ROYAL INSTITUTE OF TECHNOLOGY

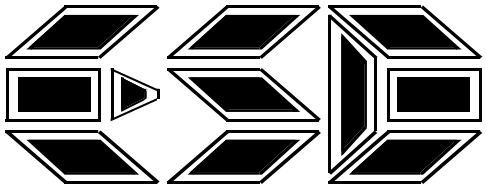
TRITA-ESD-99-1

ISSN 1104-8697

ISRN KTH/ESD/FOU--99/1--SE

A Survey of Design Transformation Techniques

Wenbiao Wu
Axel Jantsch



ELECTRONIC SYSTEMS DESIGN LABORATORY
ROYAL INSTITUTE OF TECHNOLOGY
ESDLAB/KTH-ELECTRUM
ELECTRUM 229
S-164 40 KISTA, SWEDEN

A SURVEY OF DESIGN TRANSFORMATION TECHNIQUES

Wenbiao Wu, Axel Jantsch
ESD Lab, Department of Electronics
Royal Institute of Technology
S-164 42 Kista, Sweden

Abstract

As the digital hardware systems grow in complexity and size, the trend in system design has been to design at higher and higher levels of abstraction. Program transformation supports the idea of designing at a higher level of abstraction which implies describing the behavior of the required system and then transforming this into a structural and behavioral description at a low level, possibly in a hardware description language. Such kind of design methodology can reduce the overall design time and ensure the correctness of the implemented system. In this paper, recent progress in the transformational development of complex hardware and HW/SW co-design systems is reviewed. A brief review of the research activities in software design is also provided in the first part of this paper.

1. INTRODUCTION

1.1 Scope

This paper is a survey of the current state of the art of research on methods of transformational techniques. The scope of this paper is restricted to the transformational techniques in hardware or HW/SW co-design research area. However, a brief review of the research activities in software design is provided in the first part of this paper. This survey does not claim to be fully exhaustive although an attempt has been made to cover most of the main systems using transformational design methodology in hardware or HW/SW co-design. Many of the technical details of the different approaches discussed have been glossed over or simplified; full details may be found in the cited references.

1.2 Program Transformation

Current hardware system design methods for specification, design and test are typically empirical and informal. As hardware designs grow in size and complexity, this kind of design method is proving less adequate. Formal system design methods which include a set of techniques based on mathematical foundation and analysis will guarantee correct and efficient system design. Since simulation for large designs is normally not exhaustive in reasonable time and post hoc verification is extremely costly, it is advisable to design with program transformation method. This process is based on a transformational approach that constructs the implementation by repeatedly applying a set of semantic-preserving transformation rules. The transformation rules are based on the calculus associated with the language used. System design constraints or optimization strategy can also be introduced during this process. Therefore, this design method is quite different from what a translator or compiler usually does. At the same time, the transformation approach has the potential benefit that the verification of the resulting design is achieved by the application of a sequence of correct transformation steps. That is, since each transformation step preserves the correctness of the original function, the resulting design is guaranteed to be correct. And since the above development

process can be automated because of the formal nature of program transformations. Therefore, design productivity can be largely improved in this way.

2. RESEARCH ACTIVITIES IN THE AREA OF SOFTWARE DEVELOPMENT

The concept of program transformation has its origin in Dijkstra's top-down stepwise refinement [Dijkstra68]. Program transformation methodology has been of great interest during the last three decades with the trends of increasing size and complexity of the software systems. The following are some of the intentions of these transformation systems: [Partsch83]

1. Program modification, such as optimization of program's control structure and efficient implementation.
2. Program synthesis, i.e. the generation of programs from a description of formal specification of the problem. The specification may be in a restricted natural language or in some formal languages.
3. Program adaptation. To change a program to some particular environments, such as change the programming language of the system.

In this paper, we are mostly interested in the program transformation technique in hardware or hardware/software co-design especially in the transformation of functional languages. The following section will give a brief view in the research activities in software development.

2.1 Burstall and Darlington's Work [Burstall75][Burstall77][Darlington81]

A great deal of the pioneering work in transformation systems was undertaken by Burstall and Darlington. Their ideas have heavily influenced today's transformation systems. Their system transformed applicative recursive programs to imperative ones. The followings are six rules used in their system:

- Definition - introducing a new recursion equation;
- Instantiation - introducing a substitution instance of an existing equation by replacing a parameter by a value;
- Unfolding - a recursive call to one of the recursion equations is replaced by the body of that equation;
- Folding - the body of an equation is replaced by a (recursive) procedure call;
- Abstraction - introducing a "Where" clause by deriving a new equation from a previous equation by replacing specific values by parameters;
- Laws (Algebraic Replacement Rule) - which are any set of data structure specific rules such as associativity, commutativity etc.

The recursive functions which have been transformed by the system are all fairly simple and mostly mathematical. This system is very primitive and is limited to transforming from recursion equations (imposing a restriction on the kinds of program that can be transformed) to improved recursion equations.

2.2 The CIP Project in Munich [Partsch90]

This work was carried out in Munich since 1975 by Bauer, Partsch and many others. The CIP is the acronym for Computer-aided, Intuition-guided Programming. Its main purpose is to produce a system which allows the user to construct programs by transformation.

The issues that were addressed include:

- To use a sound method (based on a formal calculus) for guiding the process of formal reasoning in program development;
- To design and define formally a wide spectrum language, CIP-L, in order to provide a uniform framework for the formulation and transformation of both specifications and programs;
- To develop an interactive system for supporting the process by performing the transformations mechanically, doing administration, and producing the documentation.

