5. Design Guidance and Design Space Exploration

For several different areas in hardware design, several alternative solutions exist, which may follow a general architecture, but differ in some properties, such as e.g. a serial or parallel multiplier implementation. The problem of design space exploration consists in the proper selection of design alternatives in order to meet a given design goal. This goal consists typically in the optimization of a property, while meeting given constraints on other properties. Often, a trade-off has to be found between several desired aspects, e.g. performance and area consumption, which depend on the selection of the design alternatives. In this section, the problem of design space exploration will be formulated in a general form, and a selection of previous work in this sector will be described briefly.

5.1 Problem Formulation

Let a design have a number n of design properties P_i, $1 \leq i \leq n$. Each property P_i is a set of alternatives for this property:

$P_i = \{p_{1i}, ..., p_{ki}\}$;

As an example, a property "Multiplier" could resemble the following set of alternatives: {"parallel", "serial", "booth"}. For practical purposes, it can be assumed, that k_i is finite for every $1 \leq i \leq n$. Then the design space DS can be described as an n-tuple as follows:

$DS = (P_1, ..., P_n)$;

A specific design d representing a point in this design space is given by an n-tuple of the form:

$d = (p_1, ..., p_n)$; $p_i \in P_i$, $1 \leq i \leq n$.

Note, that the properties P_i may apply to the design itself as well as to the design steps to be taken, thus representing alternative design actions. An obvious example for such alternatives are optimization operations, which can be applied to optimize the design for different objectives, possibly decreasing the quality of other properties, like optimization for either area efficiency or for performance.

Furthermore, the p_i may be alternative implementations of a component or different numbers of occurrences of the same component. The design space exploration problem consists in finding such a design tuple d, which comes as close as possible to a given design goal. This goal may consist of objectives to be

minimized and constraints which have to be met. Typical objectives or constraints are performance, chip area, or power consumption. Two basic tasks in the exploration process can be identified:

- Status evaluation. A given status during the process, representing a point in the design space, must be judged, in how far it satisfies constraints and the given objective. Depending on the approach, this can be done by models or by simulations.
- Status generation. The system is supposed to generate a point in the design space, which represents a good solution according to the objective. This is the main task of design space exploration. Methods to reach this point include iterative refinement or analytical techniques.

In the following, approaches used in existing systems for these tasks, as well as existing exploration strategies are discussed.

5.2 Design Space Exploration Systems

In order to search a given design space, several general strategies have been used in existing works in the literature. Basically, systems for design space exploration can be distinguished into interactive and non-interactive systems. Interactive systems are mostly meant to give advice during the design flow, leaving the main responsibility with the designer. Such systems are sometimes referred to as design guidance systems. Non-interactive systems are meant to perform an automatic exploration, creating the desired optimized solution.

Interactive approaches assist the designer in the process to develop the optimized solution. The designer represents a correcting and steering component in the system. Some approaches avoid the status generation and provide only information to the designer. This may include prediction of the effects of a planned design step, simulation results, or guided examination from a knowledge data base. However, the designer still has to take the design decisions, using expert knowledge. More sophisticated systems provide also status generation by incorporating expert knowledge and presenting advice to the designer. This advice may consist of several proposals, so the designer can select the most suitable one . Also, the advice can be ignored by the designer, who is thus a correcting element in the process. To implement the expert knowledge, a rule-based approach can be employed, realizing a kind of expert system [Barr86].

Non-interactive systems generate an optimized solution without the need of user control. Some systems employ a sequential process, using again expert knowledge to generate new states. Implementations found employ rule-based knowledge representation or a fuzzy learning approach. The rule-based

approach has the advantage, that rules allow a very flexible knowledge representation, thus also design processes consisting of quite different phases can be handled. Classical optimization strategies are also found, but require a consistent representation of quality. Finally, analytical techniques require appropriate models for both the status evaluation and the status generation tasks. These models are often hard to determine and are typically not universal.

The techniques applied for the evaluation of a given state are determined by the general methodology of the system. Basically, most systems work with models of the design point to estimate relevant properties. These models may describe the behavior of the system, or they may predict the effect of a design step. Analytical methods require models appropriate for a numeric optimization algorithm. Other approaches found are looking up the expected effects of a design step in a library, fuzzy membership functions, or simulation of benchmarks to get performance estimations.

The systems described in the following subsections are listed in table 5-1, showing the approaches employed for state evaluation and state generation.

