The ForSyDe Methodology

Ingo Sander
Royal Institute of Technology
Stockholm, Sweden
ingo@imit.kth.se

Abstract

This paper gives an overview of the ForSyDe (Formal System Design) Methodology, which targets the design of System-on-Chip applications. Design starts with a system model, that is based on a synchronous computational model. Processes are constructed by means of formally defined process constructors in order to allow the formal refinement of the system model by the application of design transformations. Each process constructor has also a hardware and software semantics, which is used for the translation of the refined model into hardware and software.

1 Introduction

A SoC (System-on-a-Chip) will be able to integrate more and more heterogeneous computing resources. They can be dedicated (logic), programmable (processors and DSP), configurable (FPGAs), passive (memory) or most likely a mixture of these. Keutzer et al. discuss system-level design in [4]. They point out, that “to be effective a design methodology that addresses complex systems must start at high levels of abstraction”. And they “promote to use formal models and transformations in system design so that verification and synthesis can be applied to the advantage of the design methodology” and believe that “the most important point for functional specification is the underlying mathematical model of computation”. These arguments strongly support the ForSyDe methodology [8], which provides a modeling technique that results in an abstract and formal system model. This system model is functional and based on a synchronous computational model. The methodology provides a library of process constructors that are used to construct processes. They implement a synchronous computational model and have a structural interpretation in hardware and software. Based on these properties the ForSyDe methodology offers a refinement and a synthesis technique to first refine a system model into a more efficient functional form and then synthesize it into a hardware/software implementation.

2 Related Work

Edwards et al. [3] give a comprehensive overview about models of computation. The ForSyDe system model is based on the perfect synchrony hypothesis, the base for the family of the synchronous languages. According to Benveniste and Berry “the synchronous approach is based on a relatively small variety of concepts and methods based on deep, elegant, but simple mathematical principles” [1]. The basic synchronous assumption is, that the outputs of the system are synchronized with the system inputs, while the reaction of the system takes no observable time. Apparently a real physical system cannot comply with the perfect synchrony assumption. But a model based on the perfect synchrony assumption would behave exactly the same as the real implementation if the physical system is fast enough.

Roeke [7] used the functional language Haskell [11] to model digital signal processing applications. Similarly to us he modeled streams as infinite lists and used higher-order functions to operate on them. Finally, semantic-preserving methods were applied to transform a model into a more efficient representation. This representation was not synthesized to hardware or software. Lava [2] is a hardware description language based on Haskell. It focuses on the structural representation of hardware and offers a variety of powerful connection patterns. Lava descriptions can be translated into VHDL and there exist interfaces to formal method tools. Hardware ML (HML) [5] is a hardware description language based on the functional programming language Standard ML. Though HML uses some features of Standard ML, such as polymorphic functions and its type system, it is mainly an improvement
of VHDL - there is a direct mapping from HML constructs to the corresponding VHDL constructs. The parallel programming community has used functional languages to derive parallel programs from a functional specification [10]. They use higher-order functions to structure a problem. This formulation is then transformed into an efficient implementation for a selected parallel architecture.

Compared with approaches based on languages like VHDL or SystemC, ForSyDe starts at a higher abstraction level, due to (1) a fully deterministic communication mechanism based solely on message passing whose timing is governed by the synchronous model, and (2) the avoidance of arbitrary control dependences allowing to fully exploit parallelism in the implementations which is only constrained by problem inherent data dependences. The abstract functional system model is first refined inside the functional domain [12] into a more detailed and efficient implementation model, which is then further translated into hardware and software descriptions.

3 Design Process

![Figure 1. The ForSyDe Design Process]

Figure 1 gives an overview about the ForSyDe design process. System design starts with the development of a formal system model, which is purely functional and based on the perfect synchrony hypothesis. The formal nature of the model allows for stepwise design refinement through formally defined design transformations which either are semantic preserving or introduce a design decision, such as constraining the size of an ideal and infinite buffer. Processes are constructed by formal process constructors. These process constructors have a hardware and software semantics, which allow the translation of a refined system model into a mixed hardware/software implementation.

4 System Model

The system model reflects the design principles of the ForSyDe methodology. In order to allow for formal design on a high abstraction level, the system model has the following characteristics:

- It is based on a synchronous computational model, which cleanly separates computation from communication.
- It is purely functional and deterministic.
- It uses ideal data types such as lists with infinite size.
- It uses the concept of well defined process constructors, which implement the synchronous computational model.
- It is based on a formal semantics and can be executed using the functional language Haskell [11].

The system model abstracts from implementation details, such as buffer sizes and low-level communication mechanisms. This enables the designer to focus on the functional behavior on the system rather than structure and architecture. This abstract nature leaves a wide design space for further design exploration and design refinement, which is supported by our transformational refinement techniques (Section 5). The system model consists of concurrent processes which communicate synchronously by means of signals. A signal is a set of events, where each event has a tag and a value. The tag is used to denote the order of events. As we deal with synchronous systems the same set of tags is used in each signal. It follows that an output event has the same tag as the corresponding input events.

