipliers with 2 c-steps as well. Results reported in Table 3 clearly illustrate the effectiveness of the new proposed graph analysis based selection scheme, which produces optimal results and inexpensive from execution time point of view. For instance, for the given DFG in Figure 1, FDS algorithm produces a schedule with 4[*], 2[+], 2[-] when we have 4 c-steps schedule length, while our approach produces a schedule with one less subtractor. That was because of the nature of the selection scheme used with FDS, which is based on distribution

graph, and doesn't exploit any information from the underlying DFG structure.

Experiment 4 is conducted with *fifth-order elliptic wave filter* [12] benchmark to demonstrate the quality of the algorithm while supporting the structural pipelining. For this experiment, we have used 2-stage pipelined multiplier. Results are presented in Table 4, which clearly demonstrates that the proposed algorithm effectively supports the structural pipelining concept and able to produce optimal results that are similar to FDS and ALPS results.

Experiment 5 is performed with *discrete cosine transform algorithm* (DCT) [13] benchmark to show the ability of the proposed algorithm to handle large DFG. Table 5 and Table 6 summarize the example results. Table 6 demonstrates the results using structural pipelining, while Table 5 reports results assuming the delay of a multiplier as 2 c-steps. For instance, for the DCT benchmark with a resource set as 2 ALUs and 3 multipliers, our algorithm finds a schedule with 16 c-steps length, while PSGA-Synth system [16] produces 18 c-steps of schedule length.

Experiment 6 is performed to demonstrate the capability of the algorithm to detect the similarity and the ability to produce well-organized schedules. Sub-graphs such as those found in DCT benchmark and *AR filter* benchmark [17] are used to run the experiment. Figures 1, 2, and 3 illustrate such results.

These experiments allow us to claim that the new proposed list-based scheduling algorithm always produces results better than the other scheduling algorithms in case of DFGs such as those given in figures 1 and 2. However, at the worst case, it produces results similar to those reported by the well-know scheduling algorithms as seen in tables 1, 2, 4, 5, and 6.

Table1
2nd order differential equation

System	No of C-steps	Multiplier	Adder
FDS [8]	6	3	2
	7	2	2
FAMOS [4]	6	3	2
	7	2	2
ALPS [1]	6	3	2
	7	2	2
List-based mobility only	6	3	2
	7	2	2
List-based New approach	6	3	2
	7	2	2

Table 2
Fifth order elliptic wave filter without FU pipelining

System	Csteps	Mult.	Adder	CPU time
FDS [8]	17	3	3	1 min
	18	2	3	3 min
	19	2	2	7 min
	21	1	2	13 min
FAMOS [4]	17	3	3	0.067 sec
	18	2	2	0.101 sec
	21	1	2	0.783 sec
SALSA [6]	17	3	3	13 sec
	18	2	3	55 sec
	21	1	2	53 sec
PSGA [16]	17	3	3	10 sec
	18	2	2	10.2 sec
	19	2	2	10.2 sec
	21	1	2	10.3 sec
ALPS [1]	17	3	3	0.26 sec
	18	2	2	3.1 sec
	19	2	2	10.5 sec
	21	1	2	34.5 sec
List-based (mobility)	18	3	3	0.1 sec
	18	2	3	0.1 sec
	19	2	2	0.1 sec
	21	1	2	0.1 sec
List-based New proposed Approach	18	3	3	0.1 sec
	18	2	3	0.1 sec
	19	2	2	0.1 sec
	21	1	2	0.1 sec

Table 3
Results of the DFG illustrated in figure 2

Res. Set	No. of C-steps			
+ - *	List-based mobility	List-based mobility+ no of successors	List-based proposed approach	Optimal
1 1 1	10	10	10	10
1 1 2	6	6	6	6
2 1 3	5	5	5	5
2 1 4	5	5	4	4
2 1 5	5	5	4	4
2 1 8	5	5	4	4
4 2 8	3	3	3	3

