Complementing EDA with Meta-Modeling and Code Generation

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*—Automation is a key to design technology. As designs are getting more complex day by day, productivity, quality, efficiency and consistency of data become more important. Also as things get more sophisticated, human errors and mistakes becomes intolerable. In order to achieve this efficiently, automation of design steps becomes crucial. EDA tried to solve this problem by creating tools to automate the design flow. But EDA often has to cater large number of customers so they cannot make focused tools for particular domain problem. Also development cost of such focused tools will be very high and will not be affordable to design companies.**

***In this paper we describe a model driven software technology approach which not only allows design companies to design their own automation but create sophisticated tools with very little effort. It also allows them to experiment with their data and quickly change the solution along with the specification. The idea with this domain specific approach is not to replace EDA and start creating all tools in design houses. But it is to complement EDA by helping reduce tool entry barriers by generating input tool views and commands from the specification.**

***The infra-structure can also be used to create more complex tooling based on reusing existing building blocks. The other reason which makes this infra-structure easy to use is the code generation feature from a UML model – as usual in model driven engineering - which reduces almost 80-90% effort for creating a new tooling. Further, features like generating automated transfer of data transfer, helps designers to reduce their effort and achieve better efficiency in their design steps.**

Keywords—metamodeling; code generation, design automation

I. INTRODUCTION

About 25 years ago, EDA industry developed tools that successfully replaced many in-house tools at semiconductor companies and system houses. It was simply better and cheaper, to develop tools once and apply these tools in many development sites worldwide. Since then, EDA evolved continuously providing new solutions in areas as deep submicron or coverage driven verification to name only some.

It's obvious; EDA cannot do all specific things for everybody but has to focus on common approaches. Therefore it's no wonder, that no widely usable synthesis above RTL synthesis is available. 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. Therefore, today only point synthesis tools beyond RTL exist covering only a small design area.

To provide a generic automation solution above implementation level, *metagen* a meta-modeling and code generation technology and methodology was developed at Infineon in the last 4 years. 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. One success factor of the *metagen* methodology is following a domain – sometimes even a design – specific automation approach which is orthogonal to the EDA strategy. Also, automation is done by the designer and covers exactly the indented application including only the generality needed.

In the next parts, terminology and history of meta-modeling is introduced, since it is not a mainstream technology in hardware design yet. Afterwards, the taken approach is elaborated in more detail and application examples are given.

II. TERMINOLOGY AND HISTORY OF META-MODELING

The target of meta-modeling and code generation is the replacement of manual entry by generation of the implementation formats and tools driving scripts as e.g. RTL for digital design, schematic for analog design, C for firmware, or TCL for UPF or other tool specific automation.

Automating by specific scripts (see e.g. [10] as one of many examples) as such is not new, but meta-modeling follows a specific, partially automated and structured approach – namely a clear separation of model and view:

- Target of generation is a so called view, which may be one of the formats mentioned in the introduction – 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. A 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 retrive data required appropriately. This data is stored in a structured way in a so called model. An API – which is generated in our approach – provides the generators access to the models.

The terminology is derived from SW technology, since also software (e.g. coded in C++), firmware (e.g. coded in C), and documentation (e.g. represented in XML) can be generated. Therefore, RTL-Models or other hardware models are treated as views in this terminology.

- 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 with a GUI

- The structure of the model is defined in a so called meta-model. Here "meta" means above and meta-model means a model above or more abstract than a model. A meta-model is also called a model of a model. The model and the model's API (to set and retrieve data) must be compliant to the meta-model, so the model's API offers to be automatically generated from the meta-model. Since writer and reader access the model via the generated API, also they comply with the meta-model.

- Since the meta-model requires formalisms for it's definition, model driven engineering also knows the term meta-meta-model, which exactly describes this formulism.

Figure 1 shows an overview on that terminology derived from OMGs MOF [5]. The different level from view to meta-metamodel are usually named M0 to M3.