2.3 Other Works

A great many of program transformation systems are reported in [Partsch83], which also points to a lot of useful references. Martin Ward [Ward87] developed a theory of program refinement and equivalence, based on a wide spectrum language (WSL), which can be used as develop practical tools for program development and modification. A program transformation system of Polya at Cornell University is given in [Efremidis93] and [Efremidis94]. In [Mason94], transformation rules are based on a theory of contextual equivalence for functional language with imperative features. An overview of the program transformation methodology and future directions in this area is provided in [Pettorossi96] and [Paige96].

3. PROGRAM TRANSFORMATION TECHNIQUES IN HW OR HW/SW CO-DESIGN SYSTEMS

Within the field of hardware or HW/SW co-design, transformation methodology is extensively explored. Recent progress in the transformational development of complex hardware and HW/SW co-design system is reported in this section. Many of these techniques are capable of handling some experimental or industrial-sized examples.

As the digital hardware systems grow in complexity and size, the trend in system design has been to design at higher and higher levels of abstraction. Some of the reasons are:

1. The modern electronic systems have become extremely complex due to the increasing demand of higher performance and improved functionality. So, new design process or methodologies are required to capture this situation.
2. The time-to-market pressure has demanded the design to be finished in a shorter design cycle which also brings up the problem of design correctness. This is especially important for safety-critical systems.
3. We now have hardware design languages such as VHDL and Verilog. However, these languages are intended to be used at a relatively low level. And they have major drawbacks, that is they have no formal semantics. It is difficult to prove properties of designs in these languages.

Program transformation, however, supports the idea of designing at a higher level of abstraction which implies describing the behavior of the required system and then transforming this into a structural and behavioral description at a low level, possibly in a hardware description language. Such kind of design methodology can reduce the overall design time and ensure the correctness of the implemented system.

In this paper, we divide the hardware or HW/SW co-design systems using program transformation into several groups based on the system specification notations or languages. That is:

1. Imperative language based transformation systems;
2. Functional language based transformation systems;
3. Logic language based transformation systems;
4. Concurrent process based transformation systems.

Transformation systems with the imperative language as input usually have the advantage of expressing the system specification much easier than others. However, imperative language programs contain side-effects, i.e. they don't have the property of referential transparency. It is difficult to perform semantic-preserving transformations on them directly. So, the specifications in these languages have to be firstly translated into an internal representation which has a formal notation of semantics. Then, the transformation will be performed on this internal representation. Functional language as its name implies is based on the mathematical notation of functions. In contrast to the logic programming language where underlying model of computation is the relation, functional language has a more efficient operational behavior since functions allow more deterministic evaluation than relations.

3.1 Transformation Systems Based on Imperative Language with Concurrency

3.1.1 CAMAD (Computer-Aided Modelling, Analysis and Design) at Linköping [Peng94] [Hallberg95]

The CAMAD high-level synthesis system takes an algorithmic (behavioral) specification of a digital system and a set of design constraints as input and generates register-transfer level implementation (RTL).

The input specification is given in a Pascal-like language, called Algorithmic Design Description Language (ADDL), which consists of a subset of Pascal with extensions to express parallelism and hardware specific operations. The ADDL program specifies the functions the digital system is to perform without prescribing the physical structure of its implementation. The RTL net-list generated by CAMAD, on the other hand, specifies the hardware modules and their connection in the final design which will implement the given algorithm under the control of the generated finite state machine.

The first step of CAMAD is to map the ADDL specification into the Extended Time Petri Net representation (ETPN). The ETPN design representation is based on a parallel model with the data/control flow notations, augmented with timing information. The data flow part of the model is captured as a data path. The control flow dictates the partial ordering of data operations and is modeled by a Petri Net notation.

The generated ETPN model can be viewed as a primitive implementation to which correctness-preserving transformations are then applied successively until the final implementation is created. During these transformations, design tradeoffs are made to optimize either the cost or the performance under a set of design constraints.

The set of transformations used can be divided into three groups based on their functions:

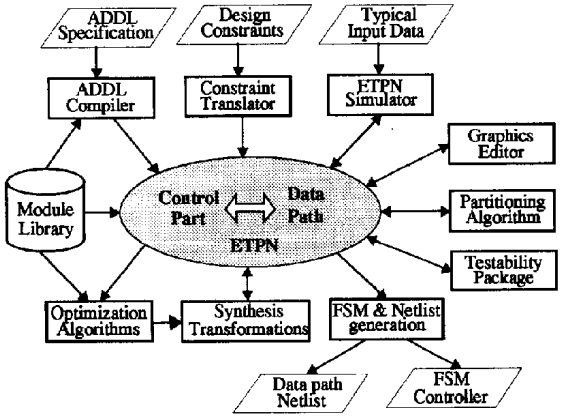
- Operation Scheduling Oriented Transformations which deals with a) determining the serial/parallel nature of the design, b) dividing or grouping operations into time steps, and c) changing the order of operations.

- Data Path Oriented Transformations which include vertex merger transformation and constant merger transformation.
- Control Oriented Transformations which include control merger and condition-signal grouping merger transformations.

The transformation process is guided by a heuristic design-search strategy into which designer interaction can also be added.

The transformation of an ETPN data path to a RTL design is carried out by a net-list generation procedure, which assumes that each ETPN data path vertex has a module in a module library which directly implements its function.

The overall structure of CAMAD is shown in the figure to the right.



As stated in [Peng94], the current version of CAMAD can run in an automatic mode or an interactive one. In the interactive mode, the designers interact with the synthesis algorithms and guide the synthesis process.