Publication Year	System / Target Architecture	Interactive	State Evaluation	State Generation
1991	ADAM Design Planning Engine	No	Abstract Models	Rule-Based
1992	Clio System	Yes	Hierarchic Prediction Models	None
1998	Datapath-Intensive ASICs	Yes	Prediction Using Library	Rule-Based
1998	Raw Processors	No	Analytical Models	Analytical
1998	ICOS System	No	Fuzzy Membership Functions	Fuzzy Learning and Greedy Search
1999	Multimedia Processors	No	Simulation	Branch and Bound

Table 5-1: Previous Design Space Exploration systems

The Clio system and the design space exploration framework for datapath-intensive ASICs by Guerra et al. [GPR98] are interactive, while the other four systems are non-interactive. Both the ADAM and the Clio system are meant for VLSI design. ADAM generates a plan to design the target circuit, while Clio

assists the user at the conceptual design. The exploration system for datapath-intensive ASICs can be used to create an optimized application specific circuit. However, this system is limited to one application. The framework for processors for RAW machines uses analytical methods to generate architectures optimized for specific applications. However, the approach requires accurate models not only for the processors, but also for the applications themselves, which are not always easy to determine. The ICOS system for multiprocessor systems uses machine learning to benefit from earlier experience for the synthesis of new components. If a component has not yet been learned, a number of alternatives is explored, getting an evaluation by combination of fuzzy membership functions. The exploration framework for multimedia processors uses simulation of a large number of benchmark applications.

5.2.1 The ADAM Design Planning Engine

An early approach of automatically planned design is the Design Planning Engine (DPE) [KnPa91] of the ADAM (Advanced Design AutoMation) [GKP85] system, published in 1991 by Knapp and Parker. This framework is meant for a VLSI design process and allows a designer to specify a description of the desired system and constraints for several design properties like area, cycle time, power, and time to design. The DPE then generates a plan how the target design, which satisfies the constraints, can be generated.

The design plan consists of a sequence of design steps, each resembling the application of an according design tool, like a microprogram generator, an optimizer or a PLA generator. Each step will generate a piece of design data, like a schematic, a data flow graph, or a layout. There are typically several options in each step, leading to different properties of the final result. The problem consists in the proper selection of one option in each step. The global approach for this task is sketched in figure 5-1.

The DPE approach distinguishes a planning space and an execution space. While the execution space holds all the actual tools and data for the design process, the planning space consists of abstract models, where the planner works on. There are two inputs to the DPE system. The first input is a specification for the target system in a so-called design data structure (DDS), which contains also a dataflow graph being used by the DPE. The second input consists of high-level information about the specification, including constraints on the target. Given these inputs, the planning engine creates an according design plan. This plan is then executed using the actual design tools in the execution space. If the execution is successful, the result is a correct design which meets the input constraints. Otherwise, a new design plan is constructed.

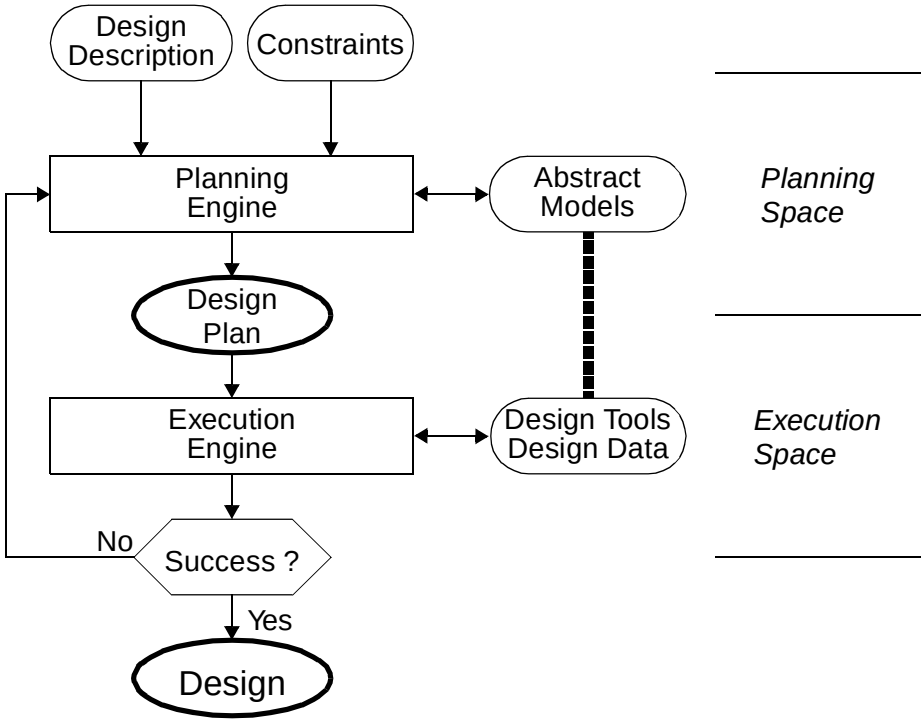

Figure 5-1: The ADAM Design Planning Engine

During the planning process, the DPE constructs a plan by adding a new design step to the chain of already existing design steps, until the process is finished. To find the next step, the system uses models of the planning space. The real design data items are represented in the planning states by assertions, while the models for operators (the design tools) include preconditions and postconditions. The preconditions are a set of assertions, that must be true to enable the operator to be applied to the current design state. The postconditions are either a set of assertions that will be true after the operator has been applied, or a set of rules, which allow to generate more complex assertions. For each design step, the DPE finds all operators, which can be applied according to the current set of assertions. Then, for all these operators, the change of state is calculated, which would result, if the operator would be applied. This is done by deleting or adding of assertions to the current design state. The most promising of these new states is selected to be the next current state, with the according operator being added to the design plan. The selection of the state is guided by a scalar value expressing the advisability of the operation. This advisability information is computed by using an inference engine from three sources: Rules from the operator frame whose advisability is currently being ascertained, assertions about the state of the design to which the operator would be applied, and the importance of each measure appearing in the constraints.

