Design Correctness of Digital Systems

Corrie Huijs
University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
Tel.: +31 53 4893697, fax: +31 53 4894590, e-mail: chuijs@cs.utwente.nl

Abstract

Transformational design is a formal technique directed at design correctness. It integrates design and verification by the use of pre-proven behaviour preserving transformations as design steps. A formal framework is necessary but hidden for the designer. Five formal aspects are integrated in the presented formal framework that is aimed at the design of complex systems composed out of different kinds of subsystems. The tagged signal model is used as 'meta' model for a heterogeneous set of computational models with different concurrency semantics. The offered possibilities of model refinement by transformations and the ability to incorporate heterogeneity are valuable extensions with respect to other transformational design approaches for high-level synthesis.

Keywords: transformational design, correctness, high-level synthesis, model refinement and heterogeneity

1. Introduction

Design correctness of high-level synthesis, the derivation of RTL (Register Transfer Level) implementations from behavioural specifications, is necessary to avoid costly design iterations. A correct design satisfies the behaviour and constraints given by the specification. Integration of design and correctness verification is needed because of the complexity and size of modern systems that are often compositions of different kinds of subsystems. *Transformational design* based on "correctness by construction" is preferred above design methodologies in which either simulation or post-verification is used to guarantee the design correctness [1]. The possibilities and limitations of this design approach are subject of research. Transformational design systems developed differ in application area, design representation, transformations and proof support. HASH is based on HOL and directed to clock synchronous systems specified without non-determinism [2]. T-Ruby is based on Isabelle and directed to regular clock synchronous systems [3]. CAMAD is based on extended petrinets and especially directed to scheduling transformations. Our TRADES (TRAnsformational DESign) system is developed to determine the feasibility of transformational design and to get insight in this design technique and its formal aspects. Feasibility is shown [5], which motivates the elaboration of our system and of its formal framework. In the TRADES system the transformations are defined on an intermediate design representation, SIL, in order to benefit from the differences of specification languages (see fig. 1) [5,8]. Each subsystem can be specified in the most suitable specification language.

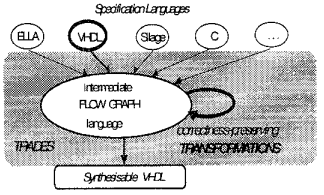


Figure 1: TRADES system

The intermediate design representation has a formal semantics on which the definition of correctness is based. This semantics is one of the formal aspects of transformational design and it supports the heterogeneity of the computational models of the specification languages. Behavioural specifications use besides abstract data types also abstract timing models. The refinement of abstract timing models to concrete timing models, like multi-rate and clock synchronous models, is an essential part of high-level synthesis and therefore is incorporated in our transformational design approach. This is achieved by integrating several system models and their interaction in our formal framework by the use of the tagged signal model of Lee [6]. This formal framework for transformational design is presented.

2. Formal aspects of transformational design

Correctness of transformations needs to be well defined before it can be proven. The definition of transformations is based on the design representation and their correctness on the semantics of this representation. The design representation and its formal semantics need to be based on system models. These formal aspects of transformational design are interrelated and therefore integrated in a formal framework. The five formal aspects integrated are:

1. **system modelling**
2. design representation with **formal semantics**
3. the **correctness definition** used
4. formal **definitions of transformations**
5. **correctness proofs** for transformations

These formal aspects are essential but hidden to the designer, which increases acceptability of this approach.

3. System modelling

System models are simplifications of reality and are used to describe and/or analyse the essential characteristics of systems. The simplifications inherent to a model limit the validity of the correctness guarantees. A guarantee of correctness therefore only is meaningful if also the model to which it is related is defined and known.

System models give besides the selection of the important system characteristics also representations for these characteristics. These representations determine model characteristics. System and model characteristics of importance for high-level synthesis by transformational design are related to behaviour and structure:

Model characteristics:

- A *mathematical representation of behaviour* in order to define correctness as behavioural equivalence or behavioural implication.
- *Compositionality* of behaviour with respect to structure. The behaviour of a composition needs to be a composition of the behaviour of the components. This allows interchanging components with equivalent behaviour.
- *Transparency.* There may be no behavioural difference between a composition used as subsystem (eventually at different locations) and the use of the same composition as system. This preserves transformations to be context dependent.
- *Hierarchy* in order to handle complexity.
- *Support for different levels of abstraction* for data, structure, behaviour and time in order to handle complexity.
- *Extendibility* in order to incorporate computational models related to new application areas.
- *Visualisation of structure* in order to handle complexity and stimulate overview.