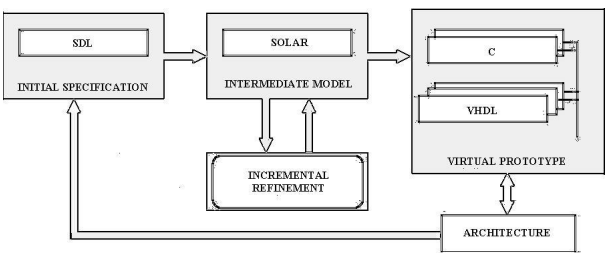
3.1.2 Cosmos/Solar at Grenoble, France [Marchioro97] [Marchioro98]

G. F. Marchioro et al. presented a semi-automatic methodology for hardware and software co-design system. The approach covers the co-design process through a set of user-guided transformations allowing semi-automatic partitioning. The transformations are based on a powerful set of primitives for functional partitioning, structural reorganization and communication transformation.

Cosmos starts with a system-level specification given in SDL and produces a distributed C-VHDL model that may be mapped on a distributed hardware/software architecture. Within Cosmos, codesign is decomposed into four major steps.

- Functional decomposition: this is aimed to split large behaviors that need to be executed on several processors.
- The virtual processor allocation fixes the number of processors and assigns an execution processor to each function.
- Communication transformation where processors communicating through high-level communication schemes are transformed into processors communicating signals.
- Prototyping. This includes the generation of C-VHDL models of the architecture and the mapping of this model onto the target architecture.

The transformational methodology is user-guided and assumes that the designer starts with an initial specification and an architectural solution in mind. System design from specification to implementation is performed through a set of primitives allowing the designer to transform the system, following an incremental refinement



scheme, in a distributed model that matches with the architectural solutions. Each step reduces the gap between specification and realization by fixing some implementation details (communication protocol, generating software or hardware code) or by preparing future implementation steps (merging and scheduling several processes to execute them in a single processor).

In Cosmos, the user-guided transformational approach makes use of three specification formats, SDL, Solar and C-VHDL. The initial specification is given in the system-level specification language SDL. The user controls the refinement process through a set of transformation primitives. The whole refinement process is based on an intermediate form called Solar. The output is a virtual prototype of the architecture given in a distributed C-VHDL model.

Within Cosmos, the partitioning steps are implemented through a set of primitives that performs basic transformations such as split, merge, move, flat and map. The system provides three sets of primitives working on the system structure, behavior and communication. The designer guides the interaction process and chooses the transformations needed in order to obtain the desired solution.

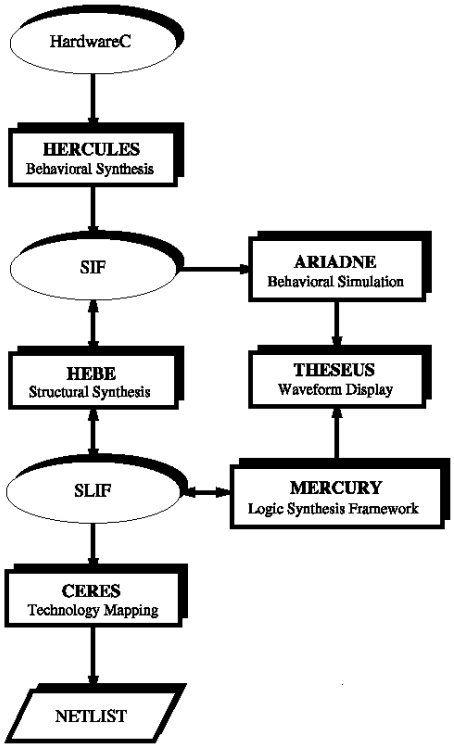
The organization of the partitioning into several small steps reduces the complexity of the problem. The designer controls the partitioning history within an interactive environment, through a fine grain control of the synthesis process. This methodology can be seen as a human guided compilation where the designer spends an additional effort to produce an efficient implementation.

3.1.3 Olympus at Stanford [Micheli88] [Micheli90]

Olympus Synthesis System is a vertically integrated design tool for specification and synthesis of digital circuits. The system has the following features:

1. A hardware design language, HardwareC, for design specification. HardwareC has C-like syntax, and a cycle-based semantics. It supports concurrency, structural and timing constraints, and has a unambiguous hardware semantics.
2. High-level synthesis tools, Hercules and Hebe for performing behavioral and structural synthesis. Hercules takes HardwareC and passes results to Hebe, which performs the tasks of scheduling and binding, and outputs a logic-level description of the design.
3. A technology mapping tool, Ceres, which translates a logic-level description into a technology dependent netlist.
4. Two simulators, Venus and Mercury for performing behavioral and logic level simulation.

The Block Diagram of the Olympus Synthesis System is shown in the right.



In Olympus, two internal models, Sequencing Intermediate Form (SIF) and Structural/Logic Intermediate Form (SLIF), are used to represent the hardware at different levels of abstraction and to provide a way to pass design information between the different tools. SIF is used in the behavior level and SLIF is used in the structural and logic level.

The objective of Hercules is to identify the maximal

parallelism that exists in the input description. Here, the input HardwareC description is parsed and translated first into an abstract syntax tree representation, which provides the underlying model for semantic analysis and behavioral transformations. The transformations are categorized into user-driven and automatic transformations.

User-driven transformations include the following:

- Selective in-line expansion of model calls;
- Selective operator to library mapping;

Automatic transformations include the following:

- For-loop unrolling, where fixed-iteration loops are unrolled to increase the scope of the optimizations.
- Constant and variable propagation, where the reference to a variable is replaced by its last assigned value.
- Reference stack resolution, where multiple and conditional assignments to variables are resolved by creating multiplexed values that can be referenced and assigned.
- Common sub-expression elimination, where redundant operations that produce the same results are removed.
- Dead-code elimination, where operations whose effects are not visible outside the model are re-moved.
- Collapse conditional, where conditionals with branches containing only combinational logic are collapsed to increase the scope in which logic synthesis can be applied.
- Dataflow analysis, where data and control dependencies among the operations are identified.