In order to satisfy multiple constraints of different design measures, like area, or performance, the DPE iterates the planning algorithm described above multiple times until a plan is found, which leads to a design satisfying all constraints. This iteration algorithm is based on the possibility to assign different importance values to the design dimensions. During iterations, these importances are adjusted using an interpolative approach, which is described shortly in the following:

As mentioned above, the different design dimensions, like area, performance, power dissipation or design time, can be given a specific importance value. The DPE starts its first iteration by making a set of plans, one for each possible design dimension, where the importance for this specific dimension is set to maximum and the importance for all other dimensions is reduced to zero. The result is a set of plans, each assuming one dimension most important and all the others unimportant. After the planner has built these plans, it then attempts to calculate the relative importance of each dimension by interpolating the given constraints of the design between the hypothetical results, which are estimated from the hypothetical designs produced by the set of single-objective plans. The result of the interpolation are now new importance values for the design dimensions. The planner then constructs a new plan with these importance values. The properties of the final state of this plan are again used, together with the previous results, to calculate new importance values. This cycle is repeated until the new plan is the same as the old one. The resulting plan is then executed like described above.

5.2.2 The Clio System

In 1992, Lopez, Jacome and Director published the Clio design assistance tool [LJD92] as part of the Odyssey CAD system [JaDi92], [SBD93]. Although this is a VLSI CAD system, the concepts of the Clio tool can be generally accepted for design assistance. Clio assists the user during the conceptual design, which includes the analyzing or predicting the outcome of alternative design decisions. The Clio tool comprises two subsystems for different types of assistance:

- A prediction subsystem, and
- An advice subsystem.

The prediction subsystem provides quantitative estimates of the consequences that each valid design option would have on relevant design properties like area or performance. The advice subsystem allows the designer to interactively consult a knowledge database, which contains advice from other designers.

Prediction of quantitative design properties is based on a collection of predictive models, which generate according property values for a given design specification. According to the design process being seen by the manager as a sequence of synthesis steps, each model is aimed at simulating the behavior of a

specific step. Further, knowledge for design domains is organized in a static hierarchy of knowledge templates. A path from a template at a high level in the hierarchy downwards resembles a specialization in the design process, with descendant templates inheriting knowledge from all parent templates. An example for such a specialization starting from a top level template "VLSI Circuit" would be "VLSI Circuit" - "Digital Circuit" - "Logic Level" - "Adder" - "Manchester Carry". In this example, "Manchester Carry" resembles the most completely designed domain template.

The Clio tool uses this hierarchy for the prediction process. Whenever a prediction is requested, Clio searches for a prediction model for the current template. If an appropriate model does not exist, the tool moves up in the hierarchy until it finds a template containing a prediction model that provides the requested types of estimation values. This technique guarantees, that the most accurate prediction model is used, but provides also a rough prediction where exact estimations are not possible.

The execution of a prediction task in Clio is done by mapping each prediction task dynamically onto a prediction job, consisting of a prediction model and a control module. The prediction model is retrieved during run-time from the knowledge base and specifies the means for calculation of the desired estimation values. Such a model is either an atomic function, which directly computes the desired value, or a hierarchy of lower prediction jobs, resembling a recursive execution of estimation functions. The control modules allow arbitrary control structures, including conditional loops. The structure of prediction jobs in the Clio system is illustrated in figur e5-2.

The advice subsystem of Clio can be invoked interactively and provides static information about different design options associated with a given design issue. This information is contained in an advice database representing the collective memory of the entire community of designers utilizing the framework. Thus, this aspect of the Clio system can be seen as a means for interchange of information and experience. The information can be organized by a predefined collection of discrimination factors that specify the basis by which the design options are to be compared.

