Development of a High-Level Design Space Exploration Methodology

Joachim Gerlach Wolfgang Rosenstiel

WSI 98-13

November 23, 1998

Wilhelm-Schickard-Institut
Universität Tübingen
D-72076 Tübingen, Germany

e-mail: gerlach@informatik.uni-tuebingen.de

ISSN 0946-3852

Research Project Report

Development of a High-Level Design Space Exploration Methodology

Joachim Gerlach
gerlach@informatik.uni-tuebingen.de

University of Tübingen
Department of Computer Engineering
Sand 13 • D-72 076 Tübingen • Germany

Wolfgang Rosenstiel
rosenstiel@informatik.uni-tuebingen.de

University of Tübingen
Department of Computer Engineering
Sand 13 • D-72 076 Tübingen • Germany
and
Forschungszentrum Informatik (FZI) Karlsruhe
Haid-und-Neu-Str. 10-14
D-76131 Karlsruhe • Germany

1 Motivation and Project Outline

Since a few years the increasing complexity of digital circuits represents the main problem in digital circuit design, existing methodologies which allow a specification of the design on higher levels of abstraction and therefore support a homogeneous design process are getting more and more important (*system design automation*). Catchwords in this domain are hardware software codesign, rapid prototyping, and embedded systems. But the increasing level of abstraction (which can be thought of as a vertical axis) also leads to an expansion of design space, and for this to an increase of optimization potential (imaginable as a horizontal axis) available on that level of abstraction (figure 1). Therefore, besides of the design process itself, the process of *design space exploration* should also start on upper design levels.

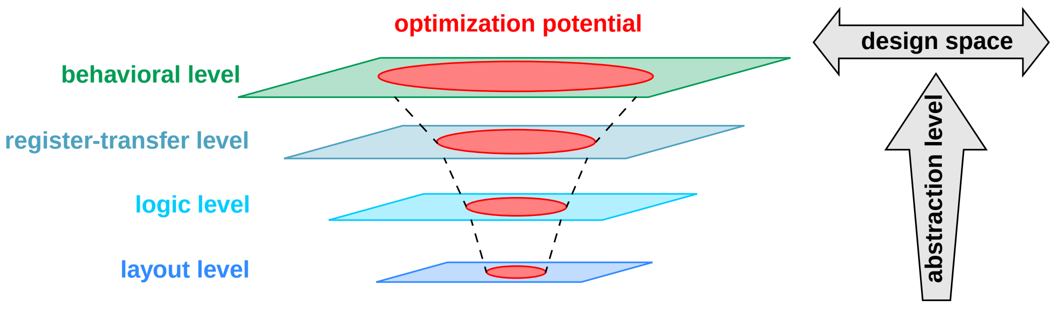


Figure 1. Abstraction level and optimization potential.

In our methodology, design space exploration is done on *behavioral level* by applying optimization steps called *high-level transformations*. If, like most of the conventional tools using high level transformations do, an evaluation of designs is performed by explicitly executing many (or even all) design steps of lower levels, the resulting design loop covers a large number of design activities and therefore a single cycle of the design loop is quite expensive to perform (referred to as *synthesis design loop* in figure 2). Our approach for realizing a design loop which is close and settled on high levels of abstraction bases on the integration of a high-level cost estimation step into the design loop (called *estimation design loop* in figure 2). Therefore, an evaluation of transformation quality is already done *within* behavioral synthesis. Figure 2 shows the integration of our methodology into the classical design level view.

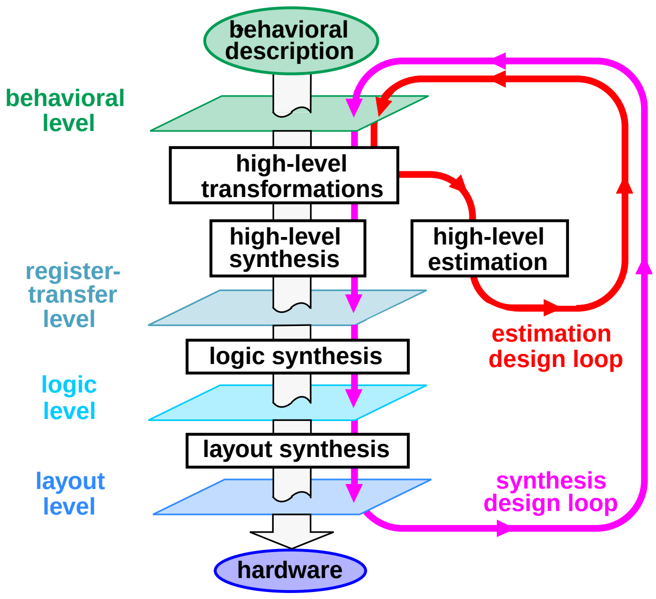


Figure 2. Design space exploration methodology.

Figure 3 gives an overview on the entire design environment developed at the University of Tübingen [GeEiHa96] and shows the integration of the high-level transformation task in the design flow. Starting point of our investigations is a specification of a system in terms of ANSI-C code, outcome is a register-transfer level implementation of the design parts identified for hardware realization. There exist several backends which allow to pass the register-transfer level design to several commercial and non-commercial lower-level synthesis tools. Regarding the entire design process, the high-level transformation task can be understood as a successor instance of hardware/software codesign and a predecessor instance of high-level synthesis.

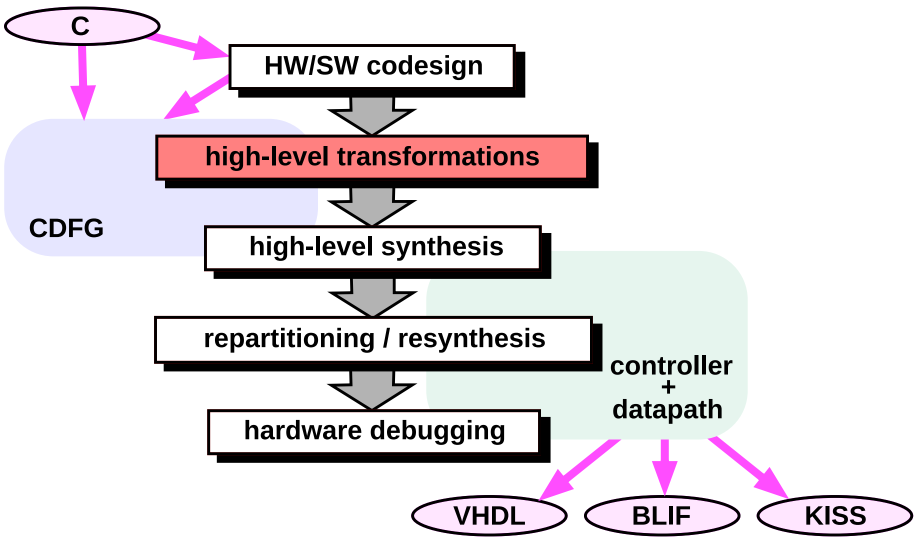


Figure 3. Design environment and interfaces.

Within a project initiated by Mentor Graphics Inc.™, in which several american and european universities are involved, the applicability of our transformation methodology to the commercial design system Monet™ was proved.[1] The Monet™ system corresponds to a high-level synthesis tool, which comfortably supports an interactive variation of resource and timing restrictions, and for this, allows to (manually) perform an architectural exploration of the design space. The first project period includes conceptual work in the domain of transformation analysis (phase-1) as well as an elaboration of the ability of or requirements for Monet™ to support our high-level transformation and estimation tasks (phase-2). This includes the implementation of an experimental format converter, which allows us to pass several designs from Monet™ to our transformation environment. In addition, an experimental evaluation of the Monet™ system was done (phase-3). Table 1 shows the project plan of the first project period.

project period 1	
contents	time schedule
phase-1: • realization of several cost heuristics • integration into prototype system • examination of combinations/parameterizations of cost heuristics • derivation of efficient cost estimation mechanisms • experimental validation of the cost estimation mechanisms	8 / 98
phase-2: • implementation of experimental format converter from Monet™ to prototype system • elaboration of requirements for Monet™-internal design representation to support high-level transformation and estimation tasks	12 / 98
phase-3: • experimental evaluation of Monet™	12 / 98

Table 1: Project plan for project period one.

The report is organized as follows: Chapter 2, 3, and 4 describe the work spend and the results achieved in phase-1, phase-2, and phase-3 of the first project period. Based on those results, chapter 5 proposes an outline of a project plan for a second project period.

2 High-Level Transformations and Transformation-Analysis

Figure 4 shows an abstract view on our transformation environment, in which three activities can be identified on highest level of abstraction:

- Application of transformations (*transformations*).
- Evaluation of transformation quality, including the step of cost estimation (*transformation analysis*).
- Control of the transformational exploration process, including the identification of transformation candidates and initiation of exploration cycles (*transformation control*).

1. Mentor Graphics Inc.™ and Monet™ are registered trademarks.

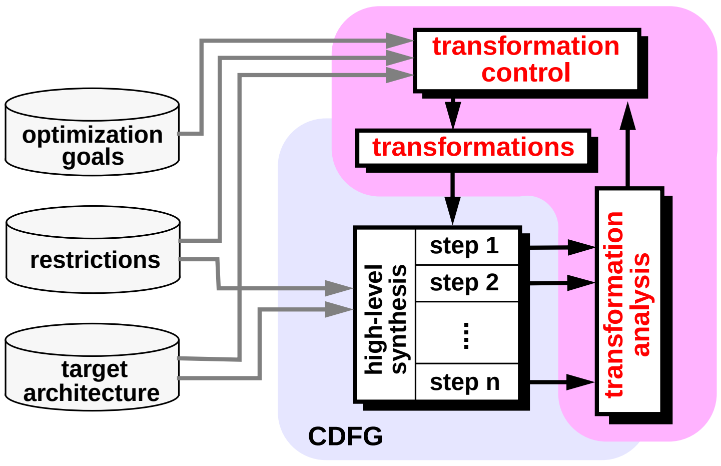


Figure 4. Transformation environment.

2.1 High-Level Transformations

The term *high-level transformations* covers optimization techniques settled on behavioral level. High-level transformations often (but not always) correspond to optimization-steps known from the theory of software compilers [AhSeUl86]. However, the preconditions of their applicability (e.g., target architecture, optimization goals, restrictions) significantly differ from software domain. In the current state of implementation, the transformation task comprises a set of thirty-four basic transformation types (see table 2). All transformation types are scalable in terms of their application (for example, application to particular circuit nodes, basic blocks, designs or entire design hierarchies). In addition, most of the transformation types are scalable in terms of their functionality (for examples, the transformation type *loop unrolling* covers the complete unrolling of a loop as well as the partial unrolling of a loop by a given unrolling factor, often referred to as *loop tiling* in the literature).

array contraction	elimination of empty paths
array scalarization	elimination of unreachable paths
optimization of memory accesses	normalization of branch conditions
constant propagation	manipulation of branch conditions
constant folding	merging of branch constructs
common subexpression elimination	normalization of loop index variables
common subexpression expansion	extraction of loop-invariant basic-blocks
elimination of intermediate data	loop-invariant code motion
optimization of boolean expressions	loop unrolling
optimization of comparator calculations	loop partitioning
algebraic optimizations	loop merging
strength reduction	loop reversal
word size reduction	loop interchange
optmimization of swap-scenarios	loop pipelining
basic-block partitioning	function inlining
basic-block merging	function cloning
basic-block shifting	elimination of unreachable functions

Table 2: Implemented high-level transformation types.

