APPlaUSE: Area and Performance Optimization in a Unified Placement and Synthesis Environment

Elof Frank Thomas Lengauer
German National Research Center for Information Technology
D - 53754 St. Augustin, Germany
lengauer@gmd.de

Abstract

We present a new methodology for high-level synthesis, which incorporates placement in an early phase of the synthesis process. This placement, prior to interconnect and storage allocation, allows for early and accurate estimates on area and net length. These cost factors are critical for the quality of the chip. The system has been implemented and extensive tests verify the estimates to be accurate.

1 Introduction

As of today, the overall synthesis process of a digital circuit is partitioned into two major phases, generally referred to as *high-level synthesis* and *physical design*. High-level synthesis is the process of computing a net list for a given behavioral description. One main aspect of physical design is placement and routing of the net list onto a rectangular surface, given area estimates on the width and height of each circuit block (node of the net list). Major cost terms considered in both phases are area and delay.

In high-level synthesis only rough estimates can be given for the placement costs, for example the sum of active hardware area (ALUs, registers, multiplexers) and the number of nets. Generally, these estimates form lower bounds on the exact area, as the overhead for wasted chip area and routing area due to placement are not known. Furthermore, without knowledge of the placement the length of each net cannot be accurately estimated. Yet, the maximum net length or path length, respectively, is critical for many designs, especially in DSP applications [1].

The incorporation of physical design issues during high-level synthesis is becoming commonplace. Several approaches [2, 3, 5, 11, 17, 23] have been presented throughout the last years. McFarland [17] was first to present an approach to high-level synthesis and placement. He synthesizes towards a two-dimensional data path architecture, with local buses within each data path and global buses connecting the local buses. By clustering the operations of the data-flow before scheduling and allocation, and cutting the cluster tree at different heights, different design alternatives can be evaluated. Unfortunately, the height of the data paths cannot be predicted. Therefore, neither wasted area nor net length can be minimized during the clustering phase.

Ewering [3] presents a synthesis methodology with a tight coupling between high-level synthesis and placement. He computes a *linear* placement of the functional units prior to register and interconnect allocation. In contrast, our target architecture is two-dimensional and allows for the synthesis of different design-styles within the data path, e.g. bit-sliced versus regular.

Recently, Fang and Wong [5] presented an integrated approach for iterative binding of functional units and floorplanning. Their approach binds operations of the data-flow to preallocated functional units. These functional units have already been assigned to a fixed floorplan. As in the 3D-scheduling approach by Weng and Parker [23], the influence of register and interconnect allocation on chip width and height is neglected in this approach.

Our approach uses a unified methodology, integrating the process of high-level synthesis and placement. Placement is done in an early phase of high-level synthesis, after scheduling and module allocation, but prior to memory and interconnect allocation. The structure to be placed is a communication graph, where the nodes represent physical hardware devices (ALUs) and the edges represent data transfers to be routed between these devices. The target architecture is a two-dimensional partitioned bus architecture (section 2).

Any placement prior to register and interconnect allocation can only be successful, if memory and interconnection requirements can be estimated beforehand (a priori) and incorporated into the placement. Otherwise, the placement will be of poor quality. In our method, we estimate memory and interconnect area during placement and validate these estimates later on. Our estimates prove to be accurate on typical data-flow dominated benchmarks.

Our paper is organized as follows. We present our target architecture in section 2. Section 3 deals with our methodology in detail as well as with the subproblems solved within our approach. Finally, we evaluate our results in section 4 and give conclusions in section 5.

2 Target Architecture

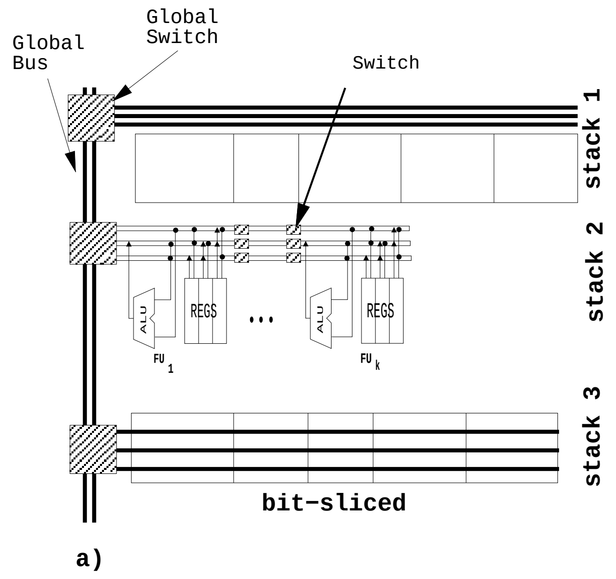

Figure 1: (a) Target architecture consists of a set of parallel stacks. (b) Interconnection within a stack through partitioned buses.