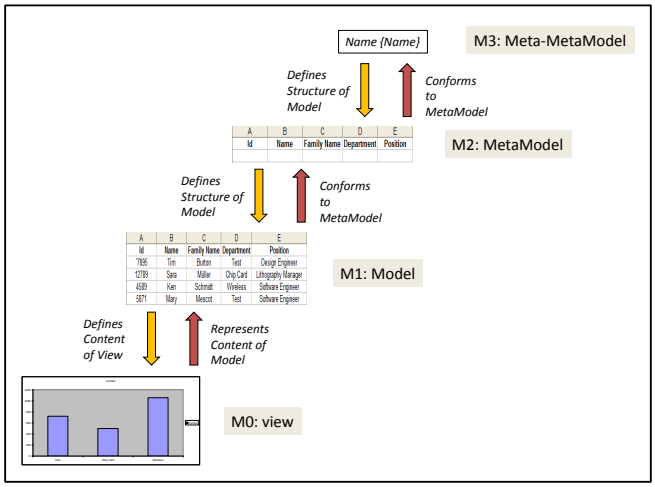

Figure 1: Overview of Abstraction Levels

In the example, the view describes a chart visualizing some data. This chart is one possible view of the model holding that data. The structure of the model – so to say to meta model – is defined by the headline. Last but not least, the headline is nothing else than a non empty sequence of names (**`Name { Name }`**), as defined in the meta-metamodel utilizing EBNF formula syntax.

Figure 2 gives an overview on the interaction of the pieces. The design automation path (specification to implementation) is from left to right and the meta-model underlying automation (meta-model API) is from top-to-down.

It is worth mentioning, that the idea of structuring (hardware) design data or defining (hardware) design data in a structured way has been evolved in several technologies as entity relationship diagram, UML modeling [6], XML technology [7], and finally metamodeling.

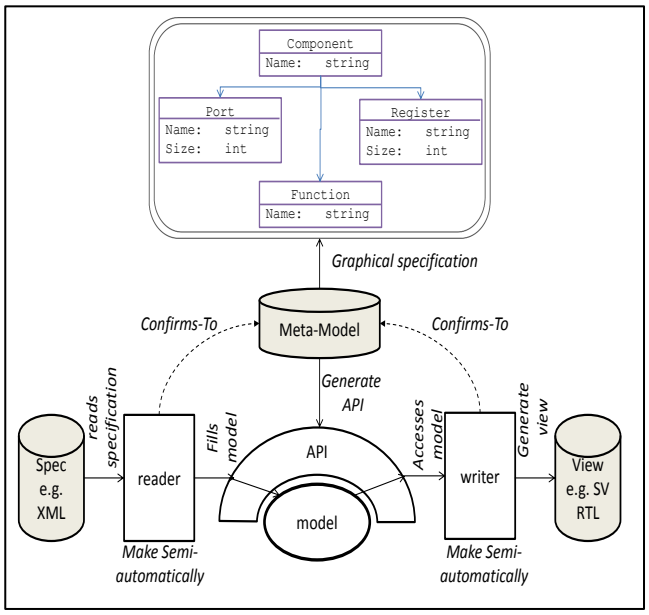

Figure 2: Flow diagram of a meta-modeling environment

III. TAKEN APPROACH

Initial point was the search for automation in typing implementations as VHDL models or C programs. A solution was established by a cross functional approach i.e. by applying a relatively modern SW technology – meta-modeling – to the implementation automation of embedded systems.

But there is no single aspect that alone enables our meta-modeling framework metagen to act as a core technology for domain specific design automation. From the implementation standpoint the use of Python showed to be beneficial, since it provides many features useful for the own implementation of a meta-modeling framework:

- 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. No inheritance needs to be introduced when polymorphism is needed.
- The interpreted language allows for fast edit-compile-execute important to fast rampup 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 [8] in our approach.
- Also important is the openness of the framework supported by powerful introspection capabilities in order to allow designers – i.e. meta-modeling users – to adapt the solution for their needs.

Nevertheless, the metamodeling and code generation take some benefits of our implementation strategy, the overall approach is not bound to metagen as such.

- So, the eclipse modeling framework (EMF)[2] provides similar capabilities as metagen. Due to its eclipse nature, EMF is Java based, which impacts some overhead in implementation due to Java's strict object oriented approach. Also the template engine is not that intuitive than the Python based template engine Mako.
- A commercial solution is provided by MetaCase [3]; the tool is called MetaEdit. Here, a graphical domain specific language can be developed. From interest are the reported use cases from SW development, e.g. software for fish pound management, since they show the wide applicability of the technology. Metacase claims a productivity improvement using their technology in SW domain by up to a factor of 20x.

But not only the underlying engine is important, also the following three implementation strategies showed to beneficial for the overall approach:

- Required improvements are mostly implemented in the framework but in the model. So it is available for all generators build with metagen. Nevertheless, we had in the last four years only two major releases of the framework.

So, the framework is very generic but the applications are much focused.

- Also the metagen framework makes heavy use of generation – e.g. the model's API is generated – the application of the meta-modeling and code generation technology in the framework itself contributes to a low development effort.
- By providing an own reader, model, and generator, the smooth integration in owns design flow can be easily achieved – in contrast to existing tools, which require not neglectable effort to make them fit in owns design flow (please see EDA tool issue in the introduction).