5.2.3 Design Guidance for Datapath-Intensive ASICs

In [GPR98], Guerra et al. proposed a design guidance environment targeting optimization for semi-custom datapath intensive ASICs at behavioral level. The framework uses an interactive approach for the exploration process, with alternate action of the system and the designer. The system suggests different possible optimizations to improve area, power or throughput of the design. Optimizations include time and loop unfolding, algebraic optimizations,

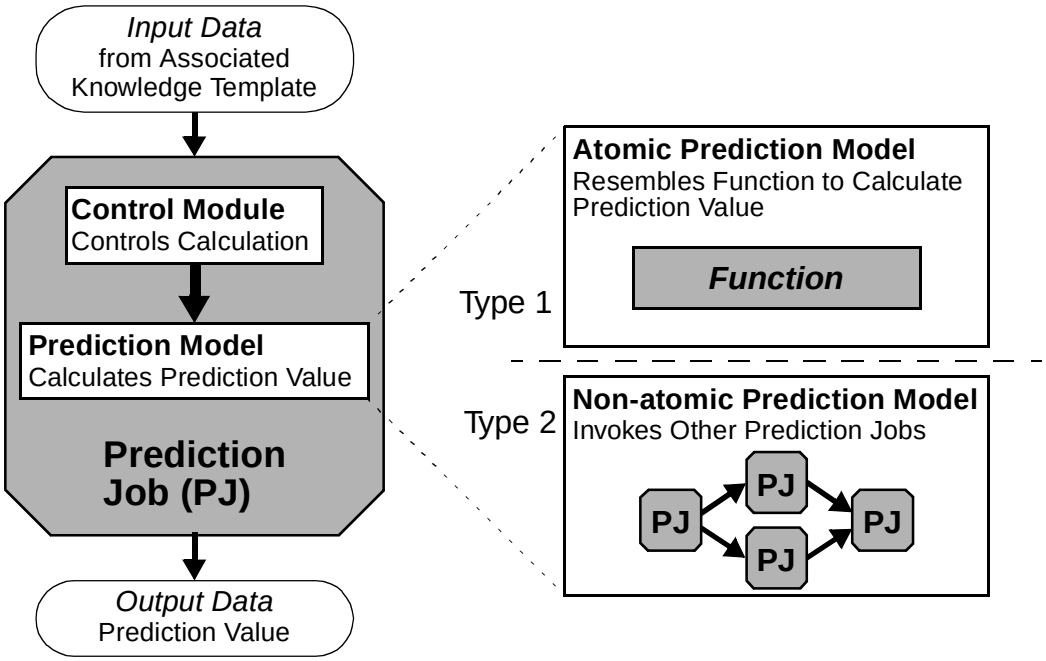

Figure 5-2: Structure of prediction jobs in the Clio system, showing two types of prediction models.

pipelining and retiming, voltage scaling, clock selection, operator chaining and hardware module selection. The suggestions are based on characterizations, which allow to provide an estimation of the optimization effects. Although the suggestions are based on encapsulated expert knowledge, the designer may as well chose a different approach.

The basic design process starts with the entry of a design specification in C++ or the Silage language [GeHi89] as well as constraints and design goals. The design guidance system analyses the design and generates feedback, which the designer uses to direct the exploration process by selecting a suggestion or proposing another alternative. The cycle continues until an optimized design has been found.

An overview on the components of the design guidance system is shown in figure 5-3. Starting from a version of the design, an optimization is applied according to the directions from the designer. The resulting new design version is then put through several estimators [RCHP91], [MeRa94], [GPR94], [RGM95] for determining its throughput, area and power dissipation. The next step analyses the current design and extracts its essential properties, considering also algorithm properties like regularity, which affects the interconnect requirements. The resulting data is used in the next step, which generates the design guidance. This step evaluates and ranks the available design options employing a library of pre-characterized optimizations.

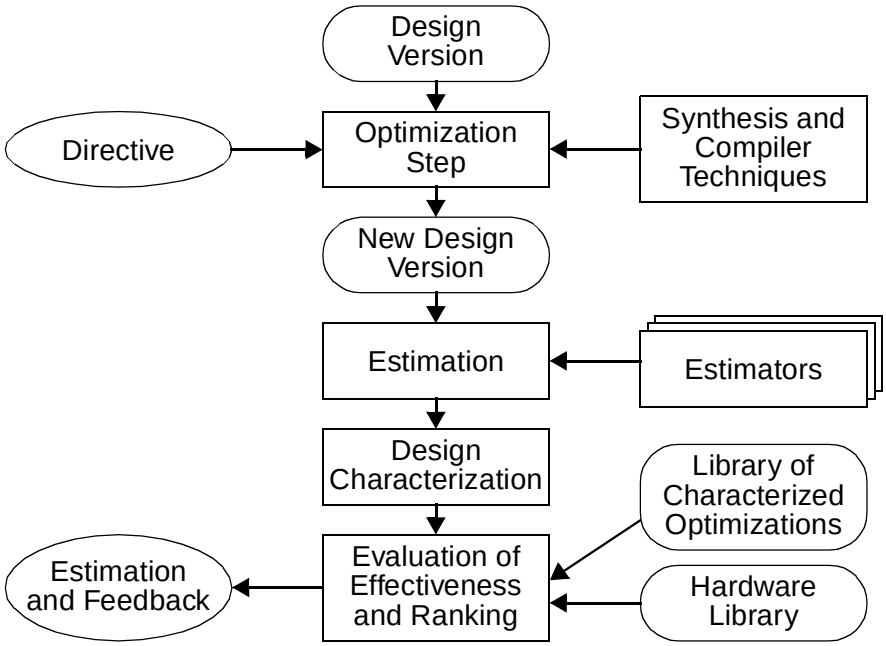

Figure 5-3: Interactive design space exploration for datapath-intensive ASICs by Guerra et al.