Figure 1(a) shows the target architecture. The architecture has the following characteristics (references point to similar architectures in the literature):

- The overall chip is composed of a set of parallel data paths, called stacks. These stacks are connected through global buses. [2, 16, 17, 22].
- A stack consists of a set of functional units, interconnected by partitioned buses [3, 18, 19]. Partitioned buses reduce not only wiring area, but also signal capacity. A data transfer does not require driving the complete length of a bus, but only those bus segments that lie on the route from its source to its target. Switches disconnect the active bus segments from the inactive ones. Thus, this architecture has advantages in reducing net length and therefore fetch cycle delay as well as minimizing power.
- Each functional unit includes local memory. This feature provides multiple storage locations for each value of the data-flow and hence flexible scheduling of communications [3, 7, 13].
- The routing graph has a tree structure and thus, for each communication the global route is uniquely determined. This is advantageous, when computing the communication density during placement (section 3).

We use two-phase clocking in our architecture. This clocking necessitates each value that is produced by an ALU to be stored in the local memory at the end of the execute phase. These values are then available for transfer at the beginning of the next fetch phase, at the earliest. The length of the execute phase is determined by the delay of the ALUs, and therefore depends predominantly on module selection. The delay of the fetch phase depends on the signal capacity of the bus segments to be driven. To a first approximation, we can assume the delay of the fetch cycle to be proportional to the maximum length of any data transfer. Thus, minimizing the maximum transfer length also minimizes the length of the fetch cycle.

We do not know of any high-level synthesis system, which can accurately estimate or compute the length of the fetch phase without knowledge of placement. To the best of our knowledge, APPlaUSE is the first system to minimize the length of the fetch cycle and the area of the resulting chip by integrating placement into high-level synthesis.

3 APPlaUSE methodology

Input to our system is a control/data flow graph (CDFG). We use the CASTLE system [24] for generating the CDFG from a behavioral description in VHDL, Verilog or C/C^{++}. The design flow of APPlaUSE is depicted in figure 2.

As in many high-level synthesis systems, we start off by performing data-flow graph scheduling, module selection and module allocation, followed by binding of operations to allocated modules. In the current implementation we use an integer programming formulation as in [9, 10] or a force directed approach [20] for time constrained DFG scheduling. All methods presented in this paper can handle a scheduled DFG, which includes pipelined ALUs and multicycling. The resulting data structure is a communication graph (figure 3(c)). The nodes of the graph correspond to allocated modules, the edges correspond to values of the data-flow, which have to be transferred over the buses. Each edge is labeled with the production time of its value and, for each terminal node, with the consumption time of this value at the corresponding ALU.

Many high-level synthesis systems continue with allocation of interconnect and memory. In contrast, we

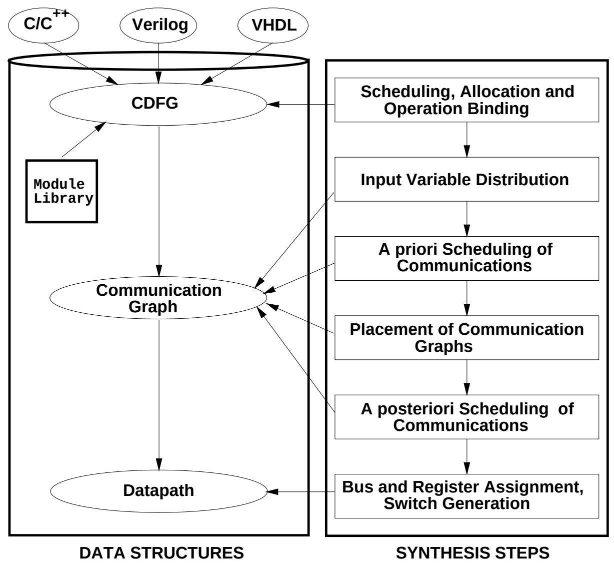
C/C++
Verilog
VHDL
CDFG
Module Library
Communication Graph
Datapath
Scheduling, Allocation and Operation Binding
Input Variable Distribution
A priori Scheduling of Communications
Placement of Communication Graphs
A posteriori Scheduling of Communications
Bus and Register Assignment, Switch Generation
DATA STRUCTURES
SYNTHESIS STEPS

