Optimizing Resource Utilization using Transformations

Miodrag Potkonjak Jan Rabaey
Department of EECS
University of California, Berkeley

ABSTRACT

The goal of the high level synthesis for real time applications is to minimize the implementation cost, while still satisfying all timing constraints. We present how a combination of two transformations, being retiming and associativity, can help to further this goal. Since the minimization problem, associated with those transformations, is NP complete, a new fast, globally optimal iterative improvement probabilistic algorithm has been developed. The effectiveness of the proposed algorithms and the transformations will be demonstrated using standard benchmark examples, with the aid of statistical analysis and through a comparison with estimated minimal bounds.

1 INTRODUCTION

The goal of the high level synthesis for application specific integrated circuits is to translate the specification of the algorithm (defined in terms of its behavioral semantics as well as its performance requirements) into architectural primitives (being an interconnection of execution units, memory and control units) in such a way that the resulting silicon implementation minimizes a certain function. Most often this function is either the area and/or the power consumption.

Unavoidable high level synthesis tasks comprise module and clock selection, scheduling, assignment and allocation. However, even when the highest quality algorithms are used for those tasks, the quality of a final result is often constrained by the computational structure of the specified algorithm.

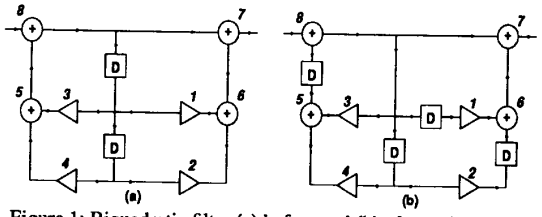

Figure 1: Biquadratic filter (a) before and (b) after retiming for resource utilization

This is illustrated using the example of Figure 1a.This Figure shows a direct form II biquadratic section often used in the realization of IIR filters. Assume that at most 4 clock cycles are available for the execution of this flow graph and that both multiplication and addition take a single clock cycle. The critical path of this computational graph equals four clock cycles as well. A potential schedule for this filter, requiring a minimal amount of hardware (for the sake of simplicity we will address only execution units here), is shown in Table 1. Regardless of the scheduling technique used, we need at least 2 multipliers, since the graph contains 4 multiplications and no multiplications can be scheduled in the last control cycle.

Also, since no addition can be executed in the first control cycle, 2 adders are needed. If we define the **resource utilization** as the ratio of the number of cycles a resource is exploited over the total number of available cycles, then the resource utilization for adders and multipliers in this example is 50%. This is an indication of a relatively low quality solution, in this case caused by the inherent structure of the computational graph.

Transformations are the convenient way to defeat these resource utilization bounds. Two particularly effective transformations to achieve this goal are retiming and associativity. **Retiming** uses the distributivity property of the delay operator over most other operators. In other words, when D is defined as the delay operator, the statement D(a) * D(b) is equivalent to D(a * b) and vice versa (with * an arbitrary operator).

	BEFORE		AFTER	
CYCLES	Multipliers	Adders	Multipliers	Adders
1	3,4	-	1	8
2	1,2	5	3	6
3	-	6	4	7
4	-	7,8	2	5

Table 1: Possible biquadratic filter schedule before (a) and after (b) retiming

Consider now the flow graph, shown in Figure 1b, obtained by moving the delays (retiming) in the original flow graph of Figure 1a. It is to check that the resulting graph has the same input/output relationship as the graph of Figure 1a.The resource utilization is far more balanced and a solution with one multiplier and one adder can now be achieved as shown in Table 1. The resource utilization for the execution units now equals 100%.

The above example illustrate that while transformations are not necessary for a bare minimum high level synthesis system, they are essential when trying to achieve a high quality solution (or even just a feasible solution).

While the basic synthesis operations, especially scheduling [11], have been the subject of extensive research efforts, transformations have received significantly less attention. Most synthesis systems apply only the basic software transformations, such as dead code elimination, manifest expression removal and common sub-expression elimination [18, 20]. Also pipelining is very often applied [14]. Although pipelining is very powerful, it is not a transformation in the strong senses, since it increases algorithm latency. It application domain is also limited to non-recursive algorithms [12].

Retiming has been successfully applied in several areas of Design Synthesis. Until recently, it has been exclusively used either to reduce the critical path in a circuit or graph [9], to minimize the number of delays in the graph [9, 3] or to optimize sequential networks using combinational logic tools by temporary moving delays to a periphery of a network [10]. The retiming for resource utilization transformation was introduced by the authors [15].

Associativity is most often used in conjunction with distributivity for a reduction of the critical path. Valiant [19] and Miller [13] presented optimal polynomial time complexity algorithms for critical path minimization. In high level synthesis, it has been used for the optimization during pipelining [e.g. 5].