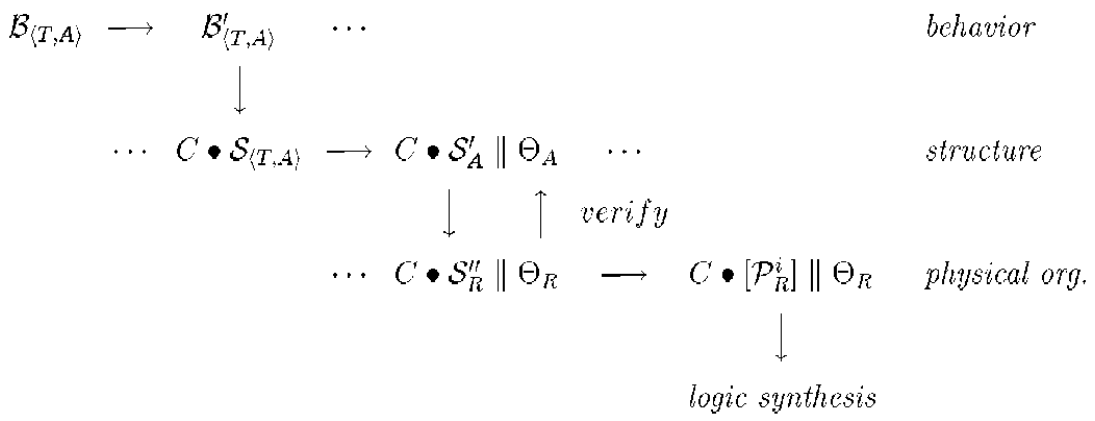
Upon completion of the automatic transformations, the behavior is optimized with respect to the data-dependencies that exist among the operations.

Olympus system has been used to design 3 ASIC chips in Stanford University and it has been tested against benchmark circuits for high-level and logic synthesis [Micheli90].

3.2 Functional Language Based Transformation Systems

3.2.1 DDD at University of Indiana [Bose94] [Johnson90]

DDD (Digital Design Derivation System) is a transformation system that implements a basic design algebra for synthesizing digital circuit descriptions from high-level functional specifications. The system is a formalization of digital design based on a functional algebra. The system is implemented in the Lisp dialect Scheme as a collection of transformations that operate on s-expressions.



Specifications are written in Scheme which are written in a purely functional style where there are no side effects. Descriptions are built from applicative terms, constants, identifiers, conditional expressions, and function definitions, and express globally synchronized systems.

In DDD, there are two classes of specifications, behavioral and structural. A construction from behavior to structure establishes the equivalence between the two classes of specification.

Design derivation in the DDD system has three major phases in which a typical design may undergo many iterations. Behavioral transformations manipulate the behavioral specification. These transformations usually involve manipulating control and architecture in a tightly integrated relation. Some examples include folding and unfolding transformations to achieve a proper scheduling of operations and transformations to move operations between control and architecture. The structural transformations manipulate the sequential system description which was built from the behavior description. These transformations are intended to refine the structural specification to an architecture. Projection transformation is a mechanism of incorporating representations of a more abstract type by another more concrete type. Projection allows the derivation of real hardware from concrete behavioral specifications. The result of these transformations is then passed to logic synthesis tools to generate hardware realization.

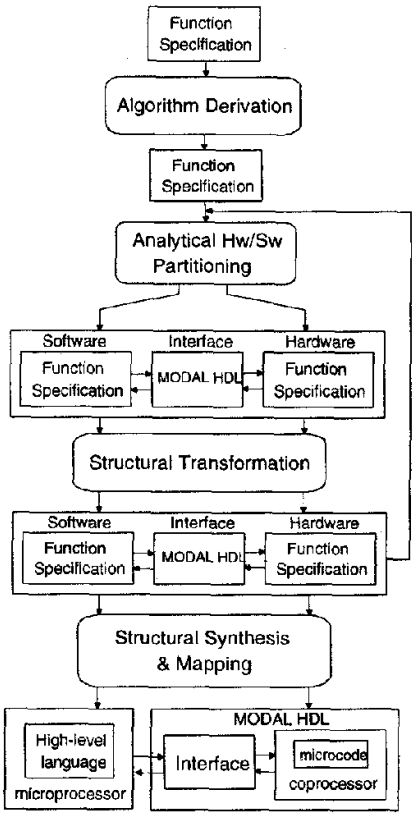
3.2.2 Works at University of South Wales [Cheung96, 97, 98]

A transformational co-design methodology was presented in their papers. The refinement process is separated into two levels, the algorithmic and the structural. Within each level, refinement is accomplished by applying sequences of transformations that preserve the functionality of the initial specification. Different algorithmic design and different spatial structures with different resources and performance costs are explored at algorithmic level and structural level.

The specification is written in a single high level functional notation called *form,* which provides for co-design a unfied system specification device. *Form* is based on a variant of FP with extensions to support multi-dimensional structured streams, delay functional and synchronized concurrent forms.

Algebraic rules which are a set of syntactic transformations that define a calculus for form are used in the transformation process. The rules permit symbolic manipulation of form expressions during structural transformation. Other transformation mechanism used include: recursion removal, conditional resources sharing, multiplexor reduction, parallelisation, serialization and loop unfolding transformation.