Our approach for realizing a design loop which is close and settled on high level of abstraction bases on a tight coupling of high-level transformation and high-level synthesis tasks. Therefore, high-level transformations immediately act on the internal high-level synthesis design representation, an attributed control-/dataflow graph [HoEiHa94]. Evaluation of transformation quality is done by trial execution of particular steps of the high-level synthesis process, followed by an analysis of the resulting intermediate synthesis result. At this point, heuristics come into view, which allow to estimate design characteristics starting from a high level of abstraction, and for this, to cover expensive design steps of lower design levels. By this, proof of transformation applicability, transformation application itself, and analysis of the transformation result (including the step of cost estimation) is completely performed on the high-level design representation. For this, a complete cycle of the design loop can be executed efficiently without leaving the high-level design representation. For derivation of efficient cost evaluation heuristics, several estimation methodologies fixed on different states of the high-level synthesis process are examined with respect to their aptitude on controlling the transformational exploration process. Based on those examinations, efficient evaluation heuristics for several design criteria are derived and experimentally validated. In the following, those investigations are reported in detail.

2.2 Transformation-Analysis

The application of a cost estimation heuristic within our transformational design space exploration methodology leads to a set of specific demands:

- To allow cost evaluation within high-level synthesis, the execution stage of the synthesis process has to become a parameter of the cost estimation heuristic. The cost estimation heuristic should be applicable to different execution states of the high-level synthesis process (and for this, is called *scalable*) and should be able to identify and utilize all design information which is available in this state of synthesis.
- In spite of the high level of abstraction, on which cost estimation acts, the heuristic has to be able to estimate design costs in a fine-grain fashion, because designs resulting from high-level transformations possibly differ in a very slight way only (e.g., because a transformation only affects very local parts of the design).
- In regard of applying the cost estimation heuristic within transformational design space exploration, the ability to quantify *relative* dependencies of design characteristics (and for this, to *compare* design alternatives) is much more important than the ability to capture *absolute* values. To quantify this characteristic, we apply the *fidelity* measure [GaVaNa94] proposed by Gajski:

$$Fidelity = 100 \bullet \frac{2}{n(n-1)} \bullet \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} \mu_{ij} \qquad \mu_{ij} = \begin{cases} 1 \text{ if } \begin{cases} R(i) < R(j) \wedge E(i) < E(j) \\ R(i) > R(j) \wedge E(i) > E(j) \\ R(i) = R(j) \wedge E(i) = E(j) \end{cases} \\ 0 \quad \text{otherwise} \end{cases}$$

  The fidelity measure supplies for a given set of *n* reference values *R(1),...,R(n)* and *n* estimation values *E(1),...,E(n)* a number describing the quality of the estimation with respect to its ability to quantify relative dependencies of pairs of reference/estimation values.

Our cost estimation methodology bases on the common application of two contrary principles, *specialization* and *abstraction*. Therefore, two levels of estimation heuristics have to be distinguished:

- Techniques to estimate the costs of the entire design, referred to as *level-1 cost heuristics*.
- Techniques to estimate the costs of isolated register-transfer component cells, referred to as *level-0 cost heuristics*.

Figure 5 shows an overview on our cost estimation methodology. In a level-1 cost heuristic, as much as possible of register-transfer information (depending on the execution state of the synthesis process) is identified (components as well as interconnection structure). This corresponds to the specialization step. The identified register-transfer components are passed to a level-0 cost heuristic, which performs an estimation of the particular component cell costs. The cost values are propagated back into the level-0 cost heuristic and used there for the calculation of an entire estimation value, corresponding to the abstraction step.

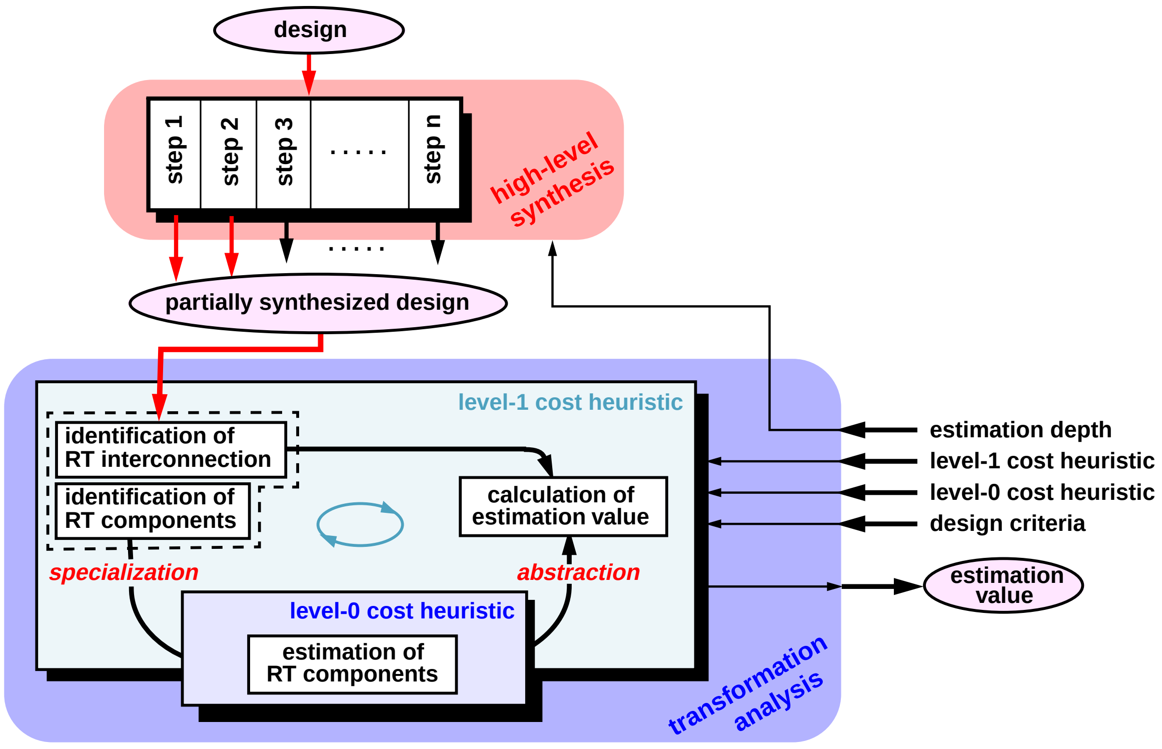


Figure 5. Evaluation of designs.

Objective of this methodology is to utilize all design information for calculation of an estimation value, which is available in the current state of the synthesis process. In our investigations, level-1 cost heuristics are applied to five execution stages of the high-level synthesis process, referred to as *estimation depths* (see table 3).

estimation depth	state of high-level synthesis
1	after module scheduling and allocation
2	after module binding
3	after register allocation and binding
4	after interconnection binding
5	after netlist generation

Table 3: Evaluation depths of level-1 cost heuristics.

In estimation depth 5, high-level synthesis is carried out completely, and for this depth 5 is equivalent to register-transfer level. By dividing the estimation step in level-1 and level-0 cost heuristics, the estimation methodology becomes „high-level“: generation of level-0 cost heuristics has to be done only once for a given target architecture, design style and high-level synthesis process (and for this, can be regarded as a precomputation step). A single estimation request only requires the evaluation of level-1 cost heuristics. In our investigations, several level-1 and level-0 cost heuristics were realized. Combinations of level-1 cost heuristics, level-0 cost heuristics, and estimation depths were experimentally extracted which lead to evaluation heuristics that can be efficiently applied in our transformational design methodology. In the following, the implemented level-0 and level-1 cost heuristics are presented in detail, and the results of some experiments are summarized.

2.2.1 Level-0 cost heuristics

Level-0 cost heuristics base on an explicite determination of costs of elementary register-transfer cells (of the generic component-library of the high-level synthesis process, which is actually in use). Typical examples of such elementary cells are full adders, logic-gates or 1-bit 2:1-multiplexors. Those elementary cells are explicitly synthesized and design characteristics are extracted. By this, characteristics of the applied synthesis tools and target architecture are taken into consideration.

A first level-0 cost heuristic, *GenCost*, determines the costs of complex register-transfer components by cost formulas. Those cost formulas are derived by analyzing the generic generators of the high-level synthesis process, which is actually in use. Arguments of the cost formulas are given by parameters of the generic generators, for example, word length, number of inputs or sign status. Figure 6 shows an example: for deriving area and delay cost formulas of a ripple-carry adder, $area_{FA}$ and $delay_{FA}$ of an elementary full adder cell are explicitly determined (applying the concrete synthesis process). Regarding the generic generator of an arbitrary *n*-bit adder, it can be found that area can be approximated by $n \bullet area_{FA}$. Same formula holds for delay, taking into account that the critical path touches all full adder cells.

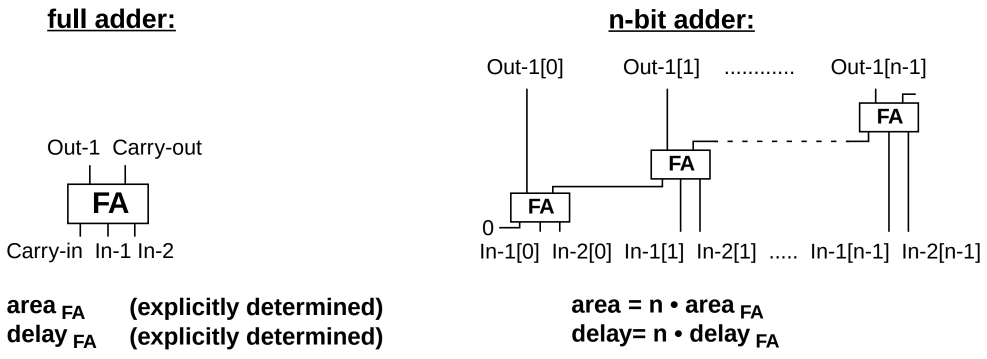


Figure 6. Level-0 cost heuristic *GenCost*.

In a second level-0 cost heuristic, *FuncCost*, costs of particular instances of a register-transfer components are explicitly determined, too, but in contrast to *GenCost*, not only elementary cells of the parameter space are regarded. The evaluated cost values are used to generate a piecewise linear approximating function, which is applied for a cost function of the corresponding register-transfer component type. For example, figure 7 shows in (a) the delay cost function of dimension one of a adder cell (parameter is word length) and in (b) the delay cost function of dimension two of a multiplexor component (parameters are word length and number of inputs).

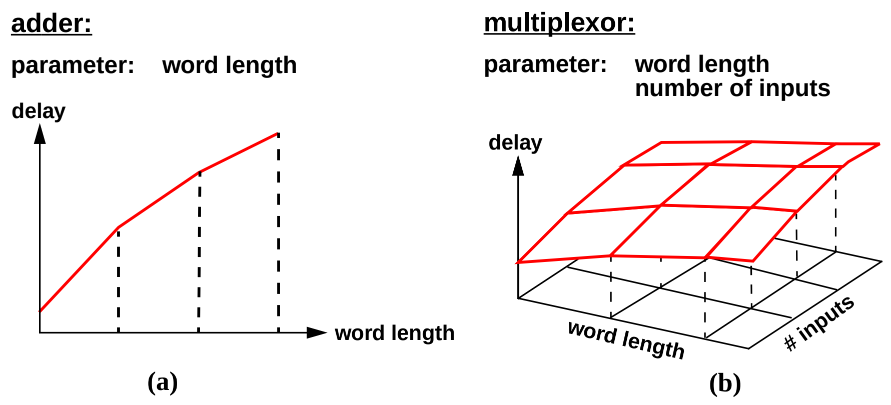
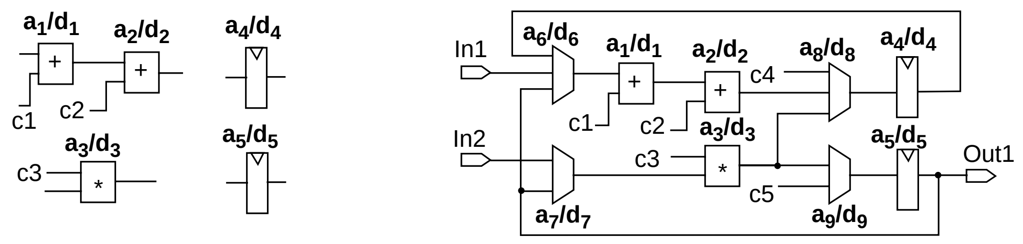


Figure 7. Level-0 cost heuristic *FuncCost*.

