The Metamodeling Approach to System Level Synthesis

Wolfgang Ecker, Michael Velten, Leily Zafari
System-Level and Verification
Infineon Technologies
Munich, Germany
<first name>.<last name>@infineon.com

Ajay Goyal
System-Level and Verification
Infineon Technologies India
Bangalore, India
Ajay.Goyal@infineon.com

***Abstract*— This paper presents an industry proven Metamodeling based approach to System-Level-Synthesis which is seen as generic design automation strategy above today's implementation levels RTL (for digital) and Schematic Entry (for analog). The approach follows a new synthesis paradigm: The designer develops a simple domain and/or design specific language and a smart tool synthesizing implementation level models according to its needs. The overhead of making both a tool and a model pays off since the tool building is automated by code generation and reuse, both based on Metamodeling techniques. Also the focus on owns demand keeps development costs low. Finally, specification data is utilized. I.e. the domain specific language simplifies to a document structure as a table. This keeps also modeling effort low since specification content is used and no model need to be built. Furthermore, increases design consistency and thus decreases debug time. Using these concepts, single design steps have been speed up to a factor of 20x and implementations of chips (specification-to-tapeout) have been speed up to a factor of 3x.**

Keywords—system level synthesis; metamodeling; code generation

I. INTRODUCTION

Today's synthesis technologies are still dominated by RTL synthesis, which started its wide introduction about 25 years ago. Several other synthesis technologies starting at modeling levels beyond RTL – which we name system level synthesis in this paper - have been introduced since then. Examples are high-level-synthesis, rule based synthesis, processor synthesis, co-processor synthesis, or state diagram synthesis. None of them made it to such a generic approach as RTL and provides productivity improvement for small domains only. In addition, each of the tools has a different input language and coding style.

This is no surprise, since the design space beyond implementation level is very huge, continuously grows with the "More-than-Moore" diversification, and requires plenty of specific things. Also the design process above RTL is very heterogeneous concerning supported formats, intended abstraction, target architectures, interfacing modules or optimization targets.

To provide a generic automation solution above implementation level, *metagen* a metamodeling and code generation technology and methodology was developed mainly in Python at Infineon in the last 4 years.

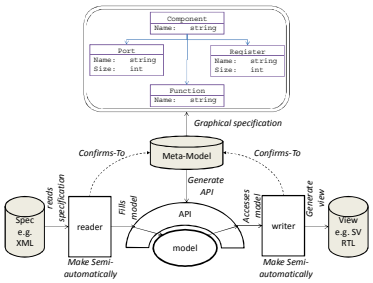
978-3-9815370-2-4/DATE14/©2014 EDAA

It has been successfully used already from the first days on and it has been continuously improved and highly beneficial applied in many Infineon designs and design areas.

In the next parts, terminology and technology of meta-modeling is introduced. Afterwards, the taken approach is elaborated in more detail and application aspects are discussed.

II. METAMODELING TERMINOLOGY AND TECHNOLOGY

Metamodeling follows a specific, partially automated and structured approach – namely a clear separation of model and view as shown in subsequent picture.



The target of generation is a so called view, which may be e.g. RTL code or a schematic description – since we want automation above today's implementation - but also XML, documentation, or any other documents occurring in the design phases. The view is generated by a so called generator, which is often a template engine. This template engine renders a so called template, a mix of target code, substitutions, and generation pragmas.

In order to generate the view according to current design needs – often the specification - the template engine has to retrieve the required data appropriately. This data is stored in a structured way in a so called model.

The structure of the model is defined in a so called meta-model. Here "meta" means above and metamodel means a model above or more abstract than a model. A metamodel is also called a model of a model.

The data of the model is read from a specification, parsed from any other document, imported from a description formulated in a so called domain specific language, or entered through a generated GUI.

The model and the model's API (to set and retrieve data) must be compliant to the metamodel, so the model's API is automatically generated from the metamodel. Since writer and

reader access the model via the generated API, also they comply with the metamodel.

But there is no single technical aspect that makes our approach that powerful. From the implementation standpoint object orientation as such is mandatory for structuring data and providing clear interfaces to each chunk of data. The polymorphic underlying language allows focusing on the design domain when doing modeling. The interpreted language allows for fast edit-compile-execute important to a fast rampup of the new environment. Interestingly, performance of the interpreted language is mostly sufficient. An easy integration of the template engine with the implementation language of the framework allows mixing both approaches for code generation, e.g. using templates for formatting statements and a programming language for formatting tokens. We use the Mako template engine in our approach.