Even more important, own readers allow to setup a single source strategy for all implementation data and thus improves consistency from the beginning.

IV. APPLICATION STRATEGY

But not only the technology, also the application strategy important for the success of the approach:

- First to mention is the radical domain specific approach as already mentioned in the introduction. The meta-model, the reader, and the generators implement only things that are absolutely needed for the 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 some days. When useful, automation scope is incrementally increased. So, the usage of the technology pays off from the first day.
- The next thing is that the taken approach does not target full automation. Supported by the template based code generation approach, often an 80% automation leaving 20% manual work is appropriate. Let's say gaining 80% productivity improvement using automation, an overall improvement of 64% is still achievable.
- Last to mention that in our approach, the domain experts (i.e. designer, verification engineer etc) build model, reader and writer – or are at least heavily involved in building it. This makes sure, that the right things and the things bringing the best benefit are automated. Further on, it reduces needed domain specific knowhow at the metagen framework side for the price of convincing designers to get in touch and use the meta-modeling approach.

V. INHOUSE DESIGN APPLICATIONS USING METAGEN - AIDING EDA TOOLS

Many design applications have been executed inside the organization which helps to get a good start point before start using EDA tools. Many applications are in classical HW, SW and HW/SW areas as generation of structure, control flows or declarations. (see e.g. [9])

But also many other applications with the generation of more hardware related items are automated with metagen. One such application is IO PAD area estimator shown in Figure 3.

The IO pad designers create these run set files which they run with inhouse spice simulator and dump netlist. Using this netlist, they plan the layout without any idea about the ideal area that must be taken by the netlist. So, the final outcome might be error prone and efficiency on the size of the IO pad depends on the experience of the designer since the final result is not known at that point of time. So using Metagen an application had been created and flow has been designed to estimate the area before they have done the final layout in the EDA tool.

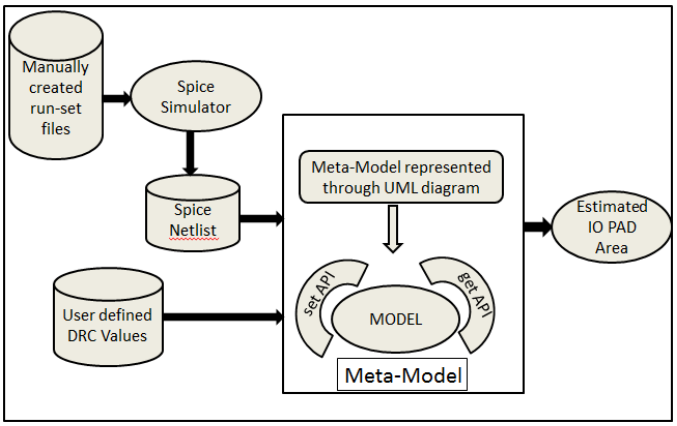

Figure 3: IO Pad Area Estimator

a) In this application, a meta-model, Figure 4, is created using the data in the netlist for defining the relationships. This meta-model generates the code for the model.

b) In next step, a reader was written which reads the netlist and user specified parameter file which contains DRC (Design Rule Check) rules.

c) Also, a writer is written on the model which does the calculation of the total IO pad by adding each transistor and sub-circuit area.

So from a new user stand point, he has to pass the netlist and user parameters to the tool and he gets estimation on the final area of the IO pad. This not only gives user a realistic while doing the circuit layout but also confidence in their work once the target is achieved.