The design characterization step and the final evaluation are the backbone of this approach for the generation of design guidance information. The design characterization analyses the design and gathers specific properties. These properties are then used to identify situations in which certain optimizations will work well and also to predict the effectiveness of these optimizations. The properties gathered by the design characterization include number of operators, bitwidth, number of memory accesses, critical path, lifetimes of variables, regularity of the data flow graph, iteration bound, and others.

The task of generating suggestions from this information is divided into two sub-steps. First, the system identifies all optimizations, which can be performed at all regarding the design properties. An example for such an optimization is the expansion of constant multiplications into adds and shifts, which can only take place if the computation features constant multiplications. After all possible optimizations have been identified, they are ranked before presented as options to the designer. The ranking is done considering the estimated effectiveness of each optimization, which in turn is calculated from three factors resembling the immediate effect that the action has on the design metrics, the potential the optimization has for other optimizations, and the resulting feasibility in meeting constraints. These measures are calculated from the properties by a set of

parameterized rules. The final effectiveness of an optimization is then calculated by a linear combination of these three measures, with the coefficients being empirically derived.

5.2.4 Design Space Exploration for Raw Microprocessors

In [MYA98], Moritz et al. present an analytical framework for the exploration of design alternatives for processors of the Raw architecture described in section 2.1.6 [WTSS97], [AABF97], [Tayl99]. The proposed environment produces an optimal Raw architecture for a given application with the focus on optimal performance for this application. The optimization process uses a given area budget and the balance of the processor tiles as constraints. The term balance denotes the proportion of area in each processor tile used for its ingredients, which are the processing part, memory, communication part, and I/O.

A coarse overview on the framework is shown in figure 5-4. The approach uses models for the processor architecture, the application, and the area costs. From the application and architecture models, the run time is estimated by performance functions. In the following optimization, the run time is minimized under cost and balance constraints. The cost constraint resembles a predefined budget of available chip area. The balance constraint is expressed by the condition, that the computation time and the communication time of the application should be equal, so the according resources of the optimal architecture are balanced.

The architecture model describes a configuration by a nine-tuple, including properties like the number of tiles in the array, the processing power of one processor in operations per cycle, the amount of SRAM in a tile, the local communication bandwidth, the interconnect latency for one word and one hop, and others. The total cost of the system is composed of the processor and memory costs per tile, cost for the local communication router, the cost for the global latency and the cost for the global bandwidth. The cost measures are expressed in the required chip area for the according resources. All measures are derived from the property values of the architectural model using approximative functions. The application model contains functions and parameters for the characterization of application performance, including measures like the problem size, the total amount of computation required per tile, the amount of memory needed, the buffer space needed for communication, and the amount of communication. Such a model has to be provided for every application to be explored. Based on the application model, the run time for a given problem size can be calculated by performance functions using the architectural model. These functions generate also information about the balance, which forms one constraint for the following optimization. With the cost function being the other constraint, the exploration can be described as a constraint based nonlinear optimization problem. The