The transformational co-design process is shown in the figure to the right. It starts with a *form* specification of a task and constructs an algorithm that computes the desired function. The resulting implementation function is to be partitioned into hardware and software parts which satisfy the cost constraints. The final structural forms are mapped into the target implementation technologies to produce high level pseudo-code for the software components and applicative

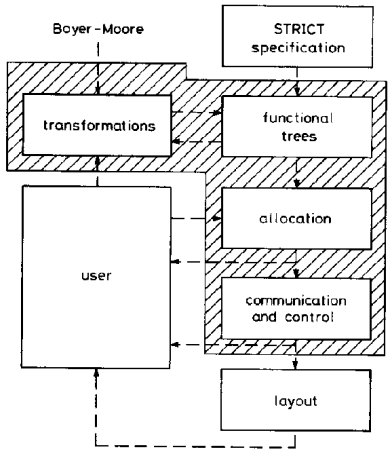


descriptions for the digital hardware components.

3.2.3 STRIDE by Burns et al. [Burns91, 94][Koelmans91]

A prototype interactive design tool, which is called STRIDE, has been implemented that allows easy use of a database of formally correct transformations. The language used in STRIDE is STRICT, which is a conventional hardware description language with features similar to VHDL. However, STRICT's behavior descriptions are mandatory and are written in a functional style.

STRIDE makes use of the fact that Boyer-Moore rewrite rules can be used as transformations. The Boyer-Moore logic is a quantifier-free, first-order logic with equality and function symbols. STRIDE works in the following way: Specifications written in STRICT are translated into an internal data structure, Boyer-Moore functions. Then STRIDE uses a set of Boyer-Moore library files to present the possible transformations to the designer. If the designer selects one, it will be applied by the making the appropriate changes to the internal data structure. Only Boyer-Moore rewrite rules are allowed as transformations. So, it is impossible to introduce bugs into the design. The block diagram of STRIDE is shown in the figure to the right.



3.3 Logic Language Based Transformation Systems

3.3.1 Ruby [Jones90, 91][Sharp93]

Ruby is a relation based language intended for specifying VLSI circuits. A circuit is described by a binary relation between appropriate, possibly complex domains of values, and simple relations can be combined into more complex relations by a variety of combining forms. The Ruby relations generate an algebra which defines a set of equivalences.

The algebra has been implemented in a rewriting tool called T-Ruby, which allows the user to rewrite Ruby terms according to pre-defined rules. The T-Ruby system enables the user to perform the desired transformations in the course of a design, to simulate the behavior of a class of implementable relations, and to translate the final Ruby description of such relations into a VHDL description for subsequent synthesis by high-level synthesis tools.

3.3.2 Larsson'd Work [Larsson93]

A transformational approach to the digital system design based on HOL proof system is reported in [Larsson93]. There are two levels of design representation. The first level is a design specification in logic (HOL) that is used for formal reasoning and the second level is a set of design annotations that are used to support design analysis and design checking.

The key component of their approach is the use of window inference package in HOL to model the transformational design process. Window inference is a style of reasoning where the user may transform an expression or restrict attention to a sub-expression and transform it. The window inference package is extended in their work since it doesn't support the design annotation. This is done in the following way. First design annotation is implemented as an abstract data type (ADT) in the ML meta-language. Then the definition of a window is extended with the design annotation

ADT and interface functions for accessing and update a windows design annotation component are added. The final implementation is also in HOL logic.

3.4 Concurrency Process Based Transformation Systems

3.4.1 Work by Barros [Barros94]

In paper [Barros94], a hardware/software partitioning methodology is proposed. It uses Occam as the source programming language and performs the partitioning by applying a series of algebraic transformations on the source program. The result is still an Occam program which consists of a set of parallel processes, one of which will be implemented in software and others in hardware. Its structure reflects the hardware and software components, and how they interact to achieve the overall goal. The software and hardware components generated by the process can then be compiled for further use.

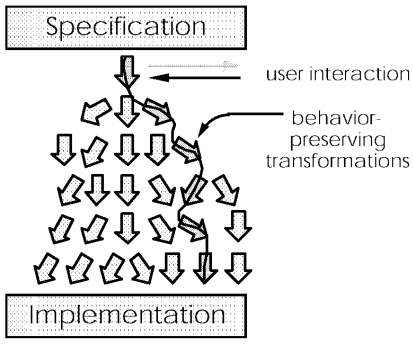
Occam, a small and formally based language which incorporate concurrency, is developed from CSP. Like CSP Occam obeys a large set of algebraic laws which can be used to carry out programming transformation with the preservation of semantics. This is also one of the reasons why Occam is chosen. However, the partition approach described in the paper deals only with a subset of this language.

The partitioning is done in the following steps. The first step is the splitting of the description into a set of communication processes using the transformation rules. Then the joining of processes in clusters takes place when building the clustering tree and placing the cut line at each clustering stage using a clustering algorithm. Transformation is also involved in this stage, which is equivalent to apply rules for joining processes.

The main emphasis of this work is to utilize the formal property of Occam, which result in a justified transformation through the use of algebraic laws. However their work is still limited. First, only a small part of the Occam is used in programming. So it still need to be expanded. Second, the underlining target architecture is too specific, which is predefined as consists of one software component and several hardware components.

3.4.2 TRADEs: a System for Transformational Design [Middelhoek93, 97]

TRADES (Transformational Design System), which has its origin in the ESPRIT SPRITE project of the EU, is developed at the University of Twente. The system is built around the single-token signal flow graph language (CDFG like) SIL (SPRITE Input Language), which combines control and data flow into a single graph format. The input to the system is VHDL code.



The design methodology of TRADE is based on the designer-driven application of small, local, behavior-preserving design transformations which include optimization transformation, refinement transformation, space-time transformation, composite transformation and so on. It is the responsibility of the designer to select between these possibilities. The design flow is shown in the figure above. The initial design specification is written in VHDL, which is first translated into SIL design representations. The transformation engine then provides a user interface that allows the designer to select a series of

transformations from a designer defined hierarchical list of transformations. After the transformation, a SIL to VHDL Translator will generate the VHDL codes, which can be synthesized using commercial RT-level synthesis tools.

