



Gabe on EDA

Gabe on EDA | A consulting organization serving the EDA industry

Best Practices for Adopting Model-Based Design in Electronic System Development

By Gabe Moretti

About the author

Gabe Moretti is a recognized expert in all aspects of the EDA industry. Gabe has over thirty years of experience developing EDA tools spanning the range from design capture to chip layout. Gabe has also worked on the development of numerous industry standards including EDIF, VME, VHDL, and Verilog. He has held senior management positions with EIS Modeling, HDL Systems, and Intergraph/Veribest.

From 2000 Mr. Moretti has been covering the EDA industry as a writer and editor first with EDN Magazine. Now with Gabe on EDA he edits EDA DesignLine an electronic publication of CMP and the monthly DAC Newsletter. During the last five years Mr. Moretti has written over forty full-length feature articles, and covered over 300 product introductions. He held the position of contributing editor at EDNonEDA, a bi-weekly electronic newsletter covering the EDA market, and also authored the yearly EDA forecast (2002-2004) for the Electronics Yearbook published by Reed Elsevier. Mr. Moretti holds a Bachelor of Business Degree from Whittier College and a Master of Sciences degree in Computer Sciences from the University of California, Los Angeles.

Best Practices for Adopting Model-Based Design in Electronic System Development

Introduction

Advances in semiconductor manufacturing process technology offer ever-larger numbers of transistors at ever decreasing unit price. These advances give electronic product companies the opportunity to deliver marketable, differentiated products if their engineers can be empowered to cope with the increase in complexity. Fortunately innovations in embedded software, electronic design automation (EDA), and electronic system-level (ESL) tools and methods are beginning to allow this.

One such innovation is Model-Based Design. In the many companies that have adopted Model-Based Design, engineers improve efficiency by:

- Using a common design environment across project teams
- Linking designs directly to requirements
- Integrating testing with design to continuously identify and correct errors
- Automatically generating embedded software code and synthesizable HDL code
- Developing and reusing test suites
- Automatically generating documentation
- Reusing designs to deploy systems across multiple processors and hardware targets

On one level Model-Based Design is quite simple: first build an executable specification (called a model) of the product, then base your design on it. But, as with any paradigm shift, there is a learning curve in adopting it. For this reason, we interviewed a number of expert users in leading electronics companies of the premier Model-Based Design tool, Simulink. Using a previous study¹ as a framework, we distilled their experience into this report, identifying ten "best practices" they used in their initial use of the tool. We feel this check list will be very valuable to those adopting Model-Based Design for the first time.

The rest of the report consists of a brief description of Model-Based Design, followed by a section on each of the ten best practices.

¹ "Best Practices for Establishing a Model Based Design Culture" by Paul F. Smith, Sameer M. Prabhu, Jonathan H. Friedman focuses on Model-Based Design for automotive drive train and aircraft flight control design.
<http://www.mathworks.com/mason/tag/proxy.html?dataid=9332&fileid=40538>

Model-Based Design

Model-Based Design aims to address the design and verification issues inherent in today's electronic systems designs by providing a formal structure that engineers can use to progress from specification to implementation. A formal design specification should be executable in order to validate the specification against requirements.

An executable specification allows the orderly progression of development within an environment that can insure continuity between steps such that all transitions from one step to another can be verified with computer based tools. The use of an executable specification provides an unambiguous description of both the functional and physical requirements for the project. Its existence significantly lowers the verification cost not only by providing clearer guidance to the development of testbenches, but also by helping designers to avoid errors due to misunderstandings of the specification. By using Model-Based Design developers assure themselves that they are building the right thing, avoiding misunderstanding that often occur when someone is trying to interpret the intentions of the author of a written specification.

Informally, validation is answering the question “Am I building the right product?” whereas verification is answering the question “Am I building the product right?”

Model-Based Design builds on existing tools and languages to implement a methodology based on the use of executable specifications. It guides engineers to design with simulation in order to continuously verify their development. Following this method, systems engineers build a model of the system that represents as much as possible of its functional and physical requirements. Then they decompose the model into models each

representing a functional block of the total system. Through a series of successive refinements, each block acquires additional definitions and constraints dictated by the technology available and the implementation capabilities unique to the company. Each refinement decreases the level of abstraction of the representation of the block's implementation.

During the design process, designers partition the initial block into smaller blocks until they reach blocks that can be implemented and verified as a unit. This process can be applied to digital hardware, analog hardware, and software components of the system. The approach is the traditional way to deal with complexity: divide and conquer. Using simulation, designers can compare the behavior of the new implementation with that of its immediate parent to insure that the two levels of abstraction are equivalent. The successive refinements process looks at the source block, identifies functional groupings and logical interfaces and divides the block into more manageable parts. Each of the parts is still a self-standing unit and as such it can be implemented and tested separately from the whole system. It is important that this process conserve an audit trail, so that if it becomes necessary to introduce a change in a lower block, designers can reflect the change backward in order to maintain equivalence among all the abstraction layers.

The key to success in this process is the precise and clear definition of interfaces, as well as the functional and timing interaction among the various blocks. Model-Based Design allows design teams to use all of the techniques described in the previous section to improve the quality and decrease the cost of the project. The result is a shorter development schedule that, in turn, maximizes the opportunity to hit the product market window and thus increase revenue.

Improved analog/digital and hardware/software implementation tradeoffs are also an offshoot of Model-Based Design. The successive refinements process allows designers to view, very early in the design cycle, the relationships among the various functional blocks. They can test within acceptable approximations the size and speed of execution of alternate implementations of each block and thus decide whether a specific function is better suited to software or hardware, digital or analog implementation. Having a complete functional view of the system divided into possible implementation blocks is also extremely useful in deciding on the implementation strategy with respect to design re-use and platform based design.