2.2.2 Level-1 cost heuristics

Level-1 cost heuristics identify as much as possible of register-transfer information in the given design (which corresponds to an intermediate result of the high-level synthesis process). Then, level-0 cost heuristics are applied to perform estimates of the isolated register-transfer components and the results are used for calculation of an estimation value for the entire design.

A first level-1 cost heuristic, *DPApprox*, analyzes the current design representation and identifies connected datapath segments of maximum size. By this strategy, all information which is available in the current state of high-level synthesis is utilized to perform an estimation value. Figure 8 shows an example: In (a), only information concerning functional units and registers is available. In (b), estimation is performed in a more advanced stage of the synthesis process, which results in more accurate predictions.



(a)

$$\text{area} = a_1 + a_2 + a_3 + a_4 + a_5$$

$$\text{delay} = \max \left\{ \begin{matrix} d_1 + d_2 \\ d_3 \end{matrix} \right\} + \max \left\{ \begin{matrix} d_4 \\ d_5 \end{matrix} \right\}$$

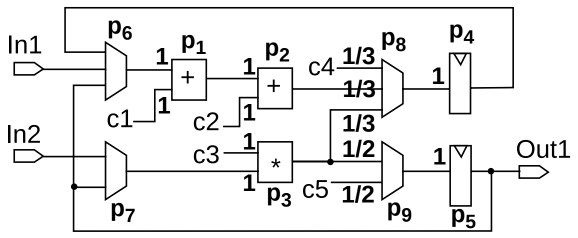
(b)

$$\text{area} = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8 + a_9$$

$$\text{delay} = \max \left\{ \begin{matrix} d_6 + d_1 + d_2 + d_8 + d_4 \\ d_7 + d_3 + d_8 + d_4 \\ d_7 + d_3 + d_9 + d_5 \end{matrix} \right\}$$

Figure 8. Cost heuristic *DPApprox*.

A second level-1 cost heuristic, *ProbDPApprox*, acts in very similar fashion, but in addition, a weighting of datapath segments (or sub-segments) corresponding to the maximum interconnection structure which can be identified in the current state of the design process, is performed. Thereby, a uniform distribution is assumed in branching nodes, which leads, as the experiments will show, to satisfactory results. Figure 9 shows the example design of figure 8 (b) with weighting factors given at each circuit arc.



$$
\begin{aligned}
\text{power} = \; & 1 \cdot (p_4 + p_8) + 1/3 \cdot (0) \\
& + 1/3 \cdot (p_2 + p_1 + p_6) \\
& + 1/3 \cdot (p_3 + p_7) \\
+ \; & 1 \cdot (p_5 + p_9) + 1/2 \cdot (p_3 + p_7) \\
& + 1/2 \cdot (0)
\end{aligned}
$$

Figure 9. Cost heuristic *ProbDPApprox*.

A third cost heuristic, *CPApprox*, considers costs produced in the controller part of the design. This is done by regarding several static characteristics of the controller (e.g., number of states, transitions, input/output lines). Applying a set of standard high-level synthesis benchmark designs, we were able to show a good correlation (fidelity > 80%) of an estimation based on such static characteristics. For example, figure 10 shows the correlation of the absolute values

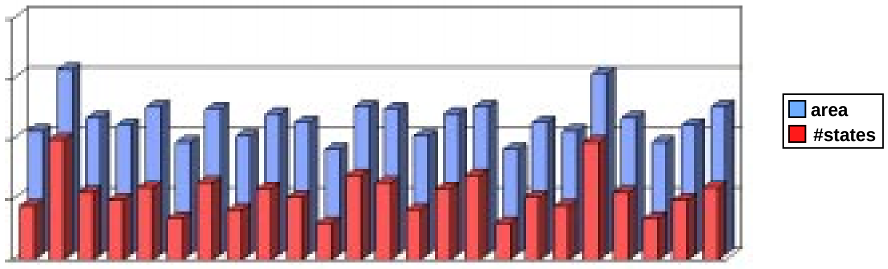


Figure 10. Cost heuristic *CPApprox*.

for the number of states of the finite state machine and the area consumption of the controller part for some benchmark designs. For a common consideration of datapath and controller, correlation factors describing the ratio of datapath costs (heuristics DPApprox and DPApprox) to controller costs (heuristic CPApprox) are derived in a statistical way. For more details see [GeRo97].

2.2.3 Experiments

Objective of our investigations is to derive combinations of level-0 cost heuristics, level-1 cost heuristics and estimation depths which result in estimation mechanisms of maximum quality/effort trade-off with respect to their application in transformation control. In the following, the design criteria area, delay, and power are considered in detail. Other design criteria (e.g., response-time, FSM size) were regarded, too, but the methodologies differ from those presented above, and for this, are not regarded here. To evaluate the quality of our estimation methodology, heuristics resulting from several combinations of level-0 cost heuristics, level-1 cost heuristics, and estimation depths were applied to multiple applications, and the results were compared with the synthesis results. In our experiments, high-level synthesis is performed by the PMOSS system [GeEiHa96], logic synthesis is done by the SIS system, and mapping the design to a concrete target architecture bases on the MCNC library [SeSiLa92].

In the following, our methodology is demonstrated in terms of the following three types of applications:

- *Fidelity*: The methodology is demonstrated in terms of the algorithm for computing the fidelity measure. For this, the evaluation algorithm itself becomes an application.

- *High-level transformation benchmark suite*: The heuristics derived from the fidelity example are validated via a set of high-level transformation benchmark designs.
- *GSM*: For an application of high complexity and industrial relevance, a submodule of the GSM fullrate speech transcoder [DIN94], integrated in mobile telecommunication systems for real-time compaction of human speech, is regarded.

Application-1: **Fidelity Algorithm**

The fidelity algorithm was transformed into a set of nine alternative design versions by applying high-level transformations (for the transformations sequence, see figure 11). In figure 12,

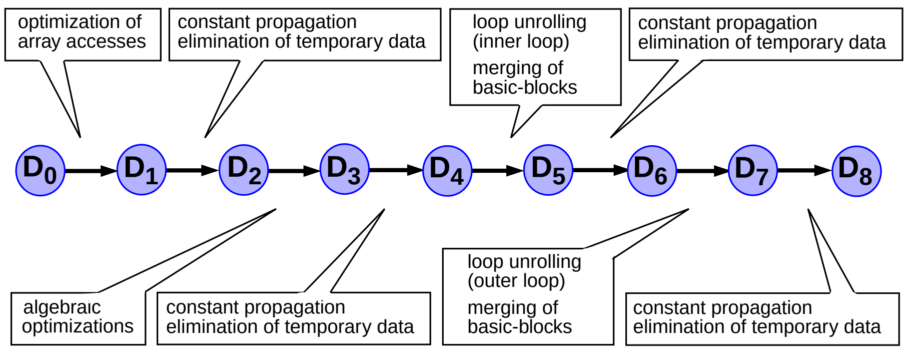
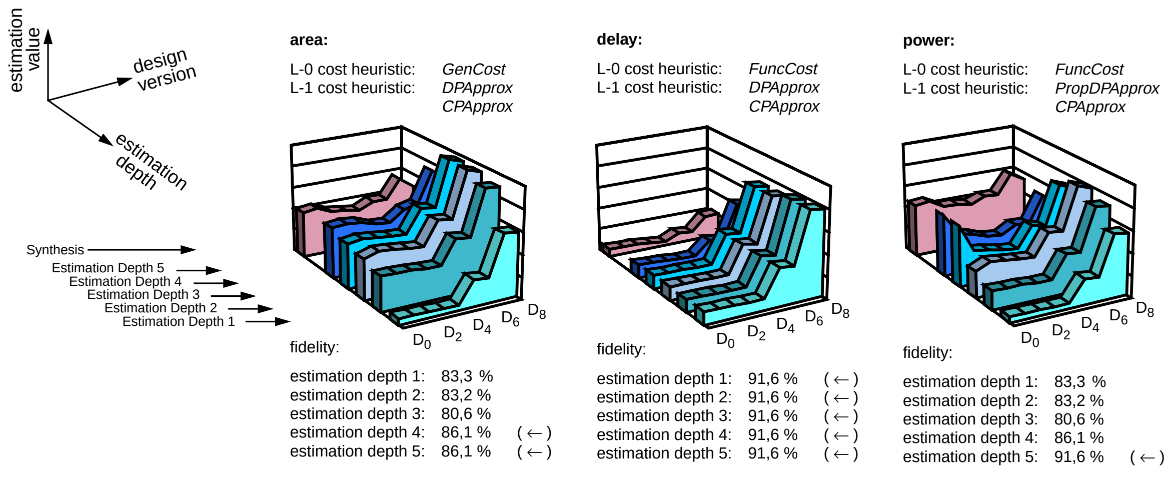

optimization of array accesses
constant propagation elimination of temporary data
loop unrolling (inner loop) merging of basic-blocks
constant propagation elimination of temporary data
D0
D1
D2
D3
D4
D5
D6
D7
D8
algebraic optimizations
constant propagation elimination of temporary data
loop unrolling (outer loop) merging of basic-blocks
constant propagation elimination of temporary data

Figure 11. Transformation sequence for the fidelity application.

combinations of level-0 and level-1 cost heuristics, which lead to estimation mechanisms with maximum fidelity (marked by „←" in figure 12) are shown. Like mentioned before, the *fidelity* of estimation heuristics is of primary interest, and for this, the *shape* of the curves, not the absolute values has to be considered (the absolute values generated by the heuristics don't correspond to any physical unit, only the relative relations are of interest). As figure 12 indicates, e.g. combination of the generic level-0 cost heuristic *GenCost* and level-1 cost heuristics *DPApprox* (non-probabilistic datapath approximation) and CPApprox (controller approximation) results starting with estimation depth 4 in area estimation mechanisms of maximum fidelity of 86%. Regarding the speedup applying our high-level estimation methodology with respect to generating corresponding values by explicitly performing lower level synthesis steps, speedup factors from $0.5 \bullet 10^2$ up to $1.1 \bullet 10^6$ can be obtained (depending on design criteria and estimation depth). The results are summarized in table 4. For details see [GeRo97].



area:

L-0 cost heuristic:	*GenCost*
L-1 cost heuristic:	*DPApprox* *CPApprox*

fidelity:

estimation depth 1:	83,3 %	
estimation depth 2:	83,2 %	
estimation depth 3:	80,6 %	
estimation depth 4:	86,1 %	(←)
estimation depth 5:	86,1 %	(←)

delay:

L-0 cost heuristic:	*FuncCost*
L-1 cost heuristic:	*DPApprox* *CPApprox*

fidelity:

estimation depth 1:	91,6 %	(←)
estimation depth 2:	91,6 %	(←)
estimation depth 3:	91,6 %	(←)
estimation depth 4:	91,6 %	(←)
estimation depth 5:	91,6 %	(←)

power:

L-0 cost heuristic:	*FuncCost*
L-1 cost heuristic:	*PropDPApprox* *CPApprox*

fidelity:

estimation depth 1:	83,3 %	
estimation depth 2:	83,2 %	
estimation depth 3:	80,6 %	
estimation depth 4:	86,1 %	
estimation depth 5:	91,6 %	(←)

Figure 12. Comparison of estimation and synthesis results.