Another important aspect is the openness of the framework supported by powerful introspection and aspect oriented capabilities that allow designers – i.e. metamodeling users – to adapt the solution for their needs without changing the core engine. Also the following three implementation strategies showed to be beneficial for the overall approach:

First, required improvements are mostly implemented in the *metagen* framework instead of in a specific model only. So it is available for all generators built with *metagen*.

Second, the *metagen* framework makes heavy use of generation – e.g. the model's API is generated – the application of the metamodeling and code generation technology in the framework itself contributes to a low development effort.

Third, by providing an own reader, metamodel, and generator, the smooth integration in own's design flow can be easily achieved – in contrast to existing tools, which require not neglectable effort to make them fit into a specific design flow. Even more important, own readers allow to setup a single source strategy for all implementation data and thus improves consistency from the beginning.

III. Application Methodology

Not only the technology, but also the application strategy is important for the success of the approach: First to mention is the radical domain or even design specific approach as already mentioned in the introduction. The metamodel, the reader, and the generators implement only things that are absolutely needed for the specific application. This keeps implementation time low and allows a very focused and efficient verification. First, applications are ready, in use, and provide benefits for the designer often in one day. When useful, automation scope is incrementally increased. So, the usage of the technology pays off from the first day.

It is worth mentioning that in our approach, the domain experts (i.e. designer, verification engineer, etc.) build their own metamodel, reader and writer – or are at least heavily involved in building it. This makes sure, that the right aspects, bringing the best benefit, are automated. Further on, it reduces domain specific knowhow at the *metagen* framework side for the price of convincing designers to get in touch and use the metamodeling approach.

Summarizing the application efforts and techniques, we estimate that the taken approach for building the smart domain specific tool utilizing the described technologies is at least 20x faster than the approaches used today for building generic applicable EDA software. This efficiency essential for the new paradigm: Making both a model and a specific synthesizer.

IV. Application Experience

So far *metagen* has been applied for synthesizing digital hardware, analog hardware and firmware area, covering also the interfaces between the areas. In addition, power and test issues have been approached. Also verification and documentation views have been addressed. Continuously, meta- modeling is applied in new applications and application areas as timing and or reliability issues.

But aren't there tools out utilizing metamodeling features such as IP-XACT tools? Of course, if they fit the needs, they should be applied! But often they do not fit, as illustrated in the following example utilizing register description: The tools offer a GUI to enter the data, but the specification already covers the features and exists in Framemaker, Word, Excel, etc. and of course following a specific document structure. Either, the data has to be re-typed or a translator to IP-XACT has to be built since direct access to the tools data is hard to achieve if at all.

Specific features for example register access control, retention for power down, or interface to control hardware also accessing the registers might be needed. Then, the intermediate has to be somehow patched to capture the data or a parallel model has to be developed. Then the generator – if possible has to be recoded. But also specific requirements for coding styles may require recoding of the generator.

As the example shows, the metamodeling approach might be even efficient when already an EDA tool exists that does not exactly match the needs. And there are many areas for synthesis (*metagen* supports at the moment over 80) that are not supported by EDA tools at all.

Summary and Outline

We presented a promising, novel, and industry proven approach for synthesis above RTL – so to say a new generic approach for system level synthesis. The approach utilizes the metamodeling and code generation technique known from SW domain since about 10 years. Frameworks such as the open source Eclipse Modeling Framework or the commercial MetaCase are ready to be used for adopting the described methodology even if we are sure that our Python based Metagen environment provides here and there substantial benefits.

Metamodeling and code generation can be seen as a system level tool synthesis synthesizer, so to say a synthesizer of a synthesizer or a "meta synthesizer". Even if already successfully used, this approach opens a new wide R&D field, e.g. for "meta algorithms" applicable for a set of metamodels.

This technology has been used in many kinds of Infineon products in the area of energy efficiency, mobility and security. Altogether, the technology has been used in over 50 design projects automating over 500 single design steps at Infineon so far. Savings of up to 95% in single design steps and up to 70% in the overall implementation of a chip has been measured.