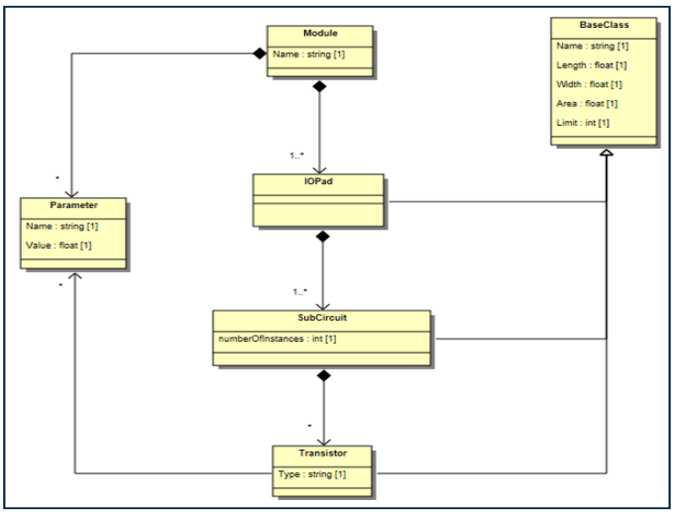

Figure 4: Meta-Model for IO Area Estimator

Another, utility created inhouse which is worth mentioning here is IO Library Information Collector shown in Figure 5. This inhouse tooling is used for IO generation, It needs information from the created schematic, layout and connectivity, logical equation etc in form of user written text files. These text file format is error prone and results into several cycles of effort to get them right. Wrong information would cause a garbage in / garbage out behavior of the IO generator. Hence a flow was created using Metagen which ease out user pain in creating the text file.

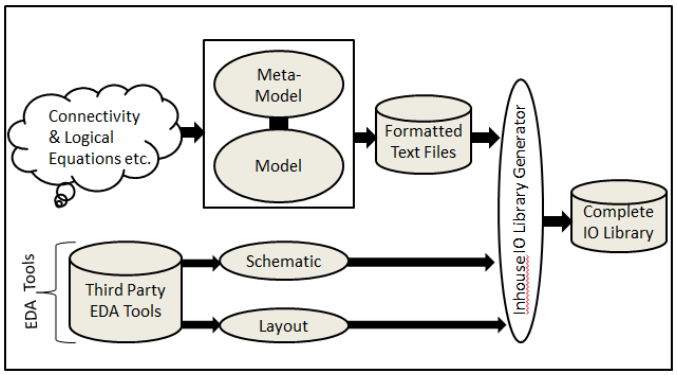

Figure 5: IO Lib Information Collector

a) The IO generator needs schematic, layout and connectivity information, logical equations etc in another text file format to generate the complete IO PAD.

b) The schematic and layout are created in other EDA tool. For the text file information a meta-model was created using Metagen. The generated GUI from the meta-model, Figure 6a, 6b, allows user to provide IO Cell characterization and other information. A generator was written to generate these text files.

c) These text files along with the layout and schematic are read by the inhouse IO generator tool to generate complete IO library.

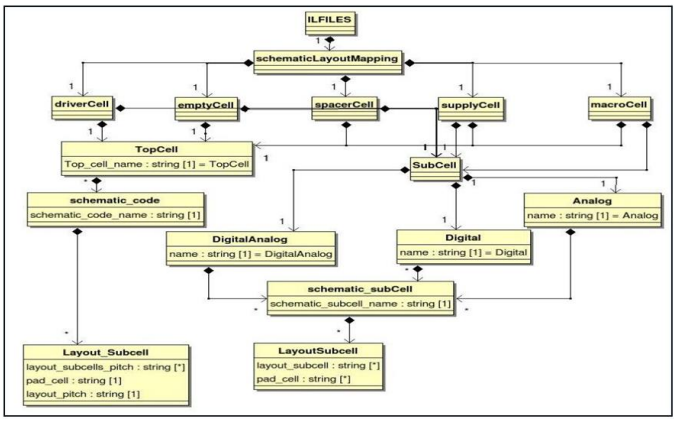

Figure 6a: Meta-Model for IOCell Information

Major benefit of this approach is the support of a more formal approach allowing early consistency checks.
In both these applications, as we see, we are not re-creating EDA tools and re-invent the wheel. But the mechanisms provide an easy way to create and maintain extensions which might be useful to make the EDA tool usage more effective and efficient, especially by reducing the possibility of error prone entry to it.

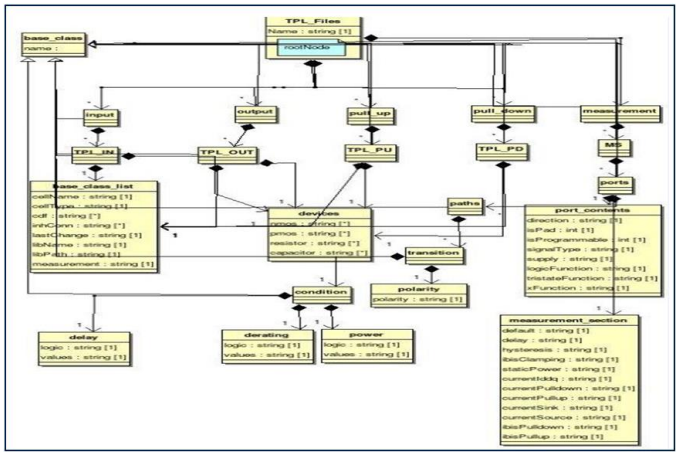

Figure 6b: Meta-Model for characterization of IO Cells

VI. DESIGN APPLICATION EXPERIENCE

Interestingly, designers accept learning meta-modeling, since it is also a modeling task (designer's daily work) and since meta-modeling is supported by the graphical capture of the meta-model. Only in the beginning, meta-modeling experts and domain experts (i.e. designers) sit together, plan, and discuss the modeling approach.