![Figure 2. Signals and Processes]

Figure 2 illustrates the modeling of signals and the behavior of processes. During the event cycle $n$ a process processes the events of each signal with the tag $n$ and outputs the result at the same tag $n$. We denote a signal $s$ with \{$s_1, s_2, \ldots$\}, where the tag is given by the position in the signal. The ForSyDe methodology obliges the designer to construct processes by means of well defined process constructors. A process constructor takes combinatorial functions and values as arguments and constructs a process.

Figure 3 shows a process constructed by the combinatorial process constructor $zipWithSY_n$. The concept
of process constructors leads to a clean separation between synchronization and computation. The process
constructor implements the synchronization according to the synchronous model, while the computation is
expressed by the combinatorial function $f$. We can also see from Figure 3, that such processes have a clear
hardware and software semantics. In hardware it is implemented by a combinatorial function with $n$ inputs
and one output, while in software it is implemented as function with $n$ parameters.

The process constructor $delaySY_n$ forms the base for more complex sequential process constructors. This
process delays the output signal $n$ event cycles by inserting initial values $m_0$ at the head of the input signal.
It is implemented as a bank of registers in hardware.

Figure 4 illustrates the process constructor $mealySY_n$. It is composed of two $zipWithSY_{n+1}$ and one
$delaySY_1$ process constructors. Processes based on $mooreSY$ model a finite state machine of Mealy
type. It takes two functions $f$ and $g$ to model the next state and the output decoder and a value $m_0$ for
the initial state as arguments.

5 Refinement of the System Model

![Diagram](image)

Figure 5. Refinement of the System Model

The next step in the design process is the step-wise refinement of the system model $S_0$ by well defined
design transformations $T_i$ into a final implementation model $S_n$ as illustrated in Figure 5.

There are two classes of transformation techniques:

Semantic Preserving Transformations A semantic preserving transformation does not change the
meaning of the model, i.e. the transformed model behaves in the same way as the original model. Semi-
getic preserving transformations are mainly used to optimize the model for synthesis.

Design Decisions Design Decisions change the meaning of the system model. A typical design
decision is the refinement of an infinite buffer into a fixed-size buffer with $n$ elements. While such a
design decision clearly modifies the semantics, the transformed model may still behave in the same
way as the original model. For instance, if it is possible to prove that a certain buffer will never
contain more than $n$ elements, the ideal buffer can be replaced by a finite one of size $n$.

We refer to [12] for a more detailed discussion about refinement through design transformations. However,
we want to point out, that our approach leads to a documented and transparent refinement process which
is repeatable. In addition the functional system model allows for the optimization of processes over subsystem
borders, since

- processes can be easily moved over process borders
- processes can be combined and their combinatorial functions can be optimized

The refined system model is still a functional system model. However, because of the introduction of design
decisions and the application of optimizing semantic preserving transformations the refined model is much
closer to the final implementation.

6 Code Generation

The refined system model is the starting point for the final phase of the design process. The system model
is partitioned into hardware and software and then translated into the corresponding VHDL and C code.
This translation is based on the hardware and software semantics of the process constructors. For each
process constructor exists a VHDL and C code template. These templates contain dummy functions that
during the code generation process are replaced by a synthesized version of the original combinatorial func-
tion. The template also contains dummy data types and dummy values, that are replaced by their synthe-
sized counterpart. Figure 6 shows the VHDL-template
use work.fpackage.all;

entity zipWithSY2T is
  port(a: in type1;
       b: in type2;
       y: out type3);
end zipWithSY2T;

architecture Comb of zipWithSY2T is
begin
  y <= f(a, b);
end Comb;

Figure 6. VHDL Template for zip WithSY

for zip WithSY. In case the function f is an addition operation for 32-bit integers, the package fpackage will define the data types type1, type2 and type3 as integers and the function f as + operation (Figure 7). The synthesis technique for ForSyDe for hardware and software is described in [9, 6]. This report presents the synthesis technique and exemplifies it by the synthesis of the system model of a digital equalizer into a hardware (behavioral VHDL) and software description (sequential C).

7 Conclusion

This paper gives an overview over ForSyDe which is a formal method for system design. Based on a formal model that is based on the perfect synchrony hypothesis and the concept of process constructors it allows the stepwise refinement by design transformation. Since each process constructor also has a hardware and software semantics the refined model can be translated into a hardware and software description.

Today the main concepts of the ForSyDe methodology are understood and formulated. We have also used our methodology on several case studies. Today the main focus of our work is on the refinement method and the development of tool support for the ForSyDe methodology.

References