System characteristics:

- *Determinism.* The designs have to be deterministic in time and value but can be specified non-deterministic which offers design freedom.
- *Causality.* No spontaneously generated output is allowed.
- *Concurrency.* Inherent to hardware and efficient.
- *Heterogeneity in computational models* of subsystems. Control as well as data dominated.

In hardware design the FSM model is important but not suitable to handle hierarchy and concurrency. Several proposals extend FSM with hierarchy and concurrency. The concurrency models used are different and often based on simultaneity. Heterogeneity in concurrency models can be obtained by disconnection of hierarchy and concurrency [7].

3.1 Model refinement

In the design process several kinds of refinements are important:

- Behaviour refinement: reduction of non-determinism
- Time refinement
- Data-refinement
- Structure refinement

The conversion of integers to bitstrings is an example of data refinement. An example of time refinement is the case in which only the order of the input and output events is used in the specification which is refined into a clock-synchronous implementation. In this implementation design freedom with respect to time is reduced. The consequence is even stronger: Only a subset of the input combinations allowed by the specification is allowed by the implementation. The specification is refined too! It is important that this is made explicit. Explicit specification refinements can prevent correctness problems but influence also the correctness definition [3].

3.2 The tagged signal model

Behaviour needs to be more than a relation between input values and output values. It has to be defined as a relation between streams of input values and streams of output values. The definition of streams needs to be flexible enough to cover abstract as well as concrete timing models: timing models in specifications as well as timing models in RTL descriptions. This flexibility is offered by the tagged signals of Lee [6] which are functions on a tag domain. Tags can be viewed as a kind of time stamps but are more abstract. Tags are partially ordered while time conceptually is totally ordered. The tag concept offers the possibility to integrate unrelated timing models. Events are values together with time stamps. At each location in a system series of events can be observed: *tagged signals*. The time stamps of the events belonging to a tagged signal are totally ordered. Two tagged signals can have unrelated tag-sets.

Definition 1: Tagged signals

Let V be a set called the *value set* and let $< T, \sqsubseteq >$ be a set T called the *tag set* together with a partial order $\sqsubseteq$ defined on it. An ***event*** is a 2-tuple $< t, v > \in T \times V$. A ***tagged signal*** is a set of events of which the tags are totally ordered with respect to $\sqsubseteq$.

4. Design representation together with formal semantics

Graphs are used as structure representations and their formal semantics relates structure to behaviour. In order to benefit from graph rewriting theory a specific kind of edge labelled hyper-graphs is chosen [8].

4.1 IO ported edge labeled hyper-graphs

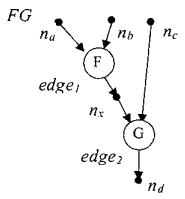
A hyper-graph consists of hyper-edges and nodes that are respectively related to computational elements and communication. This is a reversal with respect to normal use of nodes and edges in graph representations. The advantage is that order of inputs and outputs of computational elements is essential which corresponds with the order importance of variables in definitions of functions and relations. Besides the general formal definition of our graphs a constructive definition is used. The latter is of great importance in the definition of transformations and therefore illustrated by an example.

Definition 2:

2.1 X^* is the set of all finite sequences (tuples) of elements of *set X*.

2.2 An ***IO ported edge labelled directed hyper-graph over LABS,*** a set of relation labels, is a 7-tuple $G = < N, E, s, t, lab, INn, OUTn >$ in which N is the set of nodes, E is the set of directed hyper-edges, $s{:}E \rightarrow N^*$ and $t{:}E \rightarrow N^*$ are functions that give sequences of sources respectively targets of the hyper-edges, $lab{:}\ E \rightarrow LABS$ is a function that gives the labels of the hyper-edges, $INn \in N^*$ and $OUTn \in N^*$ the sequences of external nodes representing *input-ports* and *output-ports* respectively.

Example 1 : *FG*, the *IO ported edge labelled directed hyper-graph over* $\{F, G\}$



$$FG = <\{n_a, n_b, n_c, n_d, n_x\},$$
$$\{edge_1, edge_2\},$$
$$\{ edge_1 \mapsto <n_a, n_b>^1,$$
$$edge_2 \mapsto <n_x, n_c>\}$$
$$\{ edge_1 \mapsto <n_x>,\ edge_2 \mapsto <n_d>\}$$
$$\{edge_1 \mapsto F,\ edge_2 \mapsto G\}, \{ n_a, n_b, n_c\}, \{ n_d\} >$$

A hyper-edge is determined by its *label, source* and *target* and is a simple graph itself. *FG* is the composition of two graphs each being instantiations of relations. Therefore a composition operation ✣ on graphs and an instantiation ❁ of relations are defined. Besides the label of the relation the instantiation operation needs the source- and target-sequences. The composition operation needs the external nodes besides the component graphs. *FG* of example 1 is also given by the construction:

$$FG = ✣(\{❁(F, <n_a, n_b>, <n_x>),$$
$$❁(G, <n_x, n_c>, <n_d>)\}, \{ n_a, n_b, n_c\}, \{ n_d\})$$

[1] Functions are given by enumeration: $a \mapsto x$ means that the function maps a on x.