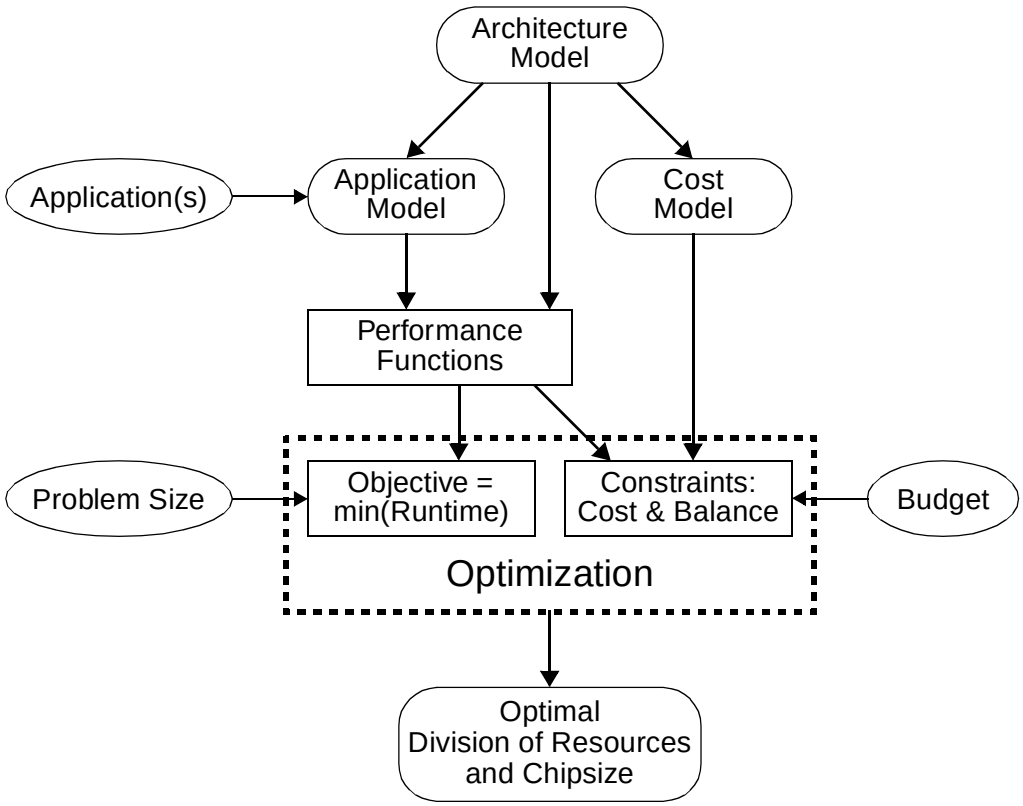

Figure 5-4: Design exploration framework for Raw microprocessors

optimization objective is to find a minimal execution time for a given problem size on a specific architecture by considering the area budget and the balance condition. As the problem size is fixed, the solution to the optimization problem is the desired optimal architecture.

5.2.5 The Intelligent Concurrent Object-Oriented Synthesis System

The Intelligent Concurrent Object-Oriented Synthesis (ICOS) was published in 1998 by Hsiung et al [HCLC98], based on the Performance Synthesis Methodology (PSM) by the same author published earlier [HCHW96]. It is meant for the automatic synthesis of a multiprocessor system from a set of system descriptions, performance constraints, and a cost bound. The resulting architecture is generated by determining the number and type of processors used, the processing cluster organization, the type of system interconnection, and the amount of memory with its logical and physical organization. ICOS features an object-oriented technique, fuzzy design space exploration, the possibility for

concurrent design of several parts of the system, and intelligent reuse of previously designed subsystems. The latter is realized by employing machine learning techniques. An overview on the ICOS design flow is given in figure 5-5.