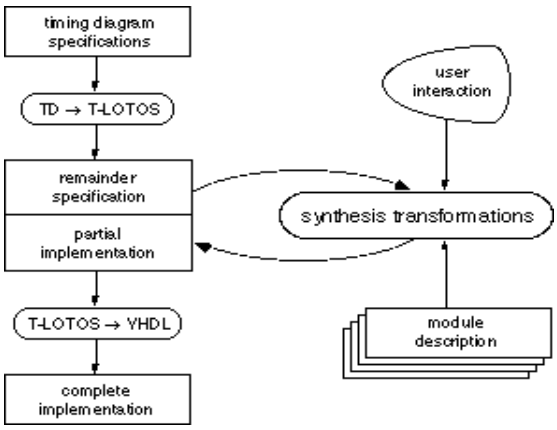
3.4.3 Self-Timed Circuit Design [Plosila99]

An action systems based design method for asynchronous self-timed VLSI circuits is presented in the thesis [Plosila99]. The action systems formalism is a framework for specification and correctness preserving development of concurrent programs, which is based on an extended version of the guarded command language of Dijkstra.

The design process consists of four main phases: initial derivation, handshake expansion, circuit extraction and implementation. The result of the system is circuit netlist.

3.4.4 FORMAT (Formal Methods in Hardware verification) Project [Grass95]

Werner Grass et al. have developed a system which transforms timing diagram specifications into VHDL code within the scope of FORMAT project which pursues a comprehensive system level design approach for communication hardware. Timing diagrams with data and timing annotations are used as a language for specifying interface circuits in their system.



System specifications are given exclusively as graphical timing diagrams with data and timing annotations. Timing diagrams are formalized in terms of a process calculus (T-LOTOS). This leads to a pure behavioral T-LOTOS description of the system. From any T-LOTOS representation a VHDL translation may be generated automatically. However, to improve the quality of the final result, the system also offered to introduce structure into the present behavioral specification. Provided a library of previously defined module descriptions (T-LOTOS and corresponding VHDL code), a user may apply a sequence of formal transformations to structurize his design. Such a procedure is know as interactive bottom-up synthesis in FORMAT. The synthesis path of FORMAT is shown above.

In FORMAT, the harpo tool automatically translates T-LOTOS specifications to VHDL. The translation model maps each of the T-LOTOS structures to VHDL.

3.5 Other Works in This Area

SynGuide [Samsom93,94] have been developed at IMEC, Belgium. Transformations are performed on Silage and can be used in an interactive and automatic mode. The most important transformations implemented in SynGuide are loop transformations, which can have a major impact on both memory requirements and number of cycles needed. The HYPER system developed at the University of Berkley provides a set of transformation for automatic algorithmic level design optimization. In HYPER [Hyper], Silage is used as the specification language and the transformations include retiming, loop unrolling and software pipelining. SHE (Software/Hardware Engineering) [Voeten96] [Putten98] is based on formal specification language POOSL (Parallel Object Oriented Specification Language) [Voeten98]. During analysis and design, POOSL

specifications are gradually transformed to incorporate architectural decisions and to satisfy design constraints. In [Pandey99], the semantics of VHDL is formalized in a declarative style using interval temporal logic. This makes it possible to validate transformations on VHDL programs and to formally reason about the timing aspects of VHDL.

LAMBDA (Logic And Mathematics Behind Design Automation) [Wang91] is a general-purpose theorem-proving based CAD tool that integrates design and verification. It is based on higher-order logic and implemented in the functional programming language ML. Busch [Busch91] use the LAMBDA system to design digital hardware with a transformational technique where various design aspects like parameterized retiming of synchronous systems and constrained refinement are formalized in a uniform framework. The derived design is not a specific implementation but a space of valid solutions. In [Wang91], a subset of VHDL has been formalized that a VHDL design description can be transformed into a design description in LAMBDA function specification language via a state-machine model of the design. Then, the LAMBDA/DIALOG system is used to synthesis correct implementation.

3.6 A Comparison of Different Systems

Table 1 shows the systems that we have reported at previous sections. The columns outlined in this table are: input (what's the languages or notations used in system specification), output (what's the final result of the transformation), internal representation (what kind of internal representation is used if applicable), user guidance (if the designer can direct the transformation steps), automatic (can the system work without the guidance of designer) and formal (are the languages and transformation rules used based on formal semantics).

		INPUT	OUTPUT	INTERNAL REPRESENTATION	USER GUIDANCE	AUTOMATIC	FORMAL
IMPERATIVE LANGUAGE BASED	CAMAD at Linköping	ADDL	RTL Netlist	ETPN	YES	YES	YES
	Cosmos/Solar at Grenoble	SDL	C-VHDL	SOLAR	YES	NO	NO
	Olympus at Stanford	HardwareC	Logic-level Implementation	SIF and SLIF	YES	YES	-
FUNCTIONAL LANGUAGE BASED	DDD at Univ. of Indiana	SCHEME	A Collection of Boolean Subsystems	NO	YES	NO	YES
	Cheung's work	Form	Structural Forms	NO	-	-	YES
	STRIDE by Burns et al.	STRICT	STRICT	Boyer-Moore functions	YES	NO	YES
LOGIC LANGUAGE BASED	Ruby	Ruby	Circuits	-	YES	-	YES
	Larsson's work	HOL	HOL	-	-	-	YES
CONCURRENT PROCESS BASED	Barros' Work	OCCAM	Structural OCCAM	NO	-	-	YES
	TRADES at Univ. of Twente	VHDL	VHDL	SIL	YES	NO	-
	Self-Timed Circuit Design	Action Systems Formalism	Circuit Netlist	-	-	-	YES
	FORMAT Project	Timing Diagram	VHDL	T-LOTOS	YES	YES	-

Table 1. Example Transformation Based HW or HW/SW Codesign Systems

4. POSSIBLE TRENDS IN THE FUTURE RESEARCH

Though a lot of progress has been made in the program transform techniques in recent years, transformational methodology is still not in large scale usage nowadays. This is possibly because the efficiency and real-time requirements are very stringent in HW design and also the design space in HW design is very large since the target architecture is always a design variable.