Figure 2: APPlaUSE: Synthesis Methodology

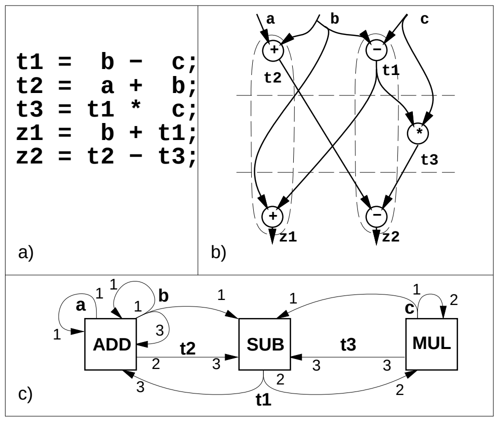

Figure 3: Example of a communication graph resulting from a data-flow graph. (a) program code (b) scheduled data-flow graph (c) communication graph

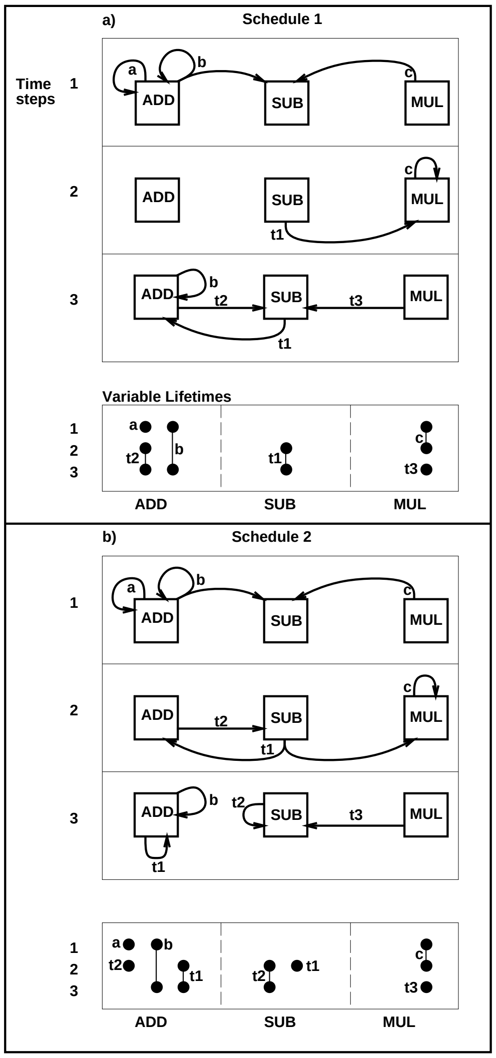

Figure 4: Two different schedules for the example in figure 3, resulting in (a) three buses and four registers and (b) two buses and six registers

first determine the placement of the communication graph onto our target architecture. *Placement of communication graphs* is the task of determining a location for each functional unit within the stacks, such that area, net length and the number of simultaneous data transfers in every region of the stack are minimized. The control step, at which these data-transfers are scheduled onto the buses, is variable within the given production and consumption time frame. *Scheduling of communication graphs* is the task of determining a control step for each data-transfer, such that the number of simultaneous transfers in every region of the stack is minimized. Consequently, placement and scheduling of communication graphs are interdependent tasks. The knowledge of transfer times influences the placement and the knowledge of placement influences the communication schedule. We resolve this interdependency using the following idea:

In order to determine transfer times, we cannot use the actual target architecture, since the placement is not yet known. Therefore, we use a simplified architecture, which is composed of a single linear data-path with non-partitioned buses as in figure 3. Each value, that is written onto a global bus, drives the complete length of the bus. Therefore, the locations of the source and target functional units are irrelevant to bus load. This architecture trivializes the placement problem sufficiently for computing the estimates. Our results justify this approach (section 4). For this architecture we solve an *a priori scheduling of communications* problem, using integer programming. We minimize a weighted sum of buses and registers. Increasing the priority of buses will increase the number of registers and vice versa, resulting in wider or higher designs, respectively. Details on how we solve this problem have been presented in [7].