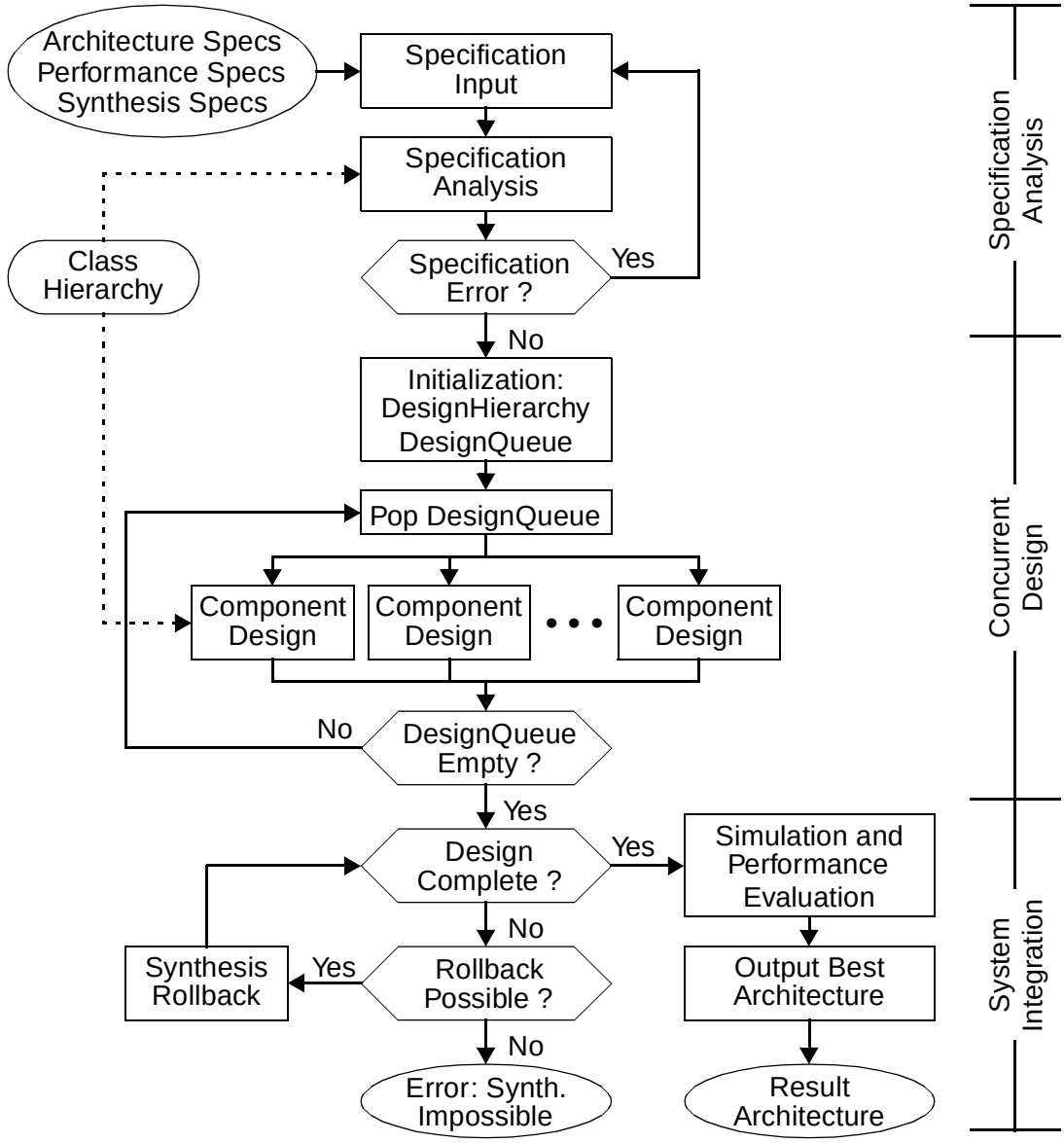

Figure 5-5: The ICOS design flow

The designer first specifies the system requirements using a special language. The specification includes architecture, performance, and synthesis descriptions. Architecture specifications allow the designer to restrict the domain space by indicating how a system part should be constructed. Performance specifications include the minimum system power, the minimum system scalability, reliability,

and fault-tolerance. Synthesis specifications include the maximum number of design alternatives to consider for further design and the choice of whether to reuse previously learned designs or to design a system from scratch.

After the specification, the design process takes place in three main steps: Specification analysis, concurrent design, and system integration. In the specification analysis phase, the specification is checked for technical, logical and typographical errors. Contradictions in the specification are detected by logic rules based on common architecture assumptions. The concurrent design is the main phase, where components of the system are either synthesized concurrently or reused from previous synthesis, employing machine learning techniques. This phase is based on an object-oriented class hierarchy to describe a computer system. A subclass of the class hierarchy, called design hierarchy, is used to keep track of the system structure under design. Further, a design queue holds the components ready for synthesis. Components which are not dependent on others to be designed earlier are synthesized concurrently. Before the actual synthesis takes place, a component class checks if learning from previous design experience is possible, using fuzzy specification-guided learning. If a reuse is viable, the actual design step is skipped and the previously learned component is used instead. The fuzzy specification-guided learning is based on a fuzzy comparison of the desired specifications of the target design with specifications of earlier designs stored in a learning hierarchy. The comparison is made using a fuzzy set representing the functional proximity of previous designs to the current one. The according membership function assigns a negative value to a previous design, if at least one specification is not satisfied, and a positive value in [0,1], if all specifications are satisfied by this design. The higher the positive value, the better is the match between the requirements and the design specifications. The membership function is calculated as a sum of the partial proximities of the single specifications, with a different calculation method for each specification.

If no reuse is possible or the designer specified not to reuse previous designs, the current component is synthesized. If the current component class is an aggregate of subclasses, i.e. the component consists of others, each subclass is considered separately and appended to the design queue for further synthesis. If the current component class is a generalization of subclasses, i.e. there are different alternative implementations for this component, a fuzzy design-space exploration takes place to select a suitable number of acceptable design components that are among the best specializations of the generalization. The fuzzy exploration allows to assign a partial order of preference to a set of specializations by assigning each specialization a penalty factor and sorting them ascendingly by this factor. The penalty factor is calculated by modeling the impact of a component to each performance factor of the whole system by a fuzzy membership function for each performance factor. The performance factors are a

set of goals and constraints, and the according membership functions assign a value between 0 and 1 to each specialization, according to the grade it satisfies the respective goal or constraint. The final penalty factor is then calculated as a linear combination of the membership functions. The coefficients for the goals and constraints are chosen in a way, that the penalty factor lies also in the range of [0,1]. The specializations with the smallest penalty factors are chosen for instantiation. After a component is designed, it is stored in the learning hierarchy for future reference and possible reuse.

In the last phase of the ICOS design flow, the full system under design is integrated, simulated, and its performance evaluated. Because of the concurrent synthesis approach, a final checking for design completion is necessary. If the design cannot be completed, synthesis rollback occurs to find other possible design alternatives. Otherwise, a design with the best performance is the final architecture output.

5.2.6 Design Space Exploration for Multimedia Processors

In 1999, Kin, Lee et al. published a framework for the design space exploration of application specific processors [LKPM99], [KLMP99]. The target architecture resembles a multiprocessor system with shared memory, where all processors are laid out on a single chip. The processors are basically general purpose, but optimized to run multimedia applications. The authors presented experiments where an optimal multiprocessor system was found for a set of multimedia applications, which should run partially in parallel on an architecture comprising several of the processors to be explored. The exploration process took place with the aim to optimize the power dissipation under area constraints [KLMP99]. Another experiment was published, optimizing area with performance constraints [LKPM99].