Table 4
Fifth order elliptic wave filter with FU pipelining

System	Csteps	2-stage pipe. mult.	Adder
FDS [8]	19	1	2
	18	1	3
	17	2	3
FAMOS [4]	19	1	2
	18	1	3
	17	2	3
SALSA [6]	19	1	2
	18	1	3
	17	2	3
ALPS [1]	19	1	2
	18	1	3
	17	2	3
List-based (mobility)	19	1	2
	18	1	3
	17	2	3
List-based New proposed Approach	19	1	2
	18	1	3
	17	2	3

Table 5
DCT without FU pipelining

Resources		No. of c-steps	
ALU (+, -)	Mult.	SALSA [6]	List-based New approach
4	4	10	11
3	3	14	14
2	3	19	16
3	2	18	19
2	2	19	18
2	1	34	34
1	1	35	35

Table 6
DCT with FU pipelining

Resources		No. of c-steps	
ALU (+, -)	2-stage pipe. Mult.	SALSA [6]	List-based new approach
4	4	10	9
4	3	10	10
4	2	11	12
3	2	13	13
3	1	19	19
2	1	20	19

7. Enhancements to the design space exploration

This algorithm was developed and tested with the other scheduling algorithms. This set includes ASAP/ALAP, force-directed scheduling (with only critical path length), static-list scheduling algorithm, and list-based scheduling algorithm. The list-based scheduling algorithm was created with different priority functions as well as with extensions of functional unit pipelining and multicycling. The priority functions that are associated with the list-based scheduler include (*Mobility* alone, *number-of-successors* alone, *Mobility+number-of-successors*, *mobility+tree-structuring,* and *mobility+treeid+ same_successor*). The priority function *number-of-successors* represents all the node successors not only those successors that belong to the current c-step (i.e. it does not represent the immediate successors, but it represents critical path). Using this set of algorithms, we have explored the design space of many benchmark algorithms [14].

In [14], we have proposed an efficient design space exploration methodology that includes scheduling, module selection and clock cycle determination schemes. The method starts by exploring the scheduling problem by using the set of algorithms described in this paper. Then, any selected solution from that step is considered one point in the design space that needs to be further explored thoroughly during the module selection step. Next, during the module selection phase, the methodology systematically explores several combinations of hardware resource configurations (*modules configuration set*) that are satisfying design specified timing constraints and report the optimal set with a minimum design area to the designer [18]. Finally, a clock cycle exploration step is guided automatically by the delays of modules composing the produced configuration set. This proposed design space exploration methodology helps designers to explore and hence to produce efficient designs in a reasonable time; more results are given in [14, 18]. However, we can say that the proposed algorithm enhanced the design space exploration process significantly, since it is able to produce optimal schedules for a set of DFGs as those illustrated in the figures 1, 2, and 3.

8. Conclusions

We have presented a scheduling algorithm aimed for a specific class of VLSI systems. The underlying methodology for the proposed schedule exploits some inherent properties of the behavioral flow graphs of DSP systems. However, it also allows us to incorporate some information extracted from DFG structure to guide the scheduler to find near-optimal/optimal schedules quickly.

The presented method also seems to be helpful at the module selection phase during HLS process in such a way that it allows assigning the same instance of a module to those operations that belong to the same successor (regularity feature), and they are from the same operation type (symmetry feature). We plan to use such features in a module selection algorithm. Nevertheless, the schedules produced by the proposed algorithm are always well-structured schedules in such a way that all operations that contribute to the same successor are scheduled as close together as possible respecting the availability of resources. However, the algorithm uses more simple techniques than the algorithm presented in [11] to achieve the same goals.

Finally, the results presented in this paper are encouraging; they demonstrate how application specific synthesis can benefit from exploiting the underlying structure of the DFG being synthesized which help schedulers to get accurate and optimal/near-optimal scheduling results. We propose to continue this research by incorporating realistic features such as functional pipelining to the described algorithm.

9. Acknowledgements

This research was performed with the grant agency of the Czech Republic under no. 102/01/1531 Formal approach to digital circuit diagnostic - testable design verification, and the research intent no. CEZ. J22/98: 262200012 – Research in information and control systems.