2 PROBLEM FORUMLATION

The resource utilization U_r for a resource r can be defined as the ratio of the number of control steps in which the resource is used over the total number of control steps available. The total resource utilization U_{tot} of a design can then be defined using the formula:

$$U_{tot} = \sum_{r \in R} w_r U_r$$

R is the set of hardware resources used. The weights w_r are proportional to the hardware cost of the resource. The cost of a design is inversely proportional to the resource utilization. Achieving a high resource utilization is in general equivalent to achieving a small design cost. During the design optimization process however, it is easier to measure (or estimate) the resource utilization than the actual hardware cost.

The problem discussed in this paper can now be defined as a constraint satisfaction problem

***Given**: A data control flow graph (DCFG) and a proposed hardware architecture.*
***Goal:** Apply retiming and associativity in a such a way that the resource utilization of the resulting data control flow graph is maximized.*

The solution to the above problem can be used as a subroutine to address both the hardware minimization problem formulation (given the time constraints) as well as the time minimization formulation (given the available hardware).

3 OBJECTIVE FUNCTION

Our goal is to apply retiming and associativity to achieve a high resource utilization (measured over all resources, being execution units, memory and interconnect). A good objective function should therefore be highly correlated to the final (unknown) hardware utilization. The objective function also should be easy to compute, since, as shown in [15], it is used in the optimization of an NP-complete problem, and it has to be evaluated many times. A simple yet effective objective function can be constructed, based on the following observations:

(1) It is easier to achieve a high resource utilization when the timing constraints on DCFG nodes are not strict;

(2) It is advantageous to distribute DCFG nodes vying for the same resource (which might be, for example, adder or a particular register file) over the time span;

(3) The critical path should be shorter than the available time;

(4) The number of variables which are alive at the same time, should be smaller than the number of available registers.

Those observations can be quantized in the following way: Any operation A has to be scheduled in the interval between its As Soon As Possible ($ASAP_A$) and its As Late As Possible ($ALAP_A$) times. Those times can be easily computed, using leveling according to the input and to the output. The slack of a node is defined as the difference between the ALAP and ASAP times, incremented by 1. A DCFG with a lot of operations with a small slack represents a highly constrained scheduling problem, which often results in a poor resource utilization. However, even when the average slack is high, scheduling can still be difficult to achieve if a number of DCFG nodes with a very small slack are present. Therefore, in order to properly determine the expected difficulty and the number of constraints during scheduling we define for each DCFG node property a measure, called the expected scheduling difficulty (SD). SD is defined as the inverse of the slack. The total scheduling difficulty of a DCFG, composed of the node-set S, is defined as the sum of the scheduling difficulties over all DCFG nodes:

$$TSD = \sum_{A \in S} \frac{1}{ALAP_A - (ASAP_A - 1)}$$

At the same time, it is important, that operations which can compete for the same resource (same type of execution unit, same interconnect, or registers in the same register files) are not happening simultaneously. The probability, that two operations A and B which require the same type of resource will happen simultaneously and therefore during assignment will require another instance of this type of resource, is proportional to the overlap of their ASAP-ALAP intervals (OA_B). This probability can therefore be approximated using the following formula: $OL_{AB} = O_{AB} / (SL_A \times SL_B \times IR_{AB})$ where SL_A is the slack of operation A, SL_B is the slack of operation B, and IR_B is number of resources of this class. A total overlap measure, called TOL, is defined over the set S of all nodes of the graph:

$$TOL = \sum_{A \in S, B \in S, A \neq B} OL_{AB}$$

The retiming and associativity transformations may also change the critical path of the graph. Obviously, no feasible schedule exists if the critical path is longer than the available time. Furthermore, retiming changes the number of delays in the graph. All variables which are associated with delays must be stored simultaneously (during the first cycle) in registers. No feasible schedule is available if the number of delays exceeds the max bound on the number of registers. Both the critical path and the number of delays, are incorporated in the objective function, such that it becomes infinity when one of those constraints is violated. Our experience furthermore indicates that a correlation exists between the number of delays and the number of registers required. Therefore, it is useful to add the number of delays (ND) to the objective function as a measure of the total register cost. All those factors can now be combined into the global objective function (OF):

$$OF = \begin{cases} \infty, \text{ if } t_{crit} > available\ \ time \\ \text{or } ND > number\ \ of\ \ registers \\ \alpha_1 \times TSD + \alpha_2 \times TOL + \alpha_3 \times ND, \quad \text{otherwise} \end{cases}$$