Application-2: **High-Level Transformation Benchmark Suite**

For validation of the estimation heuristics derived in the previous experiment, alternative design versions of ten benchmark designs (of different applications, for example, filters, sorting algorithms, mathematical computations, DSPs) were generated via high-level transformations. Then, the design criteria area, delay, and power were estimated applying the heuristics extracted in the fidelity example. The resulting fidelity values are summarized in table 4. All speedup factors have similar magnitudes as the ones presented in the fidelity example.

Application-3: **GSM Fullrate Speech Transcoder**

In scope of a project under grant of Deutsche Forschungsgemeinschaft, the GSM fullrate speech transcoder was treated by a hardware/software codesign step, wherein a real-time critical module was identified and efficiently realized in hardware [ScFeMo97]. This module was optimized applying our transformational design space exploration methodology. Figure 13

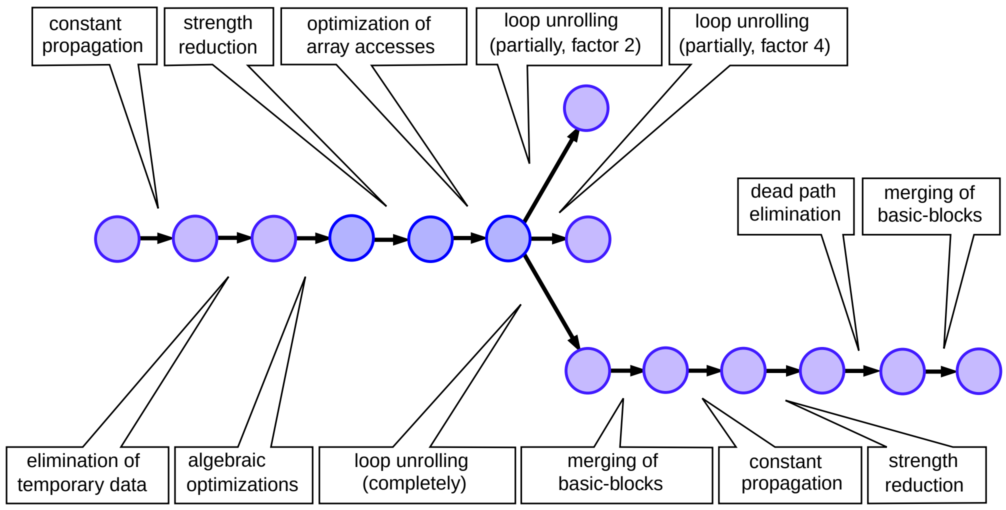


Figure 13. Transformation tree for the GSM application.

shows the corresponding transformation tree. Here, the estimation heuristics derived in the fidelity application also decide the quality/effort trade-off in an optimal way. The resulting fidelity values are summarized in table 4, the speedup disposes of similar dimensions as the ones given in the fidelity example. By rerunning the design space exploration process applying the synthesis loop instead of the estimation loop (see figure 2), the identical design version was identified for cost minimum design. Thus we were able to show that in the GSM application our estimation methodology leads to an acceleration of the design process by orders of magnitudes without any loss of design quality.

Table 4 summarizes the results of the experiments described above and for this, suggests estimation mechanisms for several design characteristics. In all applications, the proposed combinations of level-0 and level-1 cost heuristics and estimation depths lead to rapid estimation heuristics of high fidelity (≥ 80 %), which points to a high stability of our methodology. For all applications and design characteristics, our estimation heuristics were able to reliably forecast the design version with minimum cost. All speedup factors have similar magnitudes as the ones presented in the fidelity example, and for this lead to a significant acceleration of the design space exploration process.

design criteria	estimation heuristic						fidelity result											
	level-0 cost heuristic		level-1 cost heuristic															
	GenCost	*FuncCost*	*DPApprox*	*DPProbApprox*	*CPApprox*	estimation depth	fidelity (in %)	trans-bench-1 (in %)	trans-bench-2 (in %)	trans-bench-3 (in %)	trans-bench-4 (in %)	trans-bench-5 (in %)	trans-bench-6 (in %)	trans-bench-7 (in %)	trans-bench-8 (in %)	trans-bench-9 (in %)	trans-bench-10 (in %)	GSM (in %)
area	X		X		X	4	86.1	97.2	80.0	100	82.2	86.7	89.3	91.7	93.9	93.3	97.2	89.1
delay		X	X		X	1	91.7	86.1	90.0	100	86.1	88.9	96.4	91.7	86.4	100	91.7	81.8
power		X		X	X	5	91.6	83.3	80.0	89.3	80.1	88.9	96.4	94.4	89.4	93.3	91.7	synthesis timeout

Table 4: Results of estimation heuristic evaluation.

3 Elaboration of the Monet™-Internal Design Representation

Objective of the second phase of project period one is the elaboration of the ability of or requirements for the Monet™-internal design representation to support high-level transformation and estimation tasks in the way presented in chapter 2. This includes the implementation of an experimental format converter, which allows to pass several Monet™ designs into our prototype system and there to perform transformation and cost evaluation steps (figure 14). This enables us to take a look at Monet™'s internal design representation, and evaluate, if it's practicable and ingenious to, for example, *connect* our prototye system to Monet™ (to perform the transformation and estimation task within the prototype system), or to *re-implement* high-level transformation and high-level estimation concepts in Monet™ (to disconnect Monet™ from the prototype system).

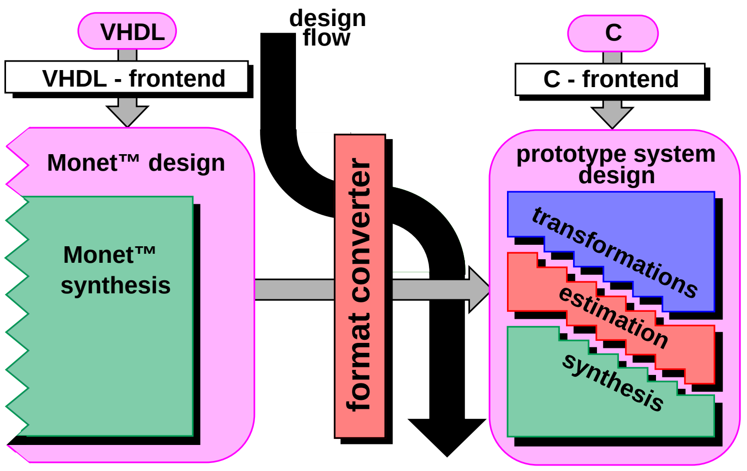


Figure 14. Format converter and design flow.

First of all, a brief overview on the internal design representation in Monet™ and in our prototype system is given:

- Monet™ design activities base on an internal design representation called *SIF* (which stands for *Synthesis Internal Form*). Intention of the SIF design representation is to define an intermediate synthesis format which allows to manage multiple synthesis methodologies, which is independent of hardware description language, and which is able to handle multiple levels of abstraction (from system downto logic level). SIF is composed of a set of (about two-hundred) object-oriented C++ classes. The SIF synthesis database is the language synthesis data repository which uses SIF objects (C++ object instances of SIF classes) as its internal design representation. By this, SIF covers a collection of graph-based synthesis oriented object models, for example a control-/dataflow graph, finite state machine, and netlist representation. In addition, SIF allows to specify timing and structure constraints. There exists a frontend, which allows to convert VHDL design descriptions into SIF, as well as a backend which allows to generate a VHDL output out of a synthesized SIF structure. VHDL generation can be applied to different execution stages of the high-level synthesis process. Details concerning the SIF design representation can be found in [Wu97].

- In our prototype system, design representation bases on an attributed control-/dataflow graph (*CDFG*), which is annotated with design information during the synthesis process step by step. The graph representation bases on the object-oriented C++ class library LEDA [MeNaSe98], which provides a collection of elementary data types (for example sets, lists, graphs, dictionaries) and member functions for data manipulation. There exists a frontend, which allows to convert ANSI-C specifications into the internal CDFG representation. There also exist several backends which allows to convert the annotated CDFG structure into register-transfer level representations (VHDL, BLIF, KISS) to be passed into several commercial and non-commercial lower level synthesis tools. Details concerning the CDFG design representation can be found in [HoEiHa94].

3.1 Realization strategies for format conversion

Figure 15 shows three realization strategies for connecting Monet™ and prototype system. The alternatives were worked out in cooperation with the Monet™ developer team. Strategy-1 and -2 uses an existing backend, which converts the SIF-representation into a textual form. Based on those textual SIF-file, strategy-1 plans to implement a complete SIF-parser plus CDFG-generator. Strategy-2 also bases on the textual SIF-file, but takes advantage of an existing lex/yacc parser for SIF. In this approach, callback procedures have to be added to generate a CDFG-structure. Strategy-3 does not base on a textual SIF-file, but takes advantage of an existing procedural SIF-interface. In this solution, an analysis of the SIF-structure can be done directly by running through the hierarchy of SIF class instances. In table 5, some characteris-

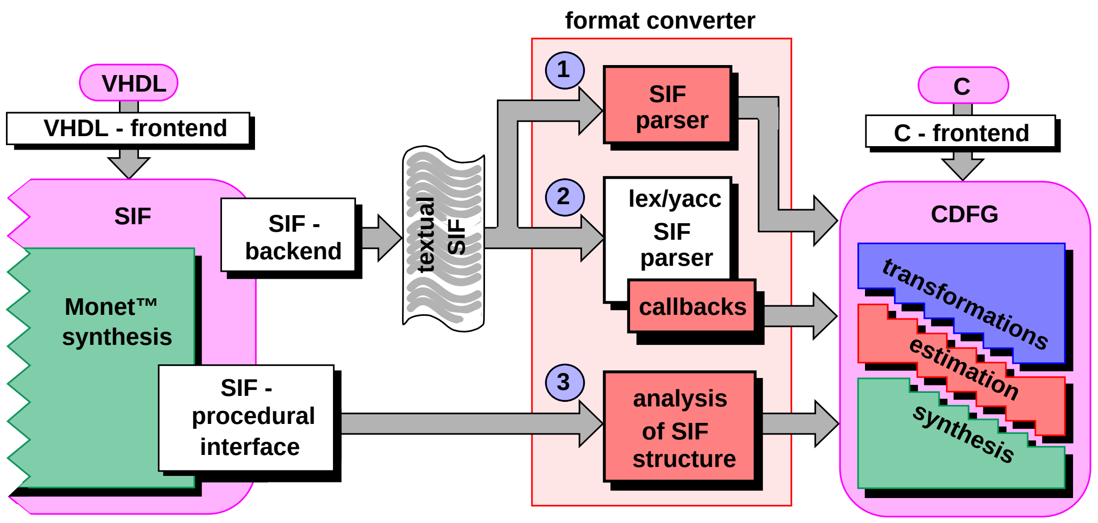


Figure 15. Realization strategies for SIF to CDFG format conversion.

tics of the realization alternatives are summarized. Since strategy-2, the extension of an existing lex/yacc parser, seemed to be optimally (because of the medium realization effort for writing callback procedures and the weak coupling of the systems via a textual file), our first investigations were focussed to this solution. Main drawback of this strategy turned out to be the fact, that there is no documentation concerning the textual-SIF format available up to now. Because of this, we have to concentrate on strategy-3, the procedural interface solution. At this point, the term procedural interface covers access to the C++ hierarchy of class instances presenting the SIF structure.

strategy	realization effort	documentation	strength of coupling
① textual-SIF parser	high	not available	weak
② lex/yacc textual-SIF parser	medium	not available	weak
③ procedural SIF interface	high	restricted available	tight

Table 5: Evaluation of realization strategies.