However, compared with other design methodology, the benefits of transformational design methodology are still very attractive. For example, for large and complex digital designs, post hoc

verification or exhaustive simulation is still not feasible now. Here we give out some of the possible trends in this research area.

- System that supports incremental specification. Specifications of large systems are not written as one homogeneous document. This process should be supported by a technique that allows the designers to add, modify and remove some part of the specification without a large impact on the rest of the specification.
- System that supports the verification of the transformational steps. The system specification should based on a formal semantics which can guarantee the correctness of transformation steps.
- System that supports model checking at the specification level.
- System that supports transformational optimization to the design. Timing, spatial and other constraints should be used in the guidance of transformational design for system optimization purpose.
- System that has functional language as the specification language to simplify the design. In [Jantsch98], it suggests that specification languages based on functional paradigm can be more suitable than others. Because of the formal mathematical bases of functional language, it is also very suitable in the transformational design.

REFERENCES:

[Barros94] E. Barros and A. Sampaio, Towards provably correct hardware/software partitioning using OCCAM, Proceedings of the Third International Workshop on Hardware/Software Codesign, pp. 210-217, 1994.

[Bose94] Bhaskar Bose, DDD-FM9001: Derivation of a Verified Microprocessor. PhD thesis, Computer Science Department, Indiana University, USA, 1994. Also Technical Report No. 456, 165 pages.

[Burns91] F. P. Burns, D. J. Kinniment and A. M. Koelmans, Correct Interactive Transformational Synthesis of DSP Hardware**,** European Design Automation Conference, Amsterdam, pp. 16-21, IEEE Computer Society Press, 1991.

[Burns94] F. P. Burns, D.J. Kinniment and A.M. Koelmans, STRIDE: a Tool for Formal Interactive Systems Synthesis, IEE Proc. on Computers and Digital Techniques, Vol. 141, No. 6, pp. 347-355, 1994.

[Burstall75] R. M. Burstall and John Darlington. Some Transformations for Developing Recursive Programs. In Proceedings of International Conference on Reliable Software (Los Angels) IEEE, New York, pp. 465-472, 1975.

[Burstall77] R. M. Burstall and John Darlington. A Transformation System for Developing Recursive Programs. Journal of the ACM, 24(1), pp. 44-67, January 1977.

[Busch91] Holger Busch and Gerd Venzl, Proof-aided Design of Verified Hardware, Proceedings of the 28th Conference on ACM/IEEE Design Automation Conference, pp. 391-396, 1991.

[Cheung96] T. Cheung and G. Hellestrand, Multi-level Equivalence in Design Transformation, Proceedings of International Conference on Computer Hardware Description Languages, Chiba Japan, pp559-566, Sep. 1996.

[Cheung97] T. Cheung, G. Hellestrand and P. Kanthamanon, A Transformational Codesign Methodology Design Automation Conference, 1997. Proceedings of the ASP-DAC '97 Asia and South Pacific, pp. 299 –305, 1997.

[Cheung98] T. Cheung, G. Hellestrand and P. Kanthamanon, A Multi-Level Transformation Approach to HW/SW Codesign: A Case Study, Proceedings of 4th International Workshop on Hardware/Software Codesign, Codes/CASHE '96, IEEE Computer Society Press, pp10-17, 1998.

[Darlington81] John Darlington, A Experimental Program Transformation and Synthesis System, Artificial Intelligence pp. 1-46, 16(1981).

[Dijkstra68] E. W. Dijkstra, A Constructive Approach to the Problem of Program Correctness. BIT 8, No. 3, pp. 174-186, 1968.

[Efremidis93] Sofoklis G. Efremidis and David Gries, An Algorithm for Processing Program Transformations, TR93-1389, Cornell University, October 1993.

[Efremidis94] Sofoklis G. Efremidis, On Program Transformations, PhD Thesis, TR94-1434, Cornell University, June 1994.

[Grass95] W. Grass, C. Grobe, S. Lenk, W. D. Tiedemann, C. Delgado Kloos, A. Marin and T. Robles, Transformation of Timing Diagram Specifications into VHDL Code, Proc. 12th IFIP Int. Conf. on Computer Hardware Description Languages and their Applications CHDL'95, Chiba, Japan, pp. 659-668, 1995.

[Hallberg95] J. Hallberg and Z. Peng, Synthesis under Local Timing Constraints in the CAMAD High-Level Synthesis System, Proc. of IEEE EUROMICRO 95, Como, Italy, 1995.

[Hyper] http://infopad.eecs.berkeley.edu/~hyper/Background/paperBackground.html

[Jantsch98] A. Jantsch, S. Kumar, I. Sander, B. Svantesson, J. Öberg, and A. Hemani, P. Ellervee, M. O'Nils, Comparison of Six Languages for System Level Descriptions of Telecom Systems, Proceedings of the Forum on Design Languages, vol. 2, pp. 139-148, 1998.

[Johnson90] Steven D. Johnson and Bhaskar Bose. A System for Mechanized Digital Design Derivation. In IFIP and ACM/SIGDA International Workshop on Formal Methods in VLSI Design, 1991. Also Technical Report No. 323, 47 pages.