Weight factors can be explicitly set by the user. For example, a relatively large α_3 often results in fewer registers, but more interconnect and execution units. For all examples discussed in the experimental section, we used the following default settings: $\alpha_1 = 0.8, \alpha_2 = 0.1, \alpha_3 = 0.1$

The close correlation between our objective function and the quality of the obtained solution is depicted in Figure 2 for the example of an 11th order FIR filter. The x-axis shows the value of the objective function, while the y-axis respectively contains the corresponding number of control steps, necessary to execute the graph on a fixed amount of hardware (as determined by the scheduling process) (Fig. 2a) or the amount of hardware needed to execute the graph in a fixed amount of time (here 13 cycles) (Fig. 2b). It is easy to observe that a small value of the objective function invariantly predicts a high quality solution.

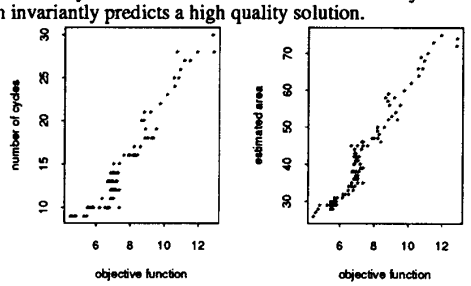

Figure 2: Correlation between Objective Function and Solution Quality

4 LEARNING WHILE SEARCHING ALGORITHM

While the traditional retiming problems (for critical path and minimum number of delays) have a polynomial complexity, we have proven that the retiming for resource utilization algorithm is an

NP-complete problem [15]. It is therefore very unlikely that a worst case polynomial complexity algorithm exists.

In order to efficiently solve the problem, a new probabilistic iterative improvement algorithm, **learning while searching**, has been developed.

Two classes of basic moves can be defined, being retiming and associativity moves. The retiming move was discussed in the first section.

In order to enhance the power and the application range of associativity, we have expanded its definition, such that it addresses several additional cases, not covered by the standard definition. The generic associativity move corresponds to entry 1 in Table 2. The basic moves have been extended with 6 additional transformations, as shown in Table 2. Cases 3 and 4 are especially interesting, since they allow to trade off between respectively the number of additions and subtractions and the number of multiplications and divisions in the DCFG.

entry	initial	transformed	initial	transformed
1	a+(b+c)	(a+b)+c	a*(b*c)	(a*b)*c
2	a+(b-c)	(a+b)-c	a*(b/c)	(a*b)/c
3	a-(b+c)	(a-b)-c	a/(b*c)	(a/b)/c
4	a-(b-c)	(a-b)+c	a/(b/c)	(a/b)*c

Table 2: Basic Associativity Moves

When applying the above transformations on a typical example, a large number of moves are possible at every point in time. As even more basic moves might be introduced in the future, time-efficient algorithms are definitely necessary. We first tried the popular Simulated Annealing [7] technique. Although the results were satisfactory, the run times were excessive for large examples, even when adaptive cooling [1] and a rejectionless approach [15] were applied. Therefore, we decided to use a new and faster probabilistic approach.

The new technique is organized as a two phase process. In the first phase, the solution space is scanned in an organized fashion to detect areas where the objective function has a small value. Those areas are used in the second phase as the starting points for a more elaborate search towards a final solution.

The goal of the first phase is thus to discover (learn) k solutions where the objective function has a small value. In order to achieve this goal, we will traverse the search space a number of times, each time favoring one particular direction of traversal. For instance, we will first (probabilistically) favor moves in the forward direction (moving the delays from the inputs to the outputs for the retiming and favoring the forward associativity moves). After 1/4 of the optimization process, the preferred direction is reversed: moving the delays from output to input as well as the reverse associativity moves are now probabilistically favored. Finally, for the last 1/4 of the time, the forward moves are favored anew.

At every point in the optimization process, we select a move in a probabilistic fashion, proportional to the improvement in the objective function, while also accounting for the favored direction at that time. No selected moves are ever rejected. Moves, which increase the objective function, can be selected, but the changes for this to happen are inversely proportional to that increase. To increase the speed of the design space search, we decided to evaluate the objective function only every four steps, resulting in a speed-up of approximately four times. This is acceptable, since neighboring solutions in the design space (solutions which can be reached in at most m moves) tend to display a strong correlation in their objective function values. Since the exact location of a local minimum is only determined in the second phase, no degradation of the solution quality was observed as a result of this simplification.
Table 3 shows the minimum, average, and maximum correlation among neighboring solutions for several examples.

m	1	2	3	4	5	6
min	0.971	0.932	0.919	0.902	0.866	0.808
average	0.978	0.952	0.927	0.909	0.878	0.841
max	0.994	0.978	0.959	0.932	0.907	0.879