Main drawback of strategy-3 is, that the coupling of Monet™ and prototype system is very tight (an analysis of the hierarchy of SIF class instances is done directly via member functions of the SIF C++ classes). Because of the fact, that there exist no documentation concerning the class interfaces, parts of the Monet™ source code has to be made available for analysis. Those parts are the SIF database modul plus the SIF generate module (assuming VHDL input) or the SIF database modul plus the SIF read module (assuming textual SIF input). In coordination with the Monet™ developer team, the later alternative was selected (see figure 16 for a brief overview on the Monet™ modules and their interaction).

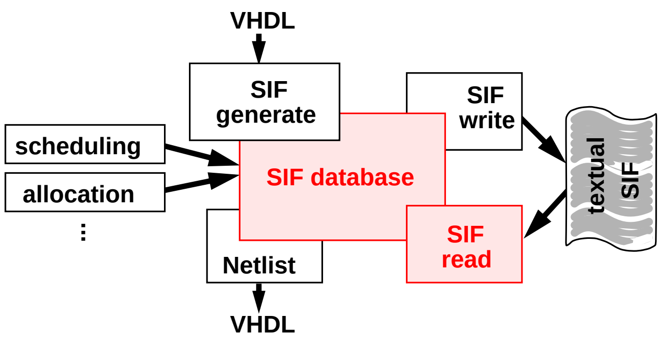


Figure 16. Monet™ modules and interaction.

3.2 SIF to CDFG format conversion

According to realization strategy-3 identified for most practicable in the previous section, the conversion of a SIF structure into a prototype system CDFG structure is done by

- running through a Monet™ designs hierarchy of SIF class instances,
- identifying SIF information which is relevant for CDFG generation, and
- creating of a corresponding CDFG structure.

In the following, details concerning the format converter and its implementation are given.

Objective of the format conversion from SIF to CDFG is the elaboration of requirements and methodologies to make high-level transformation and high-level estimation activities available in Monet™. The implemented format converter has an *experimental* character, which means that only a subset of the SIF expressive domain can be transformed into CDFG structures. Reasons for that restriction are:

- Input of SIF is a description of a hardware design given in a *hardware description language* (VHDL), since input of CDFG is a description of a hardware design given in a *software language* (C). Intention of SIF is to support the whole VHDL language, in the ideal case, since intention of CDFG is to examine approaches to use software language descriptions for the specification of hardware behavior. This concludes that SIF is, on principle, more suited for describing hardware, because CDFG is focussed on coming out of a software language (and for this, is restricted in some features to the expressiveness of software language descriptions). With respect to the format conversion, this means, that only those SIF designs can be successfully converted to CDFG, which fall into the expressive range of CDFG (see figure 17).
- An important feature of SIF is the support of multiple levels of abstraction. According to the methodology presented in chapter 1, high-level transformations are settled on behavioral level. Because of this, a conversion of SIF designs into a CDFG structure is limited to upper (behavioral) level SIF descriptions (see also figure 17). In particular, synthesis information generated by Monet™ cannot be adapted to the CDFG structure.

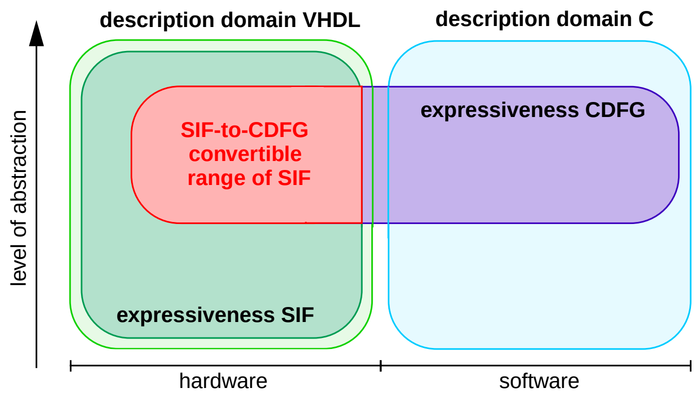
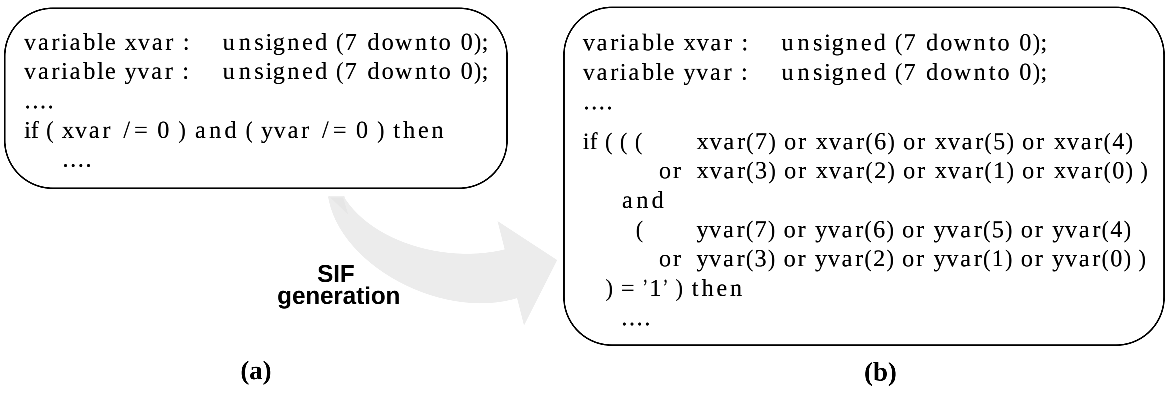


Figure 17. Description domains and expressiveness.

- In regard of the fact, that a complete documentation of the SIF structure does not exist up to now (the existing SIF documentation unfortunately gives only a superficial and incomplete overview on SIF in terms of a set of examples), informations concerning SIF have to be generated by inspection of source code and by analysis of benchmark designs. For benchmark designs, we orientate on the standard examples which were part of the Monet™ package and a set of inhouse high-level benchmark designs of different size, complexity, and application domain.
- In Monet™, some conversion steps for optimization (for examples, propagation of constant values) and for representation simplification (for example, representation of the *≠0* operator, see below) are *inseparably* coupled to the SIF generation process. The problematic nature of those built-in conversions is shown in terms of the *≠0* operator: Figure 18 shows in (a) a segment of VHDL code, which is transformed by the SIF generation process into an internal SIF representation corresponding to VHDL code segment (b). As figure 18 indicates, no „high-level“ representation of the *≠0* operator exists in SIF, but the *≠0* construct is

automatically mapped to a „low level“ sequence of bit operations. However, the high-level representation is much more suited for high-level transformation purpose (for example, application of algebraic simplifications), and for this, has to be reconstructed out of the low-level representation.[1]



```
variable xvar :    unsigned (7 downto 0);
variable yvar :    unsigned (7 downto 0);
....
if ( xvar  /= 0 ) and ( yvar  /= 0 ) then
    ....
```

```
variable xvar :    unsigned (7 downto 0);
variable yvar :    unsigned (7 downto 0);
....
if ( ( (       xvar(7) or xvar(6) or xvar(5) or xvar(4)
           or  xvar(3) or xvar(2) or xvar(1) or xvar(0) )
    and
      (        yvar(7) or yvar(6) or yvar(5) or yvar(4)
           or  yvar(3) or yvar(2) or yvar(1) or yvar(0) )
  ) = ’1’ ) then
    ....
```

Figure 18. Representation of ≠0 constructs in SIF.

For the implementation of an experimental SIF to CDFG format converter, the SIF database modul plus the SIF read module was isolated by the Monet™ developer team and made available in source code format to University of Tübingen. Figure 19 shows realization strategy-3 which was identified for implementation in section 3.1 (according to figure 15).

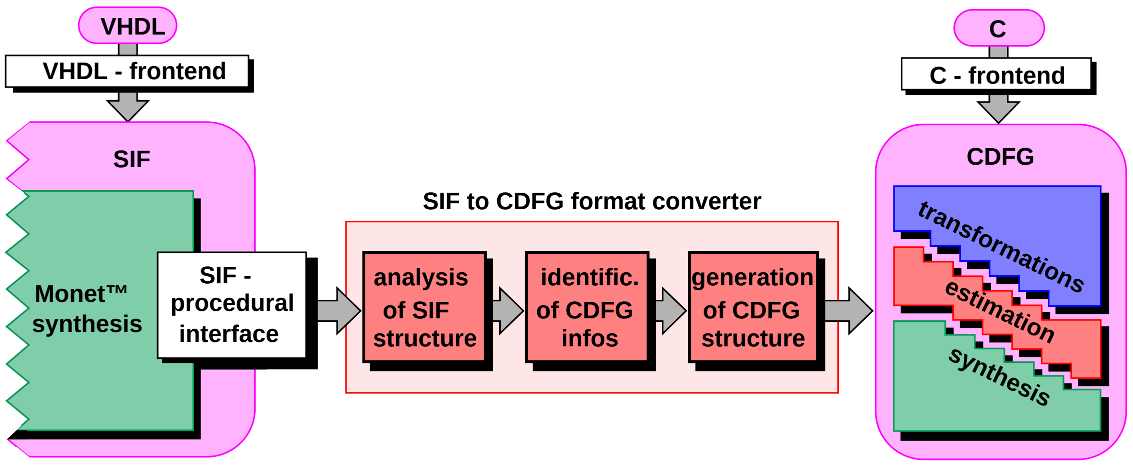


Figure 19. Realization strategy-3 for SIF to CDFG format conversion.

In figure 20, an overview on the implementation details are given:

- The availability of the SIF database and SIF read modules allows an integration of the procedural interface into the format converter. By this, the strength of coupling of Monet™ and format converter (which was found to be a drawback in section 3.1) can be decreased. Communication between Monet™ and format converter takes place via *textual SIF format* (in the same way like in realizations strategy-1 and -2 in figure 15), but the defacto conversion is performed by *directly* analyzing the hierarchy of SIF class instances via the procedural interface (according to realization strategy-3).

1. For the presented example of the ≠*0* construct, this reconstruction is performed by the SIF to CDFG format converter automatically, but leads to high conversion effort and is possibly not supported for other examples (because of the absence of a complete SIF documentation).