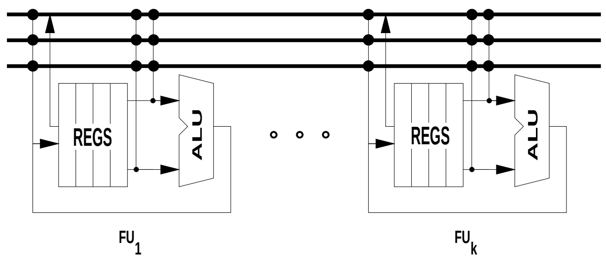

Figure 5: Linear data path architecture without partitioned buses

For any given placement of the communication graph on our two-dimensional *partitioned* bus architecture, we use the a priori communication schedule for calculating the local bus load (local communication density) and the local register requirement (local lifetime density) at each functional unit. Thus, we are able to estimate interconnect and storage allocation and, therefore, we can extend the bounding box of each ALU to include the area for the registers and the buses required for the respective functional unit (figure 6). This approach of added area during placement is also common in floorplanning with wiring-area estimation [14, 25]. Also, care has to be taken in es-

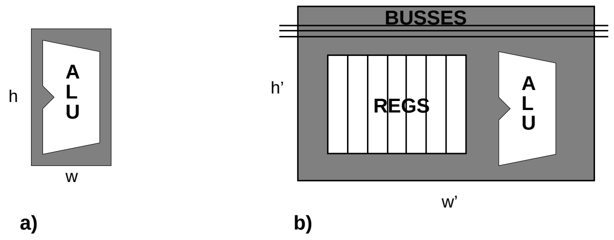

Figure 6: Memory and interconnect estimation in APPlaUSE. (a) Bounding box without extra area. (b) New bounding box for each functional unit incorporating estimated area for registers, buses, multiplexers, and drivers.

timating the area for multiplexers, bus drivers, and switches. The exact number of drivers and multiplexers as well as the number of switches partitioning the buses can be determined only after bus and register assignment. For each estimated register, we add extra area of half the width of a multiplexer and one and a half the width of a driver. We have extracted these approximate static factors by inspecting our synthesized results for the benchmark set [8] (section 4). The number of switches is estimated by analyzing the communication load in each data path [6].

After placement, we determine an optimal communication schedule with respect to this placement, which we call *a posteriori scheduling of communications*. We extend the integer program formulation for the non-partitioned architecture to the partitioned two-dimensional architecture. Here again, we have the possibility to trade off buses against registers.

After this phase, we know the placement of the functional units and the data transfer times. With this knowledge, the number of registers and buses is determined. We proceed with register and bus assignment, as well as multiplexer, bus driver and switch generation and finally complete the resulting data path. These subtasks are modeled as well-known graph theoretic problems and are solved using optimal or heuristic approaches. Specifically, we use the left-edge algorithm for register assignment [12], Tseng's heuristic [21] for bus assignment and we solve a minimum clique partitioning problem on interval graphs for switch generation. Details of these methods are given in [6].

4 Results

In order to evaluate our system, we have to answer two questions.

1. How good are the estimates on area and net length, which we compute during placement? In order to answer this question, we compare the estimates of the minimal cost placement, which we choose after simulated annealing, to final chip area and net length.

2. How good are our designs in comparison to other systems? This question is somewhat more difficult to answer. There have only been few publications of data path chip area for the benchmark set. We compare our system with HAL [20], PARBUS [3] and CASS [2].

We have synthesized the well-known benchmarks *Differential Equation Solver*, 5^{th} *Order Elliptical Filter*, and *Discrete Cosine Transformation* (DCT) [8] with different bounds on the number of control steps and also different module libraries, resulting in different schedule lengths and numbers of allocated ALUs. Here we present the results for the DCT example. Table 1 shows our results for this example and answers the first question above. The estimated area and net length are given and, after each figure, respectively, the deviation in percent from the final area and the maximum net length is presented. For example, in row 1 APPlaUSE estimated an area of 11.52 mm^2 and a maximum net length of 6188 μm. The exact values were 11.88 mm^2 for the area and 5870 μm for the maximum net length. For all benchmarks the estimates were within a range of -12% to 12% for both the area and the maximum net length. Note, that this maximum net length estimates the length of the fetch phase in a first approximation. The main deviation is in the width of the data path, because we use static estimates for the number of multiplexers and drivers.