Table 3: Correlation among solutions on distance of **m** moves

The first phase results in k starting points for the second phase. Those are used as the seeds for a greedy search towards the final solution. The objective function is now observed at each step and for all possible moves. The move offering the best decrease in the objective function is automatically selected. For each starting point, the search is concluded when a local minimum is reached. The best of those minima is selected as the final solution. We have set the number of starting points k to 10 for the examples, discussed in the next section. The length of the first phase was set such that the number of moves during the first forward traversal equaled 10 times the number of nodes in the graph. Moves in the forward direction were preferred with a ratio 4:3. We have varied these values of large ranges and did not notice any significant changes in the quality of the solution, although the effects on the run time were outspoken. The presented approach can easily be augmented with a cooling mechanism (phase 1) or backtracking.Experiments have shown however that those extensions do not produce any improvements and have a detrimental effect on the run time.

It is interesting to notice that the presented approach resembles, to some extent, the simulated annealing approach [7] as well as the Kernighan-Lin iterative improvement approach [6]. When only the second phase is applied, the approach is equivalent to Kernighan-Lin (which was almost uniquely used for partitioning problems until now). While phase 1 uses an iterative, probabilistic improvement technique, just as simulated annealing, some major differences with the annealing approach can be observed: First of all, the presented technique uses a directed search, while annealing executes random moves. The major difference however is the combination of probabilistic exploration and greedy solution generation.

5 EXPERIMENTAL RESULTS

The ultimate proof of the usefulness and effectiveness of a proposed transformation or optimization algorithm is to apply it on real life examples. We have applied our technique on 40 DCFGs, which include common DSP, communication and error-correcting code examples (such as FIR, IIR, adaptive and wave digital filters and simultaneous polynomial division and multiplication). For each of those examples, we varied the available time. We also varied the relative execution lengths of the operators (such as shifts, adds and multiplications).

The primary objective was to measure how much hardware can be saved if the transformation is applied. A secondary task was to evaluate the potential of the transformation to minimize the execution time of an algorithm without increasing the latency (in contrast to pipelining). While improvements in execution units and memory cost can be measured exactly, the cost of interconnect could only be estimated, since precise numbers are only available after time consuming tasks such as floorplanning and routing. We have compared the interconnect cost based on the number of busses and assuming that all busses have the same cost.The ratio of the implementation cost after the application of

the transformation over the cost of the initial implementation for benchmark examples are shown in Table 4.

Comparing the final solution with the initial DCFG, provided by the designer, has only a limited significance, since this highly depends on the amount of manual optimization, applied by the designer. We therefore compared the cost of the generated solutions against the cost of a 20 random solutions (generated by randomly applying retiming and associativity moves) as well. The average and median savings against the initial implementation and the random implementation are tabulated in Table 4.

	EXU		MEMORY		INTERCONNECT	
	Random	Initial	Random	Initial	Random	Initial
average	40.1	32.5	25.4	23.6	33.8	29.7
median	28.5	33.3	26.6	22.2	36.6	30.0

Table 4: Savings against the Initial and Random Implementations (in%)

The generation of the above results required the use of a particular set of scheduling, assignment and allocation tools. Since those problems are NP-complete as well, it might be argued that the obtained improvements were not the result of the transformation on itself, but are due to the fact that the heuristic scheduler performed better on the transformed graphs. This argument can be discarded using the following simple procedure. For each instance of a DCFG, it is possible to establish sharp minimum bounds on the resources, by using the facts that during some control steps no candidates are available for scheduling [17]. These bounds can not be outperformed, regardless of the used scheduling, If the application of the transformations results in a decrease of those bounds, then this improvement is a pure consequence of the transformations. The values of the median, average, maximum improvement for execution units area respectively equal 21.6%, 21.7% and 47.1%. Only once was the min bound not reached.

To evaluate the effects of the transformation on a well known benchmark, we have applied the technique on the popular 5th order elliptical wave digital filter example. The results are tabulated in Table5 for the available times ranging from 15 to 19 clock cycles (in correspondence with the standard benchmark, we assume that a multiplication and an addition take respectively 2 and 1 clock cycle). As can be observed from the table, the average hardware savings due to the transformations are impressive. Also interesting to notice is that the fastest solution after retiming for critical path still needs 16 clock cycles [Har89]. The combination of retiming and associativity succeeds in producing a solution which requires only 15 cycles.