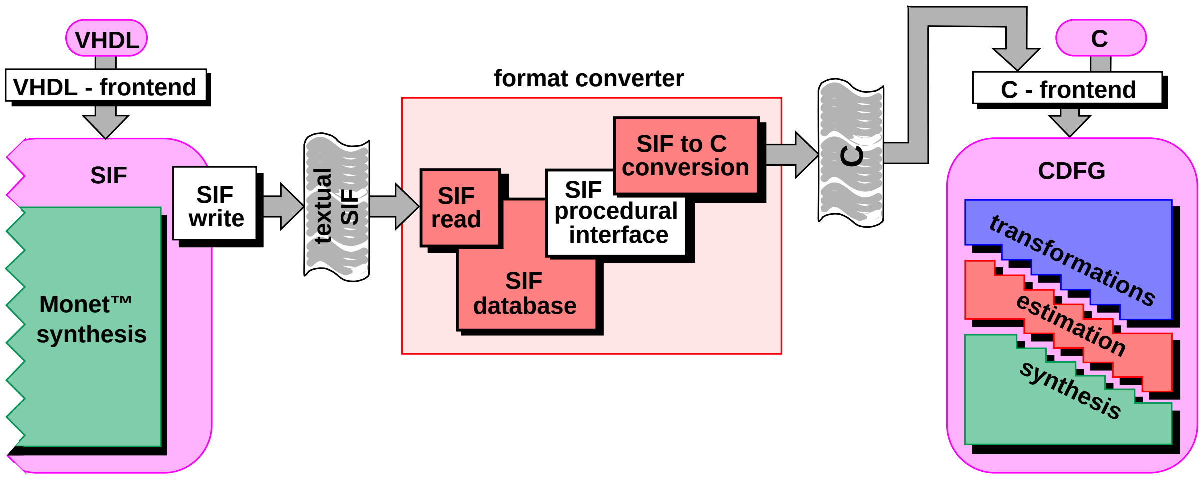


Figure 20. Implementation of realization strategy-3.

- Inside the converter, a generation of the CDFG structure is not done directly, but takes advantage of the existing ANSI-C frontend. Therefore, C-format is produced out of the SIF structure and passed to the prototype system C-frontend (to be translated into CDFG). The resulting C-code is extended by specific frontend commands which allows to pass hardware-related information (for example, word lengths or port attributes of signals and variables) into the prototype system CDFG structure. By this, the strength of coupling of format converter and prototype system can also be decreased. This results in a format converter, which acts for an isolated instance (binary) to transform textual SIF input into extended C-code output. In particular, the C output can be made available for the designer to be used for test and validation purpose (see section 3.3).

3.3 Experiments

Objective of the implementation of an experimental format converter was to pass several designs from Monet™ into the prototype system and there to perform high-level transformation and high-level estimation steps. The applicability of the methodology can be shown in terms of a set of high-level synthesis and high-level transformations benchmark designs. Table 6 shows some characteristics of the benchmark designs and the conversion process.

According to the methodology described in section 3.2, Monet™ was applied for creation of textual SIF format out of behavioral level VHDL design descriptions. Those textual SIF files were used for input of the experimental format converter which generates (extended) C code. The C code files were then passed to the prototype system using the existing C-frontend. As table 6 shows, most of the SIF benchmark designs can be successfully transformed into CDFG structures without any problems. Inside the prototype system, the design can be treated by high-level transformation and high-level estimation activities.

design	description	conversion OK ?	lines of VHDL (behav. level)	SIF [a]	CDFG [b]		lines of C
					CFG	DFG	
Monet™ benchmark designs							
gcd	greatest common devisor factorization	yes	37	45	13	64	31
memlab	memory lab exercise	yes	53	265	53	403	155
loop13	simple loop	yes	20	10	2	14	6
loop13_pipe	simple loop for pipelining	yes	26	70	11	67	33
simple_math	simple mathematical calculation	yes	25	36	2	30	18
matmult	matrix multiplication	no[c]	54	-	-	-	-
complx_mult	complex multiplication	yes	46	33	2	28	16
memlab_ram	memory lab exercise	no[c]	56	-	-	-	-
matmult_kl	matrix multiplication	no[c]	54	-	-	-	-
benchmark designs from University of Tübingen							
gcd	greatest common devisor factorization	yes	30	30	9	39	21
bubblesort	bubblesort sorting	yes	32	152	71	298	103
kmp	Knuth-Morris-Pratt pattern matching	yes	63	440	214	910	318
diffeq	differential equation	yes	35	71	6	68	40
ellip	elliptical wave filter	yes	64	139	2	102	78
kalman	Kalman filter	no[d]	78	3384	-	-	-
fancy	simple mathematical calculations	yes	80	159	38	207	99
traffic	traffic lights control	yes	57	137	34	177	85
fibonacci	fibonacci numbers	yes	30	34	6	38	20
vectoradd	vector addition	yes	40	51	19	91	34
matrixmult	matrix multiplication	no[e]	54	-	-	-	-
sum	sum calculation	yes	35	25	6	31	15
sum (2)	sum calculation and test	yes	44	38	10	48	25
exponent	exponent calculation	yes	54	38	10	49	25
determinant	determinant calculation	yes	42	41	2	35	14
binomial	coefficients of binomial expansion	yes	81	91	19	105	52

Table 6: Benchmark designs and results.

a. Complexity of SIF representations approximated by the number of controlflow/dataflow class instances.

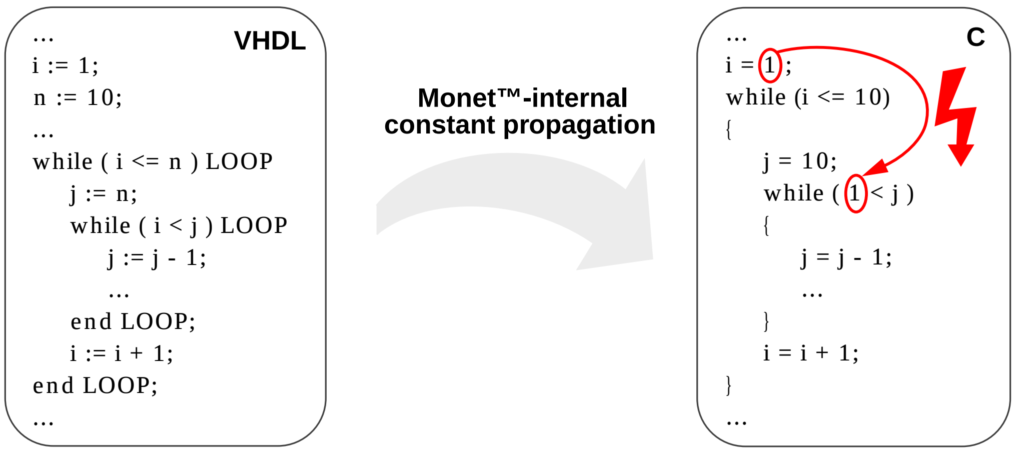
b. Complexity of CDFG representations approximated by the number of nodes of the controlflow-/dataflowgraph.

c. Not working with reference version of SIF database and SIF read module (no dualport memory supported).

d. Design too large for prototype system.

e. Internel Monet™ error during generation of textual-SIF format.

Furthermore, the experimental format converter can optionally be induced to produce an output file, which includes non-extended C code (and for this, can be compiled and executed). This feature can be used by the designer for test and validation purpose, for example, the identification of a bug in the Monet™-internal constant propagation algorithm: Figure 21 shows in (a) a segment of behavioral-level VHDL code (taken from the fibonacci benchmark design). During the conversion of the VHDL description into a SIF structure, Monet™ performs a propagation of constant values. Applying the C-output feature of the experimental format converter, there results C code segment (b), in which the illegal propagation of constant value 1 into the inner loop condition block can be easily backtracked (or identified by executing the C code).



```
...
i := 1;
n := 10;
...
while ( i <= n ) LOOP
    j := n;
    while ( i < j ) LOOP
        j := j - 1;
        ...
    end LOOP;
    i := i + 1;
end LOOP;
...
```

```
...
i = 1;
while (i <= 10)
{
    j = 10;
    while ( 1 < j )
    {
        j = j - 1;
        ...
    }
    i = i + 1;
}
...
```

(a) (b)

Figure 21. Identification of a bug in Monet™-internal constant propagation algorithm.

3.4 Conclusion

The experiments spent with the experimental format converter show, that on principle the application of the transformation and evaluation methodology described in chapter 2 to Monet™ designs is practicable and ingenious. For the coupling of Monet™ and transformation/evaluation tasks, two different realization strategies are imaginable:

1. Transformation of Monet™ SIF designs into CDFG representation by applying the SIF to CDFG format converter. Based on those CDFG structure, a transformation and evaluation of designs can be done within the prototype system. After finishing the transformational optimization process, a backward conversion of the CDFG structure into SIF representation has to done (by applying a CDFG to SIF format converter to be implemented). The optimized SIF representation is then treated by Monet™ synthesis (see figure 22).

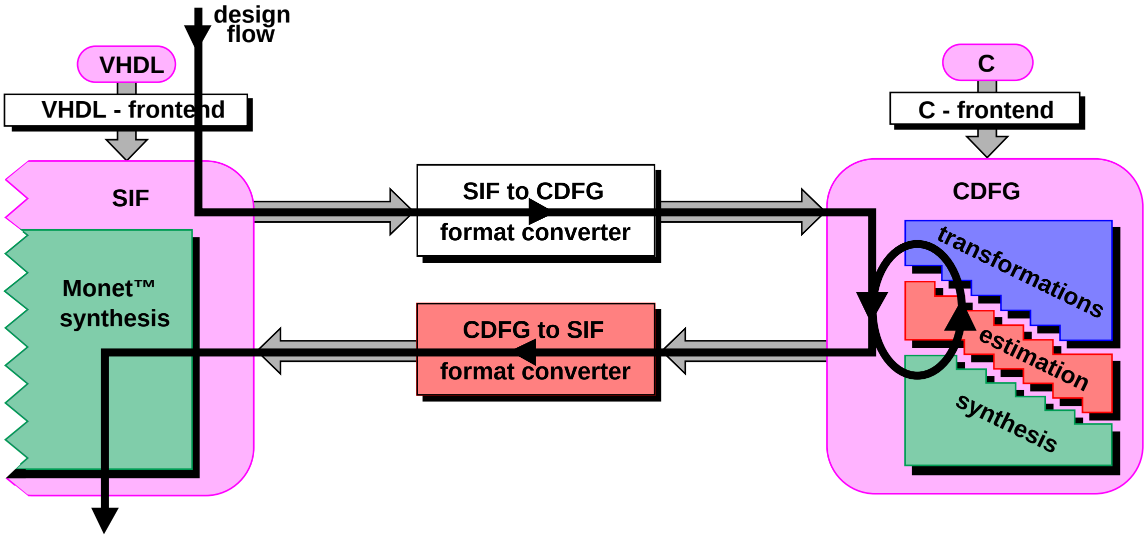


Figure 22. Strategy-1 for transformation/estimation integration into Monet™.

2. (Partially) reimplementation of transformation and evaluation concepts of the prototype system in Monet™. According to the methodology presented in chapter 2, the transformation/evaluation mechanisms immediately act on the internal high-level design representation, in case of Monet™ the SIF datastructure (see figure 23).