4.2 Formal semantics

The formal semantics of the hyper-graphs defines the behaviour, the tagged signal relation, specified by the hyper-graphs. *Tables* are useful representations of relations [8] and are used in the definition of the denotational semantics of our design representation in a similar way as described in [8] but with tagged signals as data-types. A composition $\bowtie$, projection $\Vdash$ and renaming ∞ operator on *tables* known from the *Table Algebra* of Brock [10] are used in the definition of this semantics. The external behaviour of a graph is the projection of the overall behaviour to the external nodes. The overall behaviour is the composition of the behaviours of the hyper-edges. The hyper-edges are instantiations of tagged signal relations to which their labels refer. This reference is specified by a (partial) function *lab2rel* of type *LABS* $\rightarrow$ *RELATIONS* and can be viewed as context definition. Similar labels in different graphs can refer to different behaviour. This is unambiguously defined by the use of *lab2rel* as parameter in the semantic function ***T*** that maps hyper-graphs to *tables*.

The external behaviour of the hyper-graph *FG* of example 1 is formally given by ([|.|] are the brackets used in case of semantic functions):

$$(\boldsymbol{T}\,[|FG\,|]\ lab2rel\,) \Vdash \{ n_a, n_b, n_c, n_d\}$$

where: $(\boldsymbol{T}\,[|FG\,|]\ lab2rel\,) =$

$$\boldsymbol{T}\,[|❁(F, <n_a, n_b>, <n_x>)\,|]\ lab2rel$$
$$\bowtie \quad \boldsymbol{T}\,[|❁(G, <n_x, n_c>, <n_d>)|]\ lab2rel$$

5. Transformations and correctness

Transformations are split into transformation-rules that can be proven at forehand and the application of these rules. Transformation-rules are defined as graph rewritings [8] and use graph morfisms that are structure preserving mappings between graphs.

Definition 3: Graph morfism

A ***graph morfism*** of an edge labelled directed hyper-graph *G* over *LABS* to an edge labelled directed hyper-graph *G'* over *LABS* is a tuple $m = <m_N, m_E>$ such that $m_N{:}N \rightarrow N'$, $m_E{:}E \rightarrow E'$ and $s' \circ m_E = m_N \circ s$, $t' \circ m_E = m_N \circ t$, $lab' \circ m_E = lab$ (see figure 1):

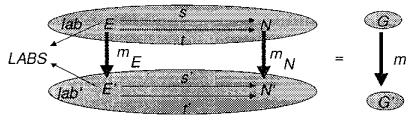


Figure 2: graph morfism

Definition 4: Transformation-rule

A *transformation-rule* is defined as a graph rewriting: a 5 tuple <*L,K,R,l,r*> of three IO ported edge labelled hyper-graphs *L, K* and *R* and two graph morfisms *l* and *r*.

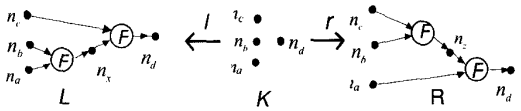


Figure 3: Transformation rule for associativity. *lab2rel(F)* has to be associative

The transformation-rule for associativity for example can be given textually by constructive parameterised definitions of *L,K,R,l* and *r:*

L=✤ ({❁(F, <n_a, n_b >, < n_x>), ❁(F , <n_x, n_c > , < n_d >)},{ n_a, n_b, n_c},{ n_d})

K=<{ n_a, n_b, n_c, n_d}, ∅, ∅, ∅, ∅, { n_a, n_b, n_c },{ n_d }>

R=✤ ({❁ (F, <n_b, n_c > ,< n_z>), ❁ (F , <n_a, n_z > , < n_d >)},{ n_a, n_b, n_c},{ n_d})

$l = r = id_K$ (= the identity function on the nodes of K)

Here it is important to realise that all graphs are determined besides isomorfy by graph morfisms.