The first four designs for the DCT use a schedule length of 7 clock cycles and 14 ALUs, the other four use a length of 10 clock cycles and 9 ALUs. Thus, each block of four designs refer to the same communication graph. Each of the four designs vary in two other parameters α and β. α controls the number of buses and registers. A small value of α minimizes the number of buses and a large value of α minimizes the number of registers. Similarly, a small value of β minimizes the maximum net length and a larger value of β minimizes area. The accuracy of the estimates allows for an evaluation of physical design parameters and the selection of the *"optimal"* design of all design space alternatives prior to register and interconnect assignment. Our experiments have shown, that the time for a design space exploration with different parameters of α and β can be reduced by approximately 25% to 40%, due to the fact that the final assignment phase is performed only once.

Ewering [4] presented results on data path chip area for a linear, partitioned bus architecture in the PARBUS system. He compared his results to designs made by HAL [20]. The designs by HAL are targeted towards a multiplexed architecture, which Ewering handed over to a sophisticated place-and-route tool [15] for computing the chip area and net lengths. In order to compare to the results of Ewering, we use his hardware library, which corresponds to a $2\mu m$ technology. Further, we take over his assumption, that

	α	β	Area (mm^2)		max. Net Length (μm)	
			Est	%	Est	%
7 cycles 14 ALUs	0.2	0.5	12.62	-3	6188	5
	0.8	0.5	14.13	5	6340	3
	0.2	0.9	11.53	-1	8088	6
	0.8	0.9	11.39	3	7040	8
10 cycles 9 ALUs	0.2	0.5	8.76	9	6012	1
	0.8	0.5	8.71	12	5580	1
	0.2	0.9	7.10	6	6712	0
	0.8	0.9	6.87	11	6316	5

Table 1: Results for the DCT

multipliers can be designed in a bit-sliced fashion and therefore only one linear data path is feasible.

The multiplexed architectures by HAL proved to be nearly twice as large in area as those of PARBUS. Our results are 1% to 8% better than those of PARBUS and also 5% to 8% better than those of CASS.

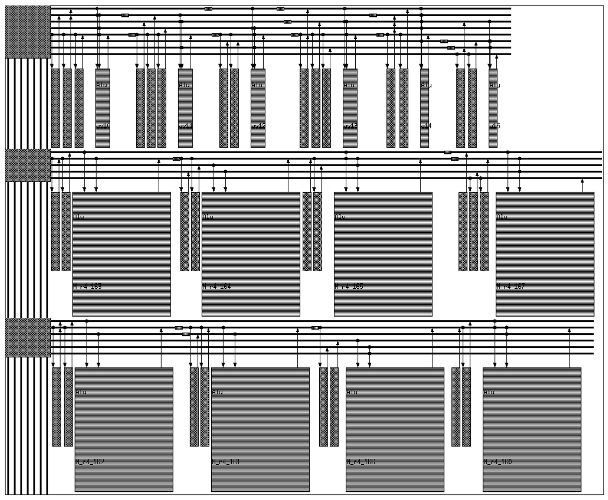

Figure 7: Screen dump of the synthesized Discrete Cosine Transformation. This design is optimized for area.

5 Summary and Conclusions

We have integrated placement into high-level synthesis by placing an intermediate structure, called the communication graph. The nodes of the communication graph are labeled with tight estimates on the width and height of the hardware needed for each functional unit. We derive these estimates by scheduling the data transfers of the data flow on a restricted architecture prior to placement. After placement we compute the optimal communication schedule together with the allocation of register and interconnect. The tight estimates on physical design data allows a considerable reduction in design time, because the designs can be evaluated prior to interconnect and register allocation and assignment. This difficult and thus rather time consuming optimization procedure only need to be performed once for the best design.

Acknowledgements

The authors wish to thank Raul Camposano, Majid Sarrafzadeh, Frank Buijs, Salil Raje, as well as all members of the OMSI project and of the SYDIS project for their helpful discussions on this work. Thanks also to Burkhard Reike and Frank Bertelsmeier for their programming efforts on substantial parts of the system. This research was supported by a grant from the German Ministry for Education and Research (BMBF) under grant No.: 01-IS-102 C/0.

References

[1] R. Camposano and W. Wolf. *High-Level VLSI Synthesis.* VLSI, Computer Architecture and Digital Signal Processing. Kluwer Academic Press, 1991.

[2] A. A. Duncan and D. Hendry. DSP Datapath Synthesis Eliminating Global Interconnect. *Proceedings of the EURODAC Conference 1993*, pages 46 – 51, 1993.

[3] C. Ewering. Automatic High-Level Synthesis of Partitioned Busses. *Proceedings of the International Conference on Computer Aided Design*, pages 304–307, 1990.