[Jones90] Geraint Jones and Mary Sheeran, Circuit Design in Ruby, Lecture notes on Ruby from a summer school in Lyngby, Denmark, September 1990.

[Jones91] Geraint Jones and Mary Sheeran, Relations and Refinement in Circuit Design, in 3rd Refinement Workshop, 1990, eds. Carroll Morgan and Jim Woodcock, Springer, Workshops in Computing, 1991.

[Koelmans91] A. M. Koelmans, F. P. Burns and D. J. Kinniment, Use of a Theorem Prover for Transformational Systhesis, Colloqium on Formal and Semi-formal Methods for Digital Systems Design, London, IEE Press, 1991.

[Larsson93] Mats Larsson, A Transformational Approach to Formal Digital System Design, Linköping Studies in Sciences and technology, Thesis No 378, 1993.

[Marchioro97] G. F. Marchioro, J. M. Daveau and A. A. Jerraya, Transformational Partitioning for Co-design of Multiprocessor Systems, Computer-Aided Design, 1997. Digest of Technical Papers., 1997 IEEE/ACM International Conference on , pp. 508 –515, 1997.

[Marchioro98] G. F. Marchioro, J. M. Daveau, T. B. Ismail and A. A. Jerraya, Transformational Partitioning for Codesign, IEE Proceedings on Computers and Digital Techniques, Vol. 145 No. 3, pp. 181 –195, May 1998.

[Mason94] Ian A. Mason and C. L. Talcott. Program Transformation via Contextual Assertions. In Logic, Language and Computation. Festschrift in Honor of Satoru Takasu, volume 792 of Lecture Notes in Computer Science, Springer, Berlin, pp. 225-254, 1994.

[Micheli88] G. De Micheli and D. Ku, HERCULES - a System for High-level Synthesis, Proceedings of 25th ACM/IEEE Design Automation Conference, pp. 483-488, 1988.

[Micheli90] G. De Micheli, D. Ku, F. Mailhot, and T. Truong, The Olympus Synthesis System, IEEE Design & Test, pp. 37-53, October 1990.

[Middelhoek93] P. F. A. Middelhoek, Transformational Design of Digital Circuits, Proceedings of the Seventh Computersystems Workshop, Eindhoven, pp. 57-69, November 1993.

[Middelhoek97] P. F. A. Middelhoek, Transformational Design: an Architecture Independent Interactive Design Methodology for the Synthesis of Correct and Efficient Digital Systems, PhD. Thesis, University of Twente, April 1997.

[Paige96] R. Paige, Future Directions in Program Transformations, ACM Computing Surveys, Article 170, 28A(4), Dec. 1996.

[Pandey99] S. L. Pandey, K. Umamageswaran and P. A. Wilsey, VHDL Semantics and Validating Transformations, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 18, No. 7, pp. 936-955 July 1999.

[Partsch83] H. Partsch and R. Steinbruggen, Program Transformation Systems, ACM Computing Surveys, Vol. 15, No. 3, pp. 199-236, 1983.

[Partsch90] Helmut A. Partsch, Specification and Transformation of Programs, Springer 1990.

[Peng94] Z. Peng and K. Kuchcinski, Automated Transformation of Algorithms into Register-Transfer Level Implementations, IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems, Vol. 13, pp. 150-166, 1994.

[Pettorossi96] A. Pettorossi and M. Proietti, Rules and Strategies for Transforming Functional and Logic Programs. ACM Computing Surveys, Vol. 28, No. 2, pp. 360-414, June 1996.

[Plosila99] Juha Plosila, Self-Timed Circuit Design – The Action Systems Approach, Department of Physics, University of Turku, Finland, June 1999.

[Putten98] P. H. A. van der Putten, J. P. M. Voeten, M. C. W. Geilen and M. P. J. Stevens, System Level Design Methodology, Proceedings of IEEE Computer Society Workshop on System Level Design, pp. 11-16, 1998.

[Samsom93] H. Samsom, L. Claesen, H. De Man, SynGuide: an Environment for Doing Interactive Correctness Preserving Transformations, IEEE Workshop on VLSI Signal Processing, Veldhoven, The Netherlands, Oct. 1993. Also in VLSI Signal Processing VI, L.Eggermont, P.Dewilde, E.Deprettere, J.van Meerbergen (eds.), IEEE Press, New York, pp.269-277, 1993.

[Samsom94] H. Samsom, F. Franssen, F. Catthoor and H. De Man, Verification of Loop Transformations for Real Time Signal Processing Applications, VLSI Signal Processing, VII, pp. 208-217, 1994.

[Sharp93] Robin Sharp and Ole Rasmussen, Transformational Rewriting with Ruby, In Computer Hardware Languages and their Applications (CHDL'93), pages 243-260, 1993.

[Voeten96] J. P. M. Voeten, P. H. A van der Putten and M. P. J. Stevens, Behaviour-preserving Transformations in SHE: a Formal Approach to Architecture Design, EUROMICRO 96. Beyond 2000: Hardware and Software Design Strategies., Proceedings of the 22nd EUROMICRO Conference, pp. 19-27, 1996.

[Voeten98] J. P. M. Voeten, P. H. A van der Putten, M. C. W. Geilen and M. P. J. Stevens, System Level Modelling for Hardware/Software Systems, Proceedings of 24th Euromicro Conference, pp. 154-161, 1998.

[Wang91] Xinning Wang and E. P. Stabler, Formalization of VHDL Synthesis Procedure in Higher-order Logic, International Workshop on the HOL Theorem Proving System and Its Applications, pp. 106-119, 1991.

[Ward 87] M. Ward, Proving Program Refinements and Transformations D.Phil Thesis, Oxford University, 1987.