Resource Driven Synthesis in the HYPER System

J. Rabaey and M. Potkonjak
University of California, Berkeley

ABSTRACT

A global optimization strategy for design synthesis is presented. This strategy, which drives the transformation, allocation, assignment and scheduling phases of the synthesis process uses a resource utilization table to estimate the quality of the solution and to select the next step to be taken.

The presented technique is implemented in the HYPER system, a synthesis environment targeted at high performance real time systems. The effectiveness of the techniques is demonstrated with a number of small examples.

INTRODUCTION

The computationally intensive parts of high performance real time systems, such as HDTV or speech recognition, are usually implemented on clusters of heavily pipelined data paths, controlled by a relatively simple finite state machine. The amount of resource sharing is limited. The HYPER system addresses the synthesis problem for this class of architectures, starting from an applicative, signal flow language SILAGE [Hil85] all the way down to the final layout.

The most important feature of HYPER, which distinguishes it from other synthesis systems, is a single global optimization strategy, which is used to drive not only the resource assignment and scheduling process but also the preceding optimizing transformations. In all those phases of the synthesis process, HYPER attempts to optimize a so called *resource utilization table*, which maps the utilization of the different resources (such as execution units, memory, interconnect and I/O bandwidth) onto the allotted time period. A resource in the table which is underutilized over periods of time is a good candidate for reduction through transformations.

SYSTEM OVERVIEW

The HYPER system [Chu89] consists of a library of software modules, operating on a centralized flowgraph database. The algorithm (described using SILAGE or entered in a schematic format) is first compiled into an intermediate *control data flowgraph* (CDFG), stored in the OCT database [Har86]. The CDFG represents the algorithm essentially as a data flowgraph, extended with some *macro control flow* operations such as loops and if-then-else blocks. The introduction of those control statements results in a hierarchical graph : the body of a loop or a conditional is represented by a sub-graph, which is contracted into a single node at the next hierarchy level.

During the synthesis process, a number of equivalence transformations are performed on the graph. Furthermore, structural information such as the hardware assignment and scheduling information is back-annotated onto the graph nodes. Once the synthesis process is completed, the graph is mapped into a hardware structure and silicon is generated using the LAGER-IV silicon compiler [Chu89].

This paper concentrates on the global optimization procedure, used in the different phases of the behavioral synthesis process, being the estimation, transformation, allocation, assignment and scheduling phases.

ESTIMATION

In the estimation phase, min and max bounds on the required resources are deduced. These bounds are important for several reasons : first of all, they delimit the design space, thus speeding up the search in the hardware allocation phase. Secondly, they serve as entries in the resource utilization table, which helps to guide the transformation, assignment and scheduling operations. In order to be useful, it is essential that these bounds are as accurate as possible. To obtain this goal, we have adopted a technique of gradual refinement. Let us consider first a flat graph with a max bound on the execution time t_{max}.

The estimation starts with a topological ordering and leveling of the graph with respect to the input nodes and the output nodes, yielding a minimum and maximum execution time for each operation Oi (t_{min}^{Oi} and t_{max}^{Oi}). The length of the critical path is also obtained.

An upper bound on each resource is easily obtained from the ordered graph by computing for each clock cycle the maximal possible usage of a that resource (in other words, the maximal parallelism available in the graph) and by maximizing this value over the complete time period. Notice that a resource could be an execution unit, a register, an interconnection between execution units or an input/output bus. For the sake of brevity, we will concentrate our description on execution units.

This procedure is demonstrated for the example of a 7th order biquadratic low pass filter, whose signal flow graph is given in Figure 1. The maximal number of add and shift operations, which can be performed in each cycle is plotted in Figure 2. It is assumed here that a maximum of 13 clock cycles is available and that each operation takes exactly one clock cycle. It is clear from this Figure that a maximum bound on the number of adders equals 9, while the max bound on shifters is 10.

CH2868-8/90/0000-2592$1.00 © 1990 IEEE

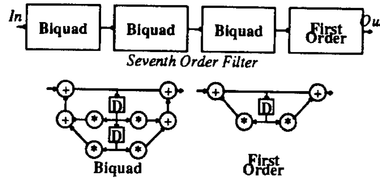

Figure 1 : Signal Flow Graph of 7th Order Biquadratic Filter

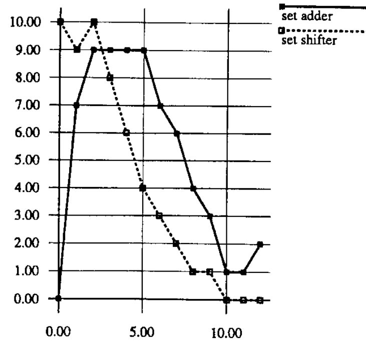

Figure 2 : Available Parallelism (add and shift) in 7th Order Filter (with t_{max}=13)

Deriving a precise lower bound is somewhat more complicated. A first, but crude, guess can be obtained by observing that, given a number of resources of class Ri (N_{Ri}), at most $N_{Ri} * t_{max} / L_{Ri}$ operations can be performed on those resources (with L_{Ri} the length of a single operation). The required number of operations (O_{Ri}) can be easily derived from the computation graph, resulting in the following lower bound on N_{Ri} :

$$N_{Ri} \geq \frac{O_{Ri} * L_{Ri}}{t_{max}} \qquad (1)$$

However, by inspecting the topologically ordered graph, it might be noticed that the actual resource availability will be smaller than $N_{Ri} * t_{max}$ (assuming here that L_{Ri}=1) , due to the fact during certain cycles the available parallelism in the algorithm will be smaller than N_{Ri}, leaving some units idle. This is demonstrated in Figure 2, where the available number of additions drops below 3 after cycle 11. This observation results in a more precise min bound :

$$O_{Ri} \leq t_{max} * N_{Ri} - Unused\ Resources \qquad (2)$$

Since the number of Unused Resources actually depends on N_{Ri}, (2) has to be solved iteratively, starting from the initial solution obtained from (1).

An even sharper lower bound can be obtained by performing an as soon as possible list schedule on the computation graph, considering only operations of class Ri and using the slack as the scheduling heuristic. Once again, this is an iterative procedure, starting from the min-bound obtained in (2) and increasing N_{Ri} till a non-feasible schedule is encountered.

The min and max bounds, obtained by the above algorithms for the example of Figure 1 are shown in Table 1. Note that the multiplications are performed as shift/add operations.

Resource	#Operations	MinBound	MaxBound
adder	28	3/3	9/11
shift	14	2/2	10/10
subtract	6	1/1	6/6
adder-adder	36	3/3	16/17
adder-shift	13	2/1	5/6
adder-subtract	2	1/1	1/1
shift-adder	10	2/2	8/8
shift-inverter	4	1/1	4/4
subtract-adder	6	1/1	6/6

Table 1: Min and Max Bounds on Execution Units and Interconnect for 7th Order Biquadratic Filter (Fig. 1). Available time is 13/15 cycles.

The situation becomes more complex when considering hierarchical graphs. Since the only time given is the total execution time for the complete graph, the time allotted to each subgraph is unknown and is a subject of optimization itself. The approach taken in HYPER is to first divide the total available time over the different sub-graphs proportional to the length of their critical paths. This division will be further refined in the Transformation Phase, described below. Once the execution times for the sub-graphs are defined, the estimation can be performed as described above. A global result can be obtained by combining the results of all subgraphs.

TRANSFORMATIONS

Due to the real time aspect of the systems, targeted by the HYPER system, the optimizing transformation phase is the crucial step in the synthesis process, resulting most of the time in a much larger payoff than subsequent steps such as hardware assignment or mapping. In general, we can divide the optimizing transformations in two classes, being performance optimizing and implementation optimizing transformations.

The performance transformations reshuffle the signal flowgraph such that the performance requirements (such as the maximal execution time) for the application can be met. It is well known from software compilers for parallel computers that the most efficient transformations in this class are the loop transformations. Real time systems, working on infinite streams of data, have the advantage that an infinite loop (over time) is always available. Typical (and most effective) transformations in this class are (loop) retiming [Lei83], software pipelining [Lam85], (partial) loop unrolling and scattered look-ahead [Mes88].

An example of how the inner loop of a dot-vector product computation (as described in equation (3)) can be optimized using first retiming followed by software pipelining is shown in Fig. 3.

$$for(i=1, \cdots, N) \quad sum[i] = sum[i-1] + a[i] * b[i] \qquad (3)$$

The **D** nodes in the graph stand for delay nodes. The retiming process moves the delay nodes around in the graph, such that the critical path is minimized. The software pipelining transformation transfers operations to other iterations of the loop, resulting once again in a shorter critical path. Notice that the pipelining is a special case of the retiming and can be performed using the same algorithm. The application of the above transformations on the dot-vector example reduce the critical path from 4 cycles/iteration to 1 cycle / iteration.

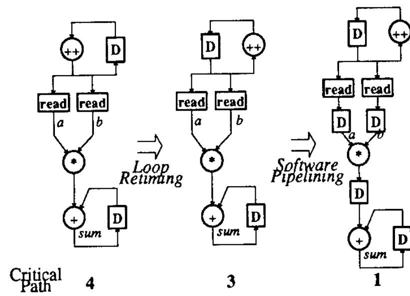

Figure 3 : Performance Optimizing Transformations

Implementation improving transformations change the flowgraph in such a way that the available resources can be exploited in a better way or that less resources are needed. For instance, an examination of Figure 2 reveals that the flowgraph of Fig. 1 is very inefficient when considering only additions : all the parallelism is available in the first cycles, so that it is highly probable that the available adder resources will be underexploited after cycle 9. This can be solved by applying a modified retiming algorithm, which moves delays around in the flowgraph such that the demand on resources becomes more equalized over time. Figure 4 shows how such a retiming (combined with some extra pipelining) improved the minimal and average available parallelism (with respect to add and shift operations) for the 7th order filter.

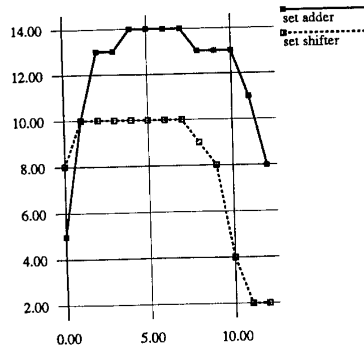

Figure 4: Available Parallelism (in adds and shifts) for 7th Order Filter After retiming (with t_{max}=13)

Other typical transformations at this level are algebraic transformations such as associativity, strength reduction, loop jamming, etc. Another transformation in this class simply redistributes the time allotted to the subgraphs, easing in this way the demand on a particular resource.

The problem when applying optimizing transformations is to determine which transformation to apply when and where. Most compilers and synthesis systems use a peephole optimization strategy, where the code is examined locally (a few lines at a time) for possible transformations. The disadvantage of this technique is the locality of the search and the lack of a global view.

DESIGN SPACE EXPLORATION AND RESOURCE ALLOCATION

The HYPER strategy is to use the Resource Utilization Table, obtained from the estimation phase as the global view on the quality of the current graph and as the guideline of where and what transformations to apply or how many resources to allocate. The table lists the bounds on the resources over time. Since the graph is normally hierarchical, the table is also constructed in a hierarchical fashion : a subgraph is represented at the next higher hierarchy level by its global min and max bounds. A sample table is shown in Table 2.

block	critical path	cycles	IO/cycle	*/cycle	+/cycle
graph1	c1	t1	1	0	1
graph2	c2	t2	0	.5	4
graph3	c3	t3	.3	2	.7
total	$c=\sum c_i$	$t=\sum t_i$	1	2	4

Table 2 : Sample Resource Utilization Table

The overall design space search and resource allocation process can now proceed as follows : First, it is checked if the graph can meet the performance requirements by analyzing the critical paths. If not, performance optimizing transformations are applied. Next, the resource allocation process is initiated. The task of this process is to come up with a minimal hardware configuration, which will meet the performance constraints.

An overall view of the resource allocation process is given in Figure 5. The search mechanism forms the core of the system. Its task is to determine where to pipeline, which transformations to apply here and what resources to provide. The search is driven by information from the resource utilization table, as obtained by the estimation routines, as well as by the feedback from the assignment and scheduling process. This feedback contains information why the scheduling process failed to complete, more specifically which resources where in short supply or in high demand during some phase. Obviously, relaxing those tensions increases the chances of successful completion.

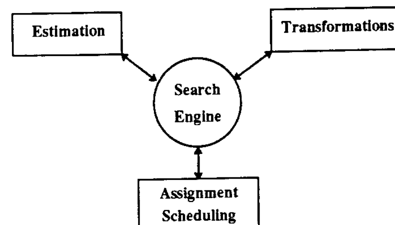

Figure 5: Design Space Exploration and Resource Allocation Process

The search mechanism attempts to minimize hardware by iteratively trying to remove some components. A resource is a good candidate for removal if it is expensive in area or if the utilization of the resource is not spread equally over time. For instance, in Table 2, the number of required adders in subgraph 2 is in disproportion with the required resources in the other subgraphs. A possible transformation is to extend the time allotted to subgraph 2 or to select a transformation (such as retiming) which reduces the min-bound on the additions in this sub-graph and in effect the overall min-bound (as is expressed in the bottom row of the utilization table). It must be noted that decreasing one resource might increase the requirements on other resources. The search will however favor those transformations, which have an overall positive result.

Since the search mechanism simultaneously has to address resource allocation and transformations, it is obvious that the optimization strategy should be flexible enough to handle the variety of constraints imposed by those problems. This suggest a probabilistic iterative improvement algorithm such as simulated annealing. However, the application of transformations (as well as the scheduling and the assignment process) is computationally expensive. We have therefore adopted a rejectionless probabilistic iterative search technique, where moves are always accepted, once executed [Wel84]. First experiments (the search mechanism is still under development at the time of writing) have demonstrated that convergence is normally obtained with a fairly small number of steps.

The effects of transformation process on the implementation of the 7th order filter are demonstrated in Table 3. The first row shows the resources, required to schedule the original graph in 13 cycles. The second row shows the requirements after a simple pipelining of the graph, while the third row contains the results for a pipelined and retimed graph (cfr. Fig. 4).

ASSIGNMENT AND SCHEDULING

The assignment and scheduling process is iteratively called by the search mechanism to determine if a given hardware implementation is feasible and, if not, to determine what resources are in short supply.

To obtain an overall good solution, it is essential that the scheduler simultaneously considers all resources (execution units, registers and interconnect) instead of handling them sequentially as is typically done in existing systems. The basic ideas and implementation details of the HYPER scheduling and assignment tools are described in [Pot89]. In short words, the scheduler can be described as a list scheduler, where a resource urgency measure is used as heuristic instead of the traditional time urgency measure. Some improvements have been introduced with respect to the published system : where the original system performs assignment and scheduling simultaneously, the current implementation assigns before scheduling. This simplifies the implementation dramatically, while compromising the solution only in a minor way. Furthermore, some more accurate heuristics have been introduced to measure the resource urgency.

The experimental results obtained for the 7th order filter in three different realizations is given in Table 3.

A	SH	SU	C	M	R	CP
5	3	1	9	28	42	13
4	2	1	7	21	36	8
3	2	1	6	20	33	7

Table 3: A is number of Adders, SH is number of Shifters, SU is number of Subtractors, C number of Connections, M number of Multiplexers, R is number of Registers, and CP is Critical Path for three different realizations of 7th Order Biquadratic Filter

CONCLUSIONS

A global optimization strategy for design synthesis has been presented. The red thread through the whole procedure is the resource utilization table, which measures the quality of the solution at a given point and helps to guide the overall control of the synthesis process. The majority of the algorithms described in this paper (such as the estimation, assignment and scheduling) have been implemented at the time of writing. Implementation of the transformation environment is currently under way.

ACKNOWLEDGMENTS

The authors acknowledge the contributions of P. Hoang and C. Chu. This research is sponsored by the Semiconductor Research Corporation (Contract No. 88-DC-08), DARPA (N00039-87-C-0182) and the Sony Corporation.

REFERENCES

[Chu89] C. Chu, et al., "HYPER : An Interactive Synthesis Environment for High Performance Real Time Applications", *Proc. IEEE ICCD Conf., Nov. 1989.*

[Goo89] G. Goossens, et al., "Loop Optimization in Register Transfer Scheduling for DSP Systems", *Proc. Design Automation Conf.*, pp. 827-831, June 1989.

[Har86] D. Harrison et al., "Data Management and Graphics Editing in the Berkeley Design Environment", *Proc. IEEE ICCAD Conf.*, Santa Clara, Nov. 1986.

[Hil85] P. Hilfinger, "A High-level Language and Silicon Compiler for Digital Signal Processing", *Proc. IEEE Custom Integrated Circuits Conference*, pp. 213-216, May 1985.

[Lam85] M. Lam, "A Transformational Model of VLSI Systolic Design", *Computer*, pp. 42-52, Feb. 1985.

[Lei83] C. Leiserson and F. Rose, "Optimizing Synchronous Circuitry by Retiming", Third Caltech Conf. On VLSI, March 1983.

[Mes88] D. Messerschmitt, "Breaking The Recursive Bottleneck", in *Performance Limits in Communication Theory and Practice*, Kluwer Academic Publishers, 1988.

[Pot89] M. Potkonjak and J. Rabaey, "A Scheduling and Resource Allocation Algorithm for Hierarchical Signal Flow Graphs", *Proc. Design Automation Conf.*, pp. 7-12, June 1989.

[Wel84] Welsh, D.J.A.: Correlated percolation and repulsive particle systems, Stochastic Spatial Processes, ed. Tautu P., Springer Lecture Notes 1212, pp.300-311, 1984.