The other important thing to mention is the high efficiency we get by applying all software-, methodology-, and design application strategy measures summarized above. We estimate that the taken approach is at least 20x faster than the approaches used today building EDA software.

The circle closes, since this efficiency enables developing own and specific automation in digital hardware, analog hardware and firmware area covering also the interfaces between the areas. Also verification and documentation views have been generated. 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. Continuously, meta modeling is applied in new applications and application areas.

VII. Comparison with existing Tools

First of all, applying meta-modeling in hardware design does not require our metagen toolbox. As said, there are other tools out as the open source Eclipse modeling framework [2] and the Meta-Case tools. We only think that our environment is a bit better tuned for hardware design. Also full featured UML tools supports meta-modeling and can therefore be used for the described techniques.

But there aren't many tools out that directly utilize meta-modeling features. First of all to mention are UML tools supporting profiles for embedded systems as SysML [11] and Marte [12]. Of course, these tools shall be applied, but often, they lack features as described below for IP-XACT tools [4]. Let's consider the utilization of register descriptions in real live designs:

- 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 using the EDA tool's GUI or a translator to IP-XACT has be be built.
- Specific features as register access control or retention for power down are needed. Then, the intermediate has to be somehow patched to capture the data or a parallel model has to be developed.
- Code shall be generated from the extended features, following a specific coding style or in a specific format. Then the generator – if possible has to be extended – or if no full access is given, it has to be completely recoded.

To avoid a wrong impression, IP-XACT is heavily used at Infineon for what it was built for: External IP integration.

As the example shows, the meta-modeling approach might be even more efficient when already an EDA tool exists but specific requirements have to be fulfilled. And there are many generation issues (metagen supports at the moment over 80) that are not supported by EDA tools at all since they are very design and/or domain specific. In these cases, meta-modeling complements EDA and gives value to the designer overall.

Summary and Outline

We presented a promising, novel, and industry proven approach for automation primarily above implementation. The approach utilizes the meta-modeling and code generation technique known from SW domain. It uses orthogonal concepts to today's EDA world – domain specific vs. generic solutions -and thus complements EDA tools in today's design automation.

Even if meta-modeling is known for over a decade now – and underlying concepts even longer – there are still many challenges in applying meta-modeling and code generation in the automation of embedded system design. To name only some, the challenges include modeling of physical aspects, modeling of functionality, or versioning of meta-models, models, and view.

The intend of the paper is to make the technology knowledgeable to the design community and act as a starting point of further discussions and developments of a technology, which we might call "meta-modeling and code generation of embedded systems".

References

[1] J.-M.. Bergé, O. Levia, and J.. Rouillard, "Performance and Information Modeling," Kluwer Academic Publishers, 1996

[2] D. Steinberg, F. Budinski, M. Paternostorno, E. Merks, "EMF: Eclipse Modeling Framework" 2009

[3] MetaCase: "Domain Specific Modeling with MetaEdit+," http://www.metacase.com/de/

[4] IEEE: "1685-2009 – IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tools Flows," 2010

[5] M. Brambilla, J. Cabot, M.Wimmer, "Model-Driven Software Engineering in Practice", Morgan&Claypool Publishers, 2012

[6] D. Pilone, N. Pitman, "UML 2.0 in a Nutshell," O'Reilly, 2005

[7] E. T. Ray, "Learning XML," O'Reilly, 2009

[8] Mako Templates for Python: http://www.makotemplates.org/

[9] R. Findenig, T. Steininger, T. Leitner, "Combining several Metamodels for the Generation of Control strucutres in Hardware", MeCoEs2012, Tampere, 2012

[10] Schneider, C.; Ecker, W.: "A parallel/serial trade-off methodology for look-up based decoders". In Proceedings of the Design Automation Conference (DAC '97), Anaheim, USA, (6 1997).

[11] Holt, J.; Perry, S.: "SysML for Systems Engineering (Professional Applications of Computing)", The Institution of Engineering and Technology (May 2008)

[12] Selic, B.; Gerard, S.: "Modeling and Analysis of Real-Time and Embedded Systems with UML and MARTE: Developing Cyber-Physical Systems", The MK/OMG Press