In the past the design on which the model was based was hand coded in languages such as assembly, C, Verilog and VHDL and verified against the test environment in the model by cosimulation. Recent improvements in cost-per-gate, cost-per-bit, compiler-friendly processor cores, and RTL synthesis tools have enabled an additional "arrow" in the implementation "quiver," namely automatic code generation. Such code is valuable for proof of concept, prototyping, executable specifications in the language preferred by downstream teams, as a baseline for formal verification of hand written code, and as production code. Obviously, the availability of this new technology doesn't mean you have to make an all-or-nothing choice between 100% hand coding and 100% automatically generated production code. Seek out the best of both worlds.

Ten Best Practices in Adopting Simulink and Model-Based Design

I: Identify and Prioritize the Problems You Are Trying To Solve

Before adoption starts, it is important to establish metrics that identify the weak points in your current process. Examples included inability to hit release dates, excessive software defects, and lack of availability of prototype hardware.

For example, Lockheed Martin builds large systems that often include both hardware and software components. Their specific needs included hardware/software partitioning and co-development. They found that using both MATLAB and Simulink allowed them to evaluate partitioning tradeoffs as well as allowing software developers an earlier start than in previous projects².

Once these weak points are listed, rank them and attack your greatest weaknesses first. The idea is to demonstrate a return on investment (ROI) early on, by creating a business case for investment in the change.

A companion white paper “Return on Investment in Simulink for Electronics System Design³” illustrates the ROI that can be achieved.

2: Complete a Pilot Project

With business case in hand, you can schedule resources for a pilot project. These resources must be explicitly allocated (hence the need for a business case). The pilot project can't be done “in our spare time.”

Select a pilot project to evaluate how Simulink could be used within your organization, and to justify your choice within the organization. Choose a pilot project that is representative of a mission critical project, so that the results can be extrapolated to the rest of the company's product line.

Ideally you can evaluate and learn Simulink in parallel to using the previous method and tools. For example, BAE Systems chose to re-implement a software defined radio project they had completed previously in order to measure the benefits of using Simulink⁴. Because the team had already designed the product previously the evaluation project did not require any new inventions. This gave each member of the team the time to learn the tool and explore alternatives. Their experience showed that using Simulink they reduced the project schedule by 80%. This was due mostly to two major improvements attributed directly to using Simulink. Using iterative design and verification they identified and eliminated design and implementation errors much earlier in the process, which significantly shortened or eliminated debug and bug fixing activities. They also found that both clocking and timing could be significantly

2 http://www.mathworks.com/company/user_stories/userstory12094.html

3 <http://www.mathworks.com/roi>

4 http://www.mathworks.com/company/user_stories/userstory12386.html

simplified as a result of using Simulink to model system behavior and perform design tradeoffs in simulation.

Remember that the first objective is to learn how to use the new tool in the most effective way, so allow time for exploration and evaluation of the alternative designs you will generate. Your goal is to find the features that will solve the problem at hand and to learn the best way to use them in your development environment.

One of the key issues in adopting a tool is to assess how it will accommodate legacy IP both during the pilot and beyond. Simulink is integrated with MATLAB, which provides immediate access to algorithms that a company may have developed in the MATLAB language, as well an extensive range of tools for further algorithm development, data analysis and access, and numerical computation. Simulink also supports integration of design components written in other languages such as C, C++, or HDLs. Thus, adopting Simulink does not invalidate the inventory of IP blocks or block design methods available to the design team, whether such blocks were developed internally or obtained from a third party. ST Microelectronics found that because of the flexibility of Simulink, the model could be refined from a functional model in the very early stages to a more detailed one later in the design stages. Their advice to a new user is to take enough time to learn how to use the tool properly before starting a complex project.

The end of the pilot should be a milestone: schedule a post-pilot project review to determine whether the challenges are being addressed and the return on investment is as expected.

3: Use the Models for at Least Two Things—"Rule of Two" —But Limit the Number of Design Challenges Initially

All of your previous designs have presented challenges, or you would not be looking to adopt a new tool. But solving all of them with the first use of the tool is impractical and will in fact complicate the measurement of the productivity improvement of the team.

“MATLAB is an ideal environment for developing and understanding our algorithms. Simulink integrates well with MATLAB and lets us produce a design that looks very similar to what we end up with ultimately in hardware.”⁵

**—Francis Swarts,
Broadcom**

However, don't take this limitation to the extreme of using the models for only one purpose. The ROI increases significantly with multi-use models. You may, for example, use the models for both a "golden double precision reference" and refine it to explore implementation loss due to fixed-point effects. Or to create an executable specification and to verify the implementation via cosimulation and hardware in the loop. Specification and automatic code generation is another powerful combination.

You should pick a small number of challenges, certainly no more than a handful. Remember that each challenge must be measurable so that you can document, not just feel, the contributions that Simulink brings to the productivity of the team.

4: Treat the Models as "The One Truth"

It is important to prevent the divergence of design and model. If you don't treat the models as "The One Truth," then the models will not be maintained, and will be useless. Many projects fail because the de facto specification is not the actual original specification, but what the design in fact evolved to deliver. Think of that famous cartoon of the child who really just wanted an old tire tied to a tree branch, but who received a Rube Goldberg contraption that involved a tree propped up with scaffolding, a fancy but useless triple-decker swing – and a tree with part of its trunk missing!

Preventing divergence requires discipline in both top-down and bottom-up design components. For top-down refinement, use automatic code generation of your embedded C code and Verilog HDL or VHDL ("HDL") code. If the automatically generated code is "tweaked" use formal verification methods to verify against the functionality of the automatically generated code. Avoid having three partial truths