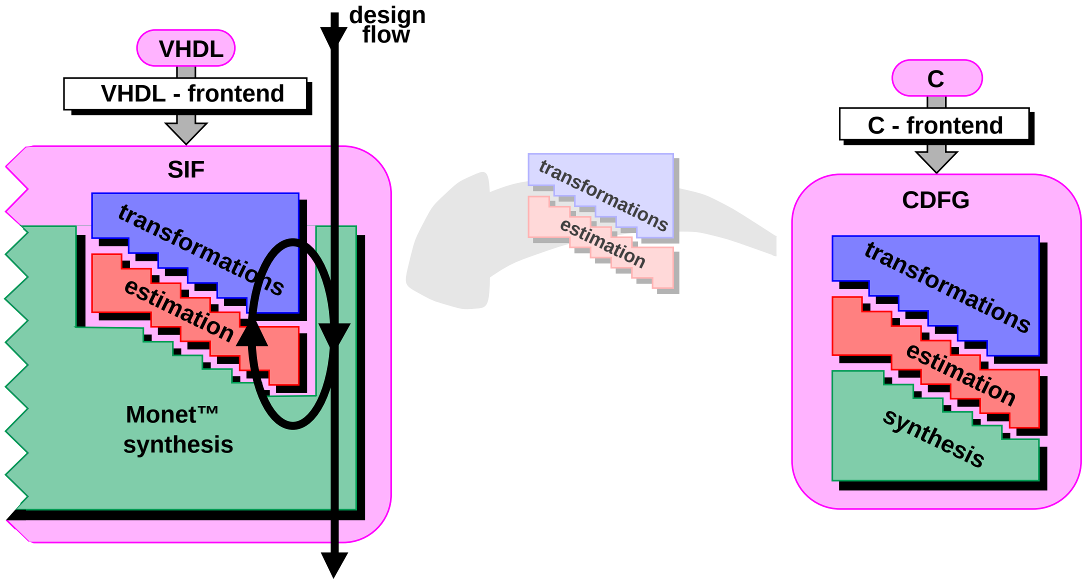


Figure 23. Strategy-2 for transformation/estimation integration into Monet™.

After careful analysis of the SIF design representation and strategies for SIF to CDFG conversion, we advise to favor the second realization strategy (reimplementation of transformation/ evaluation concepts in Monet™). The decision bases on following arguments:

- Due the fact, that SIF allows to handle *hardware* designs coming out of a *hardware* description language in a more accurate and detailed way than CDFG does (see section 3.2), the round trip including SIF to CDFG conversion, transformation/evaluation within the prototype system, and CDFG to SIF backward conversion leads to a loss of design information and compactness of representation (for examples, because of the fact that for specific SIF constructs there exist no directly corresponding CDFG construct). This leads to an alteration of the transformational behavior as well as a reduction of accuracy of the cost evaluation mechanisms. Another implication is, that CDFG to SIF backward conversion will be much more difficult to realize than SIF to CDFG conversion (because of the fact that design information has to be added during conversion, which means that CDFG to SIF translation represents more than a pure format conversion step).
- SIF corresponds to a well-devised and well-organized object oriented design representation and, for this, is well suited for supporting high-level transformation activities. There exist a set of transformation steps (for example, constant propagation and loop unrolling), which can be used for an object of analysis to understand how transformations manipulate the SIF structure. Some of the basic transformation steps (for example, constant propagation) are inseparably integrated into the SIF generation module. A first step will be to disconnect them from SIF generation and make them available in an isolated interactive manner (in combination with a correction or replacing of the algorithms, see section 3.2). The next step will be the extension of the basic transformation set by adapting and reimplementing CDFG transformation algorithms to/in SIF.

- SIF is also well suited for supporting high-level cost evaluation activities in the way presented in section 2.2: Similar to the prototype system, Monet™'s high-level synthesis steps are realized by isolated algorithms which immediately manipulate the internal SIF design representation. So, an analysis of the design representation can be applied to different execution stages of the high-level synthesis process, which is an important demand of our two level estimation approach. For a first approximation, existing cost estimation concepts for design criteria area and delay can be utilized for generation of level-1 cost estimation values. The Monet™ library concept provides cost estimation values corresponding to the ones produced by the prototype system level-0 cost heuristics (for design criteria area and delay). Level-0 cost estimation values for design criteria power can be received by the power macro-modeling approach of Barocci, Benini, Bogliolo, et al. [BaBeBo98]. So, an integration of the cost evaluation methodology outlined in section 2.2 can be realized in a step-by-step fashion.

4 Experimental Evaluation of Monet™

Project phase-3 of the first project period also includes an evaluation of the Monet™ high-level synthesis process. This chapter presents our designer's results and experiences in applying Monet™, and can be regarded independently of the previous chapters.

4.1 Application-1: Kohonens self-organizing map

A first application passed to Monet™ was part of an algorithm from the domain of neuronal networks, Kohonens self-organizing map (SOM). The problem to be (quickly) solved is: for a given input vector *X*, identify those neuron of the competition layer, whose weighting vector *W* is most similar to *X* (see figure 24).

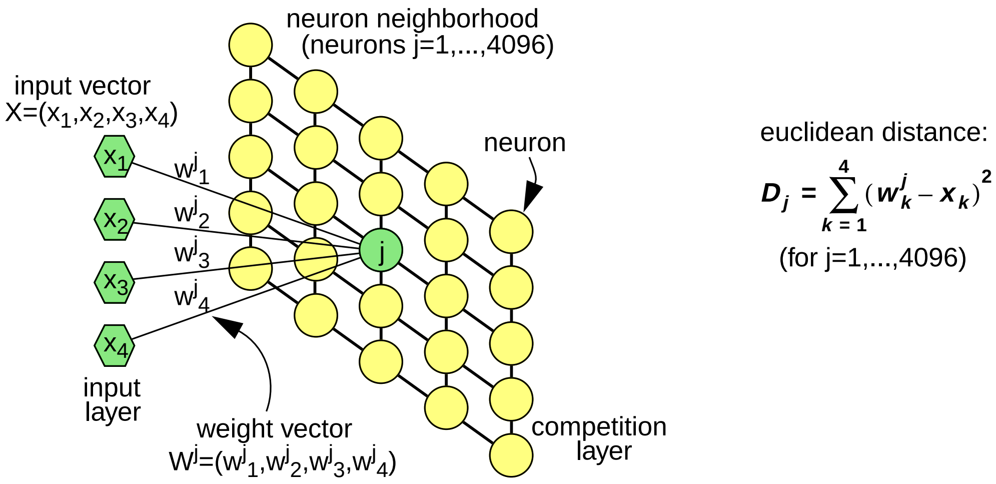

neuron neighborhood (neurons j=1,...,4096)
input vector $X=(x_1,x_2,x_3,x_4)$
x_1
x_2
x_3
x_4
w^j_1
w^j_2
w^j_3
w^j_4
j
neuron
input layer
weight vector $W^j=(w^j_1,w^j_2,w^j_3,w^j_4)$
competition layer
euclidean distance:
$D_j = \sum_{k=1}^{4} (w^j_k - x_k)^2$
(for j=1,...,4096)

Figure 24. Kohonens self-organizing map (SOM).

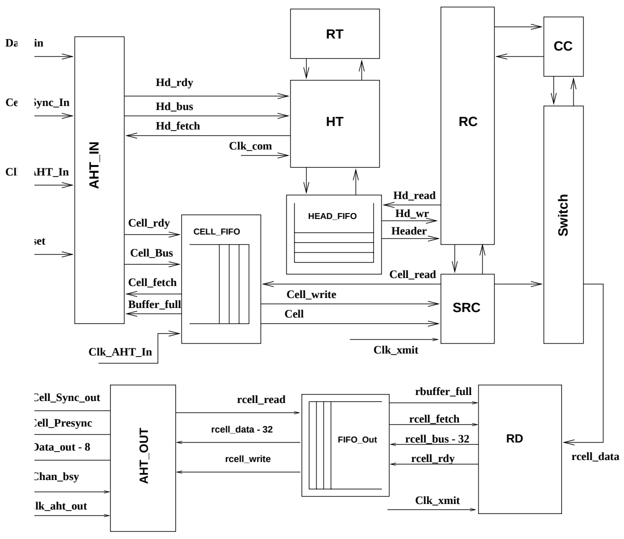
To solve this problem, the euclidean distance of *X* and *W* has to be calculated for each of the 4096 neurons of the competition layer. Table 7 shows the number of non-comment lines of behavioral VHDL code (*noc-lc*) and some characteristics of the synthesis results For this application, the ability of Monet™ to explore design alternatives (and for this, to aim at the fastest solution) turned out to be very important.

process		nc-loc	Monet™ synthesis	fastest	smallest	synthesis results			
						area [grid units]	delay [ns]	#csteps	#ops
SOM	euclidean distance	70	OK		x	2142	46.7	8	58
			OK	x		4180	27.0	6	58

Table 7: Synthesis results of distance calculation in SOM.

4.2 Application-2: ATM switch controller

Another application treated by Monet™ is an ATM switch controller [Pr94]. On highest level of abstraction, the controller can be split into seven VHDL modules of different size and complexity (see figure 25).



AHT_IN	input module
HT	header translator
RC	routing control
SRC	shift register control
CC	connection control
RD	receive data
AHT_OUT	output module

Figure 25. VHDL modules of the ATM switch controller.

For Monet™ synthesis, the lca_300k_comp_dc and lca_300k_dmag_dc libraries were used, and the superstate scheduling mode, a cycle time of 25 ns as well as the generation of a synchronous reset were selected. Since the application underlies realtime constraints, the allocation process was focussed on generating the fastest solution. Table 8 summarizes the results of the synthesis processes: all of the seven modules (containing altogether eleven processes) were able to be successfully synthesized by Monet™. Some of the modules (marked by brackets around „OK“ in the „Monet™ synthesis“ column in table 8) produce errors during the first synthesis attempts, which can all be successfully eliminated in cooperation with the Monet™ developer team. Details concerning the synthesis of the ATM switch controller modules can be found in [LaRo97] [LaRo98].

	module	process	nc-loc	Monet™ synthesis	fastest	smallest	synthesis results					
							area [grid units]	delay [ns]	#csteps	#ops	FSM	
											#states	area [grid units]
ATM	AHT_IN	ahtin	60	OK	x		1843	18.0	10	32	9	905
		payload	31	OK	x				52	238	51	15809
	HT	htproc	89	OK	x		1243	2.3	7	46	7	430
	RC	rcin	155	(OK)	x		1689	6.9	17	172	18	3590
		rcout	48	(OK)	x				6	39	7	517
	SRC	shiftproc	153	OK	x		7121	7.0	17	710	16	2093
	CC	ccin	178	OK	x		-	-	17	352	-	-
	RD	rdin	204	OK	x		14381	10.1	10	701	10	1042
		rdout	33	OK	x				6	29	5	357
	AHT_OUT	ahtout	71	OK	x		1338	23.4	29	197	30	7676
		ahtout_t	47	OK	x				52	162	57	19266

Table 8: Synthesis results of ATM switch controller.

4.3 Application-3: synthesis and transformation benchmark designs

This section summarizes the results of applying Monet™ to the synthesis and transformation benchmark design suit used for evaluation of the experimental SIF to CDFG format converter (see section 3.3). The benchmark design suit includes designs of different size, complexitie and application domain.

The synthesis process was performed twice for each module, focussed on the fastest and on the smallest solution. Again, lca_300k_comp_dc and lca_300k_dmag_dc component libraries were used, and the free scheduling mode as well as a cycle time of 25 ns were assumed. Table 9 summarizes the synthesis results for the benchmark designs which are not part of the Monet™ example set.