References

[1] C.-T. Hwang, J.-H. Lee, Y.-C. Hsu, "A Formal Approach to the Scheduling Problem in High Level Synthesis", *IEEE Transactions on Computer-Aided Design*, Vol. 10, No. 4, April 1991, pp. 464-475.

[2] C. A. Mandal, P. P. Chakrabarti, S. Ghoste, " Complexity of Scheduling in High-Level Synthesis", *VLSI systems Design*, Vol. 7, No. 4, 1998, pp. 337-346.

[3] D. Gajski, N. Dutt, A. We, S. Lin, "*High-Level Synthesis Introduction to Chip and System Design*", Kluwer Academic Publishers, The Netherlands, 1994.

[4] In-Cheol Park, Chong-Min Kyung, "FAMOS: An Efficient Scheduling Algorithm for High-Level Synthesis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 10, October 1993, pp. 1437-1448.

[5] N. Park, A. C. Parker: "SEHWA, "A Software Package for Synthesis of Pipelines from Behavioral Specifications", *IEEE Transactions on Computer-Aided Design*, Vol. 7, No. 3, March 1988, pp. 356-370.

[6] J. A. Nestor, G. Krishnamoorthy: SALSA, "A New Approach to Scheduling with Timing Constraints", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 12, No. 8, August 1993, pp. 1107-1122.

[7] P. G. Paulin, J. Knight, "Algorithms for High-Level Synthesis", *IEEE Design & Test of Computers*, Vol. 6, No. 6, Dec. 1989, pp. 18-31.

[8] P. G. Paulin, J. P. Knight, "Force-Directed Scheduling for the Behavioral Synthesis of ASIC's", *IEEE Transactions on Computer-Aided Design*, Vol. 8, No. 6, June 1989, pp. 661-679.

[9] R. Camposano, "Path-Based Scheduling for Synthesis", *IEEE Transactions on Computer-Aided Design*, Vol. 10, No. 1, Jan. 1991, pp. 85-93.

[10] R. A. Walker, S. Chaudhuri, "Introduction to the scheduling problem", *IEEE Design & Test of Computers*, Vol. 12, No. 2, summer 1995, pp.60-69.

[11] S. Govindarajan, R. Vemuri, " Cone-Based Clustering Heuristic for List-Scheduling Algorithms", *Proceedings of European Design & Test Conference (ED&TC)*, Paris, France, IEEE Computer Society, March 1997, pp. 456-462.

[12] S. Y. Kung, H. J. Whitehouse, T. Kailath, "*VLSI and Modern Signal Processing*", Prentice-Hall, Inc., 1985.

[13] S. Dutta, W. Wolf, "A Circuit-Driven Design Methodology for Video Signal-Processing Datapath Elements", *IEEE Transactions on VLSI Systems*, Vol. 7, No. 2, June 1999, pp. 229-240.

[14] A. M. Sllame, V. Drabek, "Design Space Exploration Scheme for High-Level Synthesis Systems", *Proceedings of 36th International Conference Modelling and Simulation of Systems MOSIS'02*, Ostrava, Czech Republic, April 2002, pp. 305-312.

[15] P. Eles, K. Kuchcinski, Z. Peng, "*System Synthesis with VHDL"*, Kluwer Academic Publishers, The Netherlands, 1998.

[16] M. K. Dhodhi, F. H. Hielscher, R. H. Storer, J. Bhasker, "Datapath Synthesis Using a Problem-Space Genetic Algorithm", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 14, No. 8, August 1995, pp. 934-944.

[17] R. Jain, A. C. Parker, N. Park, "Predicting System-Level Area and Delay for Pipelined and Nonpipelined Designs", *IEEE Transactions on Computer-Aided Design*, Vol. 11, No. 8, August 1992, pp. 955-965.

[18] A. M. Sllame, L. Sekanina, "An Evolutionary-Based Algorithm to the Module Selection Process in High-Level Synthesis", *Proceedings of Mendel 2002 - 8th International Conference on Soft Computing*, Brno, Czech Republic, June 2002, pp. 87-92.