The base architecture for the exploration process is the Intel StrongArm SA-110. Starting from this base architecture, a design space for multimedia processors is defined by altering the following properties of the base processor architecture:

- The issue width
- The number of ALUs
- The number of branch units
- The number of memory units
- The size of the instruction cache
- The size of the data cache

Thus, a given architecture is specified by a six-tuple, with each value specifying one of the above properties.

The approach can be summarized as follows: First, a set of possible solutions is determined. Then, this set is reduced as far as possible by applying constraints for the exploration. Finally, the remaining set is searched for an optimal solution.

The framework for the design space exploration consists of a retargetable compiler, instruction level simulators, a set of media applications, and an architectural component selection algorithm. The basic exploration process is sketched in figure 5-6, showing power optimization under area constraints. First, seven architectures are generated without considering memory and cache properties. In the next step, the complete benchmark suite, consisting of 21 applications, is compiled for all architectures using the retargetable compiler from the IMPACT tool suite [CMCW91]. The resulting executables are then simulated using the simulator from the tool suite to get the run times of the applications. Hereby, each benchmark is simulated with 25 different configurations of memory and cache. As there were already seven architectures to start with, each application is therefore simulated on 175 configurations. The run times generated by the simulation are then used to calculate power dissipation estimations for each benchmark and each configuration. After this preparation step has been completed, the area requirement of each architecture is calculated and the area constraints are applied, excluding certain elements from the set of possible solutions. First, all architectures, which do not satisfy the constraints are eliminated. From the remaining set of architectures, all those configurations are removed, which dissipate more power than another configuration for all benchmarks. An architecture, with this property is called a dominated architecture in the picture. Finally, a selection algorithm is applied, to form the multiprocessor system from several of the remaining architectures.

The selection algorithm selects a subset of the architectures in a way, that the geometric mean of power dissipation is minimized and the size of the subset is kept small. The approach taken here is divided into two steps. First, the size of the subset is determined by increasing it until the enhancement of power dissipation caused by the new subset member is less than a given threshold. Then, the actual members of the subset are determined using a branch-and-bound algorithm.

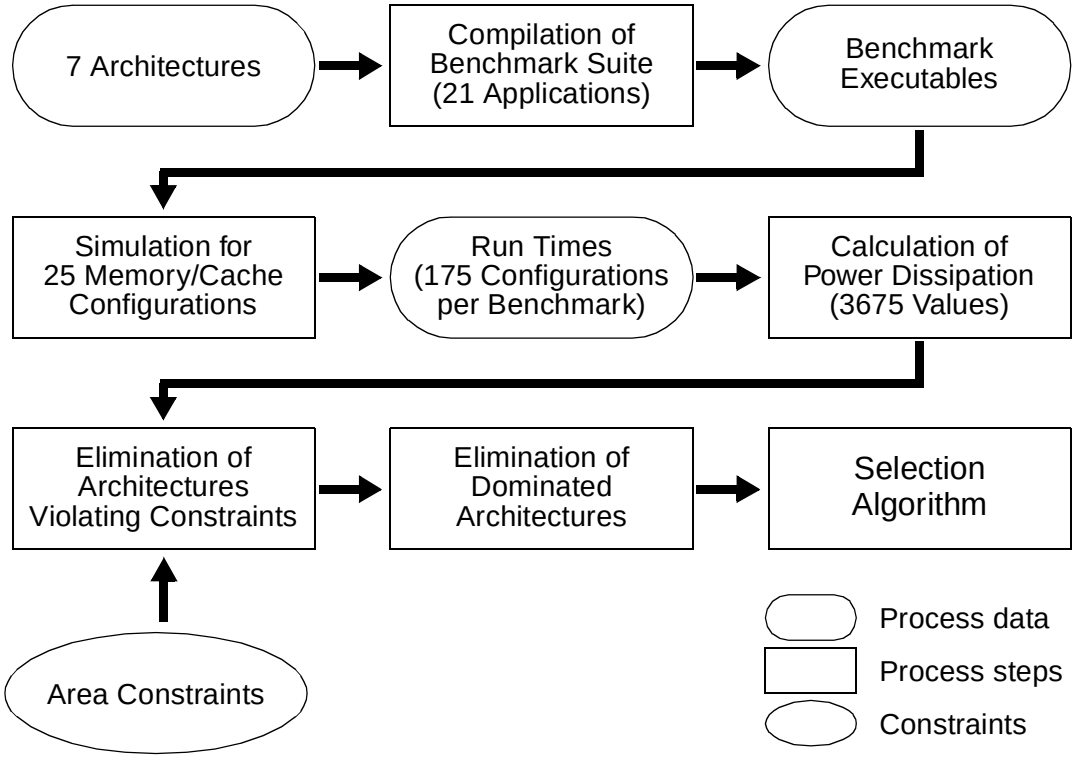

Figure 5-6: Design space exploration process for multimedia processors by Lee et al.