A transformation on graph G is the application of a transformation-rule based on a graph morfism g from L to G that selects the subgraph of G on which the transformation is applied [8]. The transformation removes *g(L-l(K))* from G and replaces it by a copy of *R: h(R)* for some graph morfism *h.* For all elements *k* of *K,* nodes as well as edges, *g(l(k)) = h(r(k)).* This constraint describes the gluing of *h(R)* into *G - g(L-l(K)).* The correctness of transformations can be derived from the correctness of the transformation rules [8]. The transformation-rule <*L,K,R,l,r*> is correct when (definition. 5): the table, that describes the behaviour of R is included in the table that describes the behaviour of L (after renaming both tables to make them comparable). This means that the non-determinism may be reduced, but all input combinations for which behaviour was well defined need to have well defined behaviour afterwards.

For correctness proofs *PVS* of *SRI* [9] is used.

Definition 5: Correctness of transformation-rule

$(\boldsymbol{T}[|\ R\ |]\ lab2rel) \circ r \subseteq (\boldsymbol{T}[|\ L\ |]\ lab2rel) \circ l$

AND $((\boldsymbol{T}[|\ R\ |]\ lab2rel) \circ r) \Uparrow INn(K) \supseteq ((\boldsymbol{T}[|\ L\ |]\ lab2rel) \circ l) \Uparrow INn(K)$

6. Conclusions and future work

A flexible formal framework for transformational design is presented that incorporates a heterogeneous set of system models. Modern systems are compositions of different kinds of subsystems. Support of this heterogeneity distinguishes our approach.

A graphical design representation is used and transformations are based on *graph rewriting.* IO ported edge labelled directed hyper-graphs describe structure. The behaviour definition is given by a denotational semantics in which the semantic algebra is the table algebra of Brock [10] and the data-structure is the tagged signal defined by Lee [6]. Tagged signals give flexibility to handle different kinds of systems and timing abstraction that is needed in high-level synthesis. A constructive definition of graphs is used to define transformation rules. The formal framework is suited for model refinement. A correctness definition for transformation-rules that can handle behavioural refinement as well as structural refinement is given. Time- and data-refinements often refine the specification and need to be made explicit to obtain design correctness. These refinements will be given more attention in future.

7. References

[1] R. Kumar, Ch. Blumenröhr, D. Eisenbiegler and D. Schmid, *Formal Synthesis in Circuit Design – A Classification and Survey*, in M. Srivas and A. Camilleri (Eds.) *Formal Methods in Computer-Aided Design*, LNCS 1166, (Springer-Verlag, Berlin, 1996) pp 294-309.

[2] Ch. Blumenröhr, D.Eisenbiegler, *An Efficient Representation for Formal Synthesis,* Proceedings ISSS'97, Sept. 17-19, 1997, Antwerp, Belgium, pp 9-15.

[3] R. Sharp and O. Rasmussen, *The T-Ruby Design System*, Formal methods in System Design 11, pp 239-264 (1997).

[4] Z.Peng and K. Kuchcinski, *Automated Transformation of Algorithms into Register-Transfer Level Implementations,* IEEE Trans. on CAD Vol.13, No.2. Feb. 1994, pp150-166.

[5] P.F.A. Middelhoek, *Transformational Design: An architecture independent interactive design methodology for the synthesis of correct and efficient digital systems,* Ph.D. Thesis University of Twente, April 1997.

[6] E.A. Lee and A. Sangiovanni-Vincentelli, *A Framework for Comparing Models of Computation,* March 12, 1998 (Revised from Memorandum UCB/ERL M97/11, University of California, Berkeley, CA 94720, January 30, 1997). (http://ptolemy.eecs.berkeley.edu/papers/98)

[7] A. Girault, B. Lee, and E. A. Lee, *Hierarchical Finite State Machines with Multiple Concurrency Models*, April 13, 1998 (revised from Memorandum UCB/ERL M97/57, University of California, Berkeley, CA 94720, August 1997)(http://ptolemy.eecs.berkeley.edu/papers/).

[8] C. Huijs, *A Graph Rewriting Approach for Transformational Design of Digital Systems,* In: Proceedings 22nd EUROMICRO conference, 2-5 September 1996 Prague, Czech Rep., pp 177-184.

[9] S.Owre, N. Shankar and J.M. Rushby, *Userguide for the PVS specification and Verification system*, Computer Science Laboratory, SRI, Menlo Park, CA, Feb. 1993.

[10] E.O. de Brock, *Foundations of Semantic Databases*, Prentice Hall, 1995.