	Number of Control Steps	15	16	17	18	19
BEFORE	# of adders / # of multipliers	NA	NA	3 / 3	3 / 2	2 / 2
AFTER	# of adders / # of multipliers	3 / 3	2 / 2	2 / 2	2 / 2	2 / 1

Table 5: Number of adders and multipliers used for Implementation of 5th Order Elliptical Filter before and after applying retiming and associativity

The effectiveness of the algorithm is illustrated by the fact that, even for graphs with several hundred nodes, the run time was shorter than one minute.

The proof that retiming for resource utilization is NP-complete problem, the rationale why it is necessary to combine associativity and retiming in order to better explore their potentials as well as the number of other interesting properties of proposed transformation are discussed in detail in [16].

6 CONCLUSIONS

A transformational approach, aimed at improving the resource utilization in high level synthesis, has been introduced. The current implementation combines retiming and associativity in a single framework. This combination of transformations results in considerable area improvements as is amply demonstrated by the benchmark examples. A novel **learning while searching** iterative improvement probabilistic algorithm has been developed and is used to resolve the associated NP-complete combinatorial optimization problem. The proposed algorithm has proven to be very effective both in reaching the optimal solution as well as in runtime.

REFERENCES

[1] F. Catthoor, et.al:"SAMURAI: a general and efficient simulated-annealing schedule with fully adaptive annealing parameters", Integration, Vol. 6, pp. 147-178, 1988.

[2] M.R. Garey, D.S. Johnson, "Computers and Intractability: A Guide to the Theory of NP-Completeness", W.H.Freeman, New York, 1979.

[3] G. Goossens, et. al., "An optimal and flexible delay management technique for VLSI" in: C.I. Byrnes, A. Lindquist, "Computation and Combinational methods in system theory", pp. 409-418, 1986.

[4] B.S. Haroun, M.I. Elmasry: "Architectural Synthesis for DSP Silicon Compilers", IEEE Trans. on CAD, Vol. 8, No. 4, pp. 431-447.

[5] R. Hartley, A. Casavant: "Tree-height Minimization in Pipelined Architectures", IEEE CAD, pp.112-115, 1989.

[6] B.W. Kernighan, S. Lin: "An efficient heuristic procedure for partitioning graphs", BSTJ, Vol. 49, pp. 291-307, 1970.

[7] S. Kirkpatrick, at. al: "Optimization by simulated annealing", Science, Vol. 220, No. 4598, pp. 671-680, 1983.

[8] M. Lam: "Software Pipelining: An Effective Scheduling Technique for VLIW Machines", ACM SIGPLAN, 1988.

[9] C.E. Leiserson, et.al., "Optimizing synchronous circuits by retiming", Proceedings of Third Conference on VLSI, pp. 23-36, 1983.

[10] S. Malik, et.al: "Retiming and Resynthesis: Optimizing Sequential Networks with Combinational Technique", IEEE Trans. on CAD, Vol. 10, No. 1, pp. 74-84, 1991.

[11] M.C.McFarland, et.al: "The High-Level Synthesis of Digital Systems"", Proceedings of the IEEE, Vol. 78, No. 2, pp. 301-317., 1990.

[12] D. Messerschmitt, "Breaking The Recursive Bottleneck", in *Performance Limits in Communication Theory and Practice*, 1988.

[13] G.L.Miller, et.al: "Efficient Parallel Evaluation of Straight-Line Code and Arithmetic Circuits," *SIAM Journal on Computing*, Vol. 17, No 4, pp. 687-695, 1988.

[14] N.Park, A.C.Parker: "Sehwa: A Software Package for Synthesis of Pipelines from Behavioral Specifications", IEEE Transaction on CAD for IC, Vol 7, No. 3, pp. 356-370, 1988.

[15] M. Potkonjak and J. Rabaey, "Retiming for Scheduling", *VLSI Signal Processing Workshop*, pp. 23-32, San Diego, Nov. 1990.

[16] M. Potkonjak, "High Level Synthesis: Resource Utilization Approach", *Ph.D. Thesis*, University of California, Berkeley, 1991.

[17] J. Rabaey, and M. Potkonjak, "Resource Driven Synthesis in the HYPER system," *ISCAS-90*, vol. 4, pp. 2592-2595, May 1990.

[18] Trickey, H.: "Flamel: A high-Level Hardware Compiler", IEEE Transaction on CAD, Vol. 6, No. 2, pp. 259-269, 1987.

[19] Valiant, et.al: "Past Parallel Computation of Polynomials Using Few Processes,", SIAM J. on Comp, Vol. 12, No 4, pp. 641-644, 1983.

[20] R.A.Walker, D.E.Thomas,: "Behavioral Transformation for Algorithmic Level IC Design" IEEE Trans on CAD, Vol 8. No.10, pp. 1115-1127, 1989.