Regarding the functionality and handling of Monet™ it can be captured, that the ability to manually intervent into the synthesis process (for example, to reschedule local parts of the design under varying constraints) was found by the designers to be a very powerful way for optimizing critical parts of the design, in particular in combination with the existence of powerful tools for visualization and cross-probing, which allows a comfortable backtracking of bottlenecks through designs and design levels.

process		nc-loc	Monet™ synthesis	fastest	smallest	synthesis results					
										FSM	
						area [grid units]	delay [ns]	#csteps	#ops	#states	area [grid units]
benchmark designs	gcd	30	OK	x		1510	13.9	2	22	3	117
			OK		x	1219	14.6	3	22	4	146
	bubblesort	32	OK	x		13105	19.5	6	98	3	207
			OK		x	16835	24.6	8	98	5	207
	kmp	63	binding error	x		-	70.9	6	553	6	-
					x	-	35.0	15	553	15	-
	diffeq	35	OK	x		22234	22.6	4	31	5	180
			OK		x	12209	15.9	13	31	14	359
	ellip	64	OK	x		13285	44.3	3	57	3	69
			OK		x	6731	8.6	52	57	52	2444
	kalman	78	binding error	x		-	61.6	14	1423	15	-
					x	-	29.1	31	1423	32	-
	fancy	80	OK	x		1859	10.5	2	101	3	207
			OK		x	2326	19.3	13	101	13	1505
	traffic	57	OK	x		464	4.2	1	68	0	0
			OK		x	464	4.2	1	68	0	0
	fibonacci	30	OK	x		531	4.1	2	19	3	117
			OK		x	476	7.3	2	19	3	117
	vectoradd	40	OK	x		963	4.2	2	37	3	117
			OK		x	925	8.9	2	37	3	117
	sum	35	OK	x		401	4.1	2	16	3	117
			OK		x	362	5.6	2	16	3	117
	sum (2)	44	OK	x		432	3.9	3	24	3	117
			OK		x	389	5.7	3	24	3	117
	exponent	54	OK	x		601	6.3	3	23	3	117
			OK		x	496	6.7	3	23	3	117
	determinant	42	OK	x		7975	25.1	2	33	2	44
			OK		x	1980	14.1	10	33	10	310
	binomial	81	allocation error	x		1089	-	-	53	-	-
					x	680	-	-	53	-	-

Table 9: Synthesis results of benchmark designs.

5 Submitted Outline for Project Period Two

This chapter includes a proposal for a second project period. According to the discussion in section 3.4, SIF was found to be well qualified for supporting our transformation and evaluation methodology. The following proposal takes as a basis the approach of *direct* integration of transformation and evaluation mechanisms into Monet™ (corresonding to strategy-2 in section 3.4). The realization of a high-level transformation environment in Monet™ can be split into three phases:

- As motivated in section 3.2 by the example of the *≠0* construct , the SIF generation process includes built-in conversion rules for representation simplification, which map high level constructs to lower level representations. Those lower level constructs will possibly be well suited for synthesis purpose, but increase complexity of high-level transformation steps significantly. For this, a first project phase includes the adaption of those conversion rules to our high-level transformation methodology. This can be done by an extension of the SIF data structure by corresponding high-level SIF constructs (combined with an adaption of SIF generation and handling) or by spending additional effort inside the high-level transformation algorithms itself (for identification of low-level segments corresponding to high-level constructs). The decision for a strategy depends on the number of built-in conversion rules and the tightness of coupling of the low-level SIF constructs to the Monet™ synthesis process (corresponding to the effort to be spent for an integration of high-level extensions to SIF). Because of the absence of a complete SIF documentation, those discussion has to be done in tight cooperation with the Monet™ developer team.
- Some of the existing SIF transformations (for example, the propagation of constant values) are closely integrated into the SIF generation process. On the other hand, the applicability of those transformation steps during the transformational optimization process is not restricted to the initial design version (but may possibly be the result of former transformation steps). For this, a very important demand is the availability of those transformation steps in the optimization process in isolated form. The second project phase includes the isolation (and correction, see section 3.3) of those transformation steps, which allows an adaption of existing SIF transformations to the methodology presented in chapter 2.
- The third project phase covers the implementation of new transformation algorithms on top of the SIF data structure. Basis of those implementations are existing CDFG transformation algorithms of our prototype system, which have to be adapted to the SIF environment. A fixation of number and types of transformation steps to be realized has to be done in close cooperation with the Monet™ developer team. For decision help, work which is currently spent at University of Tübingen into a classification of transformation steps in regard of applicability, could be utilized. Precondition of this phase will probably be the implementation of a pool (class) of basic SIF manipulation methods.

According to the discussion in section 3.4, an integration of our cost evaluation mechanisms into Monet™ can be done step by step. Starting point could be the application of existing Monet™ cost estimation mechanisms for design criteria area and delay as well as the integration of power estimation heuristics of a project partner [BaBeBo98]. Corresponding to the Monet™ philosophy of interactive architectural design space exploration, we propose to focus project period two on a realization of transformation algorithms in Monet™ (combined with application of existing cost evaluation mechanisms). By this, interactive algorithmic level optimization steps are added to Monet™, which expands the set of optimization mechanisms available in Monet™ upwards (relative to level of abstraction). Problematic turned out to be the fact, that no complete documentation of the SIF data structure is available up to now. For implementation of transformation algorithms and evaluation mechanisms based on the SIF data structure, deep insights into the SIF data structure are indispensable. Therefore, we propose a tight cooperation with the Monet™ developer team, for example coupled with a stay at the Mentor™ research laboratory. It is planned that a Ph.D. student from University of Tübingen will spend several weeks at Wilsonville.

Table 10 proposes a project plan for project period two. Outcome of project period two will be a joint WSI report (corresponding to project period one) as well as implemented extensions to the Monet™ system.

project period 2	
contents	time schedule
phase-1: • analysis of SIF data structure details • high-level extension of SIF data structure or extension of high-level transformation algorithms	6 / 99
phase-2: • isolation/correction of existing Monet™ transformations • implementation of methods for high-level SIF manipuation • experimental validation of transformation algorithms	9 / 99
phase-3: • adaption of new transformation algorithms to SIF requirements • imlementation of new transformation algorithms in Monet™ • experimental validation of transformation algorithms	12 / 99

Table 10: Proposed project plan for project period two.

Further outlook: In Monet™, architectural exploration of design alternatives is performed in an user *interactive* way. Contents of some work currently spent at the University of Tübingen is the development and evaluation of algorithmic approaches on *automated* control of the transformational optimization process (based on the transformation and evaluation methodology presented in chapter 2) [GeRo98]. For this, a longer-termed cooperation goal (based on the work spent in project period one and two) could be the integration of mechanisms for automated transformation control into Monet™ (see figure 26).

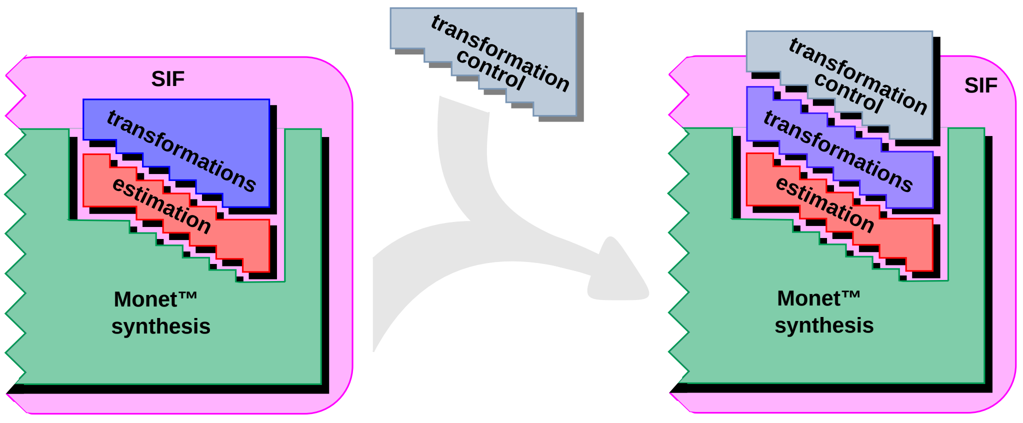


Figure 26. Integration of tranformation control into Monet™.

6 References

[AhSeUl86] A.V. Aho, R. Sethi, J.D. Ullman. Compilers: Principles, Techniques and Tools, Addison Wesley, 1986.

[BaBeBo98] M. Barocci, L. Benini, A. Bogliolo, B. Ricco, D. De Micheli. Lookup Table Power Macro-Models for Behavioral Library Components. Technical Report at University of Bologna, Stanford University, 1998.

[DIN94] European Digital CellularTelecommunications Systems, Phase 2: Full Rate Speech Transcoding. DIN Norm ETS 300580-2, DIN Deutsches Institut für Normung e.V., Beuth Verlag GmbH, 10772 Berlin., 1994.

[GaVaNa94] D.D. Gajski, F. Vahid, S. Narayan, J. Gong. Specification and Design of Embedded Systems. Prentice Hall, Englewood Cliffs, 1994.

[GeEiHa96] J. Gerlach, H.-J. Eikerling, W. Hardt, W. Rosenstiel. Von C nach Hardware: ein integratives Entwurfskonzept. In GI/ITG/GMM-Workshop Allgemeine Methodik von Entwurfsprozessen, Paderborn, March 1996 (in german).

[GeRo97] J. Gerlach, W. Rosenstiel. Ein skalierbarer Ansatz zur Kostenabschätzung für die Steuerung von High-Level Transformationen. Technical Report WSI-97-5 at University of Tübingen, September 1997 (in german).

[GeRo98] J. Gerlach, W. Rosenstiel. Transformationale Entwurfsraum-Exploration. In 43rd International Scientific Colloquium Ilmenau (IWK'98), Ilmenau, Germany, September 1998 (in german).

[HoEiHa94] A. Hoffmann, H.-J. Eikerling, W. Hardt, R. Genevriere. PSF - Paderborn Synthesis Format. Technical Report SFB 358 - B2 - 6/94. University of Paderborn, Technical University of Dresden, 1994 (in german).

[LaRo97] W. Lange, W. Rosenstiel. Modellierung einer ATM-Switch-Steuerung. Technical Report WSI-97-6, University of Tübingen, 1997 (in german).

[LaRo98] W. Lange, W. Rosenstiel. High-Level Synthese einer ATM-Switch-Steuerung. Technical Report, University of Tübingen, 1998 (in german) (to appear).

[MeNaSe98] K. Mehlhorn, S. Näher, M. Seel, C. Uhrig. The LEDA User Manual. University of Saarbrücken, Universität Halle-Wittenberg, Algorithmic Solutions GmbH Saarbrücken, www.mpi-sb.mpg.de/LEDA/MANUAL/MANUAL.html, Saarbrücken, 1998.

[Pr94] M. de Pryker. Asynchronous Transfer Mode: Die Lösung für Breitband-ISDN. Prentice Hall, 1994 (in german).

[ScFeMo97] W. Schwarz, G. Fettweis, A. Mögel. Sonderforschungsbereich 358: Automatisierter Systementwurf - Synthese, Test, Verifikation, dedizierte Anwendungen. In Wissenschaftliche Zeitung der Technischen Universität Dresden 46 (1997), Heft 2, pp. 31-46 (in german).

[SeSiLa92] E.M. Sentovich, K.J. Singh, L. Lavagno, et al. SIS: A System for Sequential Circuit Synthesis. Technical Report Memorandum No. UCB/ERL M92/41, Department of Electrical Engineering and Computer Science, University of California at Berkeley, May 1992.

[Wu97] Y. Wu. ALH SIF Design Specification. Internal Report, Mentor Graphics Inc.™, 1997.

Acknowledgments

This work was funded by Mentor Graphics Inc.™. The authors would like to thank the Monet™ developer team, in particular Mr. P. Gutberlet for helpful support and discussion, as well as Mr. W. Lange and Mr. X. Fang, both University of Tübingen, for their investigations in experimental evaluation of Monet™.