[4] C. Ewering. *Methoden zur flächenminimierenden Hardware–Synthese.* PhD thesis, University of Paderborn, Paderborn, Germany, 1991.

[5] Y.-W. Fang and D. Wong. Simultaneous Functional-Unit Binding and Floorplanning. *Proceedings of the International Conference on Computer Aided Design*, pages 317–321, 1994.

[6] E. Frank. Placement-Driven High-Level Synthesis. Technical report, GMD, elof@cartan.gmd.de, 1995.

[7] E. Frank and T. Lengauer. Optimal and Robust Scheduling of Communications in Bus Architectures. In G. Saucier and A. Mignotte, editors, *Novel Approaches in Logic and Architectural Synthesis.* Chapman & Hall, to appear 1995.

[8] *The 7th High-Level Synthesis Workshop Benchmarks*, 1994.

[9] Y.-C. Hsu and Y.-L. Lin. High-Level Synthesis in the THEDA System. In R. Camposano and W. Wolf, editors, *High-Level VLSI Synthesis*, pages 283–306. Kluwer Academic Publishers, Boston, MA, 1991.

[10] C. T. Hwang, J. H. Lee, and Y. C. Hsu. A Formal Approach to the Scheduling Problem in High-Level Synthesis. *IEEE Transactions on Computer-Aided Design*, 10(4):464–475, 1991.

[11] T. Kakuda and M. Fujita. An Approach for Layout-Driven High-Level Synthesis. *5th International Workshop on Computer Aided Design*, pages 15–20, 1991.

[12] F. J. Kurdahi and A. Parker. REAL: A Program for REgister ALlocation. *Proceedings of the 24th Design Automation Conference*, pages 210–215, 1987.

[13] D. Lanneer, M. Cornero, G. Goossens, and H. DeMan. Data Routing: A Paradigm for Efficient Data-Path Synthesis and Code Generation. *7th High-Level Synthesis Symposium*, pages 17–22, 1994.

[14] T. Lengauer. *Combinatorial Algorithms for Integrated Circuit Layout.* Teubner–Wiley Series of Applicable Theory in Computer Science. John Wiley & Sons, New York, 1990.

[15] T. Lengauer and R. Müller. A robust framework for hierarchical floorplanning with integrated global wiring. In *Proceedings of the International Conference on Computer-Aided Design*, pages 148–151. IEEE, 1990.

[16] W. Luk and A. Dean. Multi-Stack Optimization for Data-Path Chip (Microprocessor) Layout. *26th Design Automation Conference*, pages 110–115, 1989.

[17] M. C. McFarland. Using Bottom-Up Techniques in the Synthesis of Digital Hardware from Abstract Behavioral Descriptions. *23rd Design Automation Conference*, pages 474–480, 1986.

[18] A. Mignotte and M. Crastes de Paulet. Resource Assignment with Different Target Architectures. *Proceedings of the International Conference on Computer Design: VLSI in Computers*, pages 172–177, 1991.

[19] F. Monteiro, B. Rouzeyre, and G. Sagnes. High-Level Synthesis: A Data Path Partitioning Method Dedicated to Speed Enhancement. *Proceedings of the European Design Automation Conference*, pages 123–128, 1991.

[20] P. G. Paulin, J. P. Knight, and E. Girczyc. HAL: A Multi-Paradigm Approach To Automatic Datapath Synthesis. *23rd Design Automation Conference*, pages 263–270, 1986.

[21] C.-J. Tseng and D. Siewiorek. Automated Synthesis of Data Paths in Digital Systems. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 5(3):375–395, 1986.

[22] P. Villarrubia, G. Nusbaum, R. Masleid, and P. Patel. IBM RISC Chip Design Methodology. *Proceedings of the International Conference on Computer Design: VLSI in Computers*, pages 143 – 147, 1984.

[23] J.-P. Weng and A. C. Parker. 3D Scheduling: High-Level Synthesis with Floorplanning. *28th Design Automation Conference*, pages 668–673, 1991.

[24] J. Wilberg, R. Camposano, M. Langevin, P. Plöger, and H.-T. Vierhaus. Tools for Generating a VHDL Processor Description and a Compiler Back-End from a Single Schematic. *Proceedings of the International IFIP Workshop on Logic and Architecture Synthesis*, 1994.

[25] G. Zimmermann. A New Area and Shape Function Estimation Technique for VLSI Layouts. In *Proceedings of the 25th Design Automation Conference*, pages 60–65. ACM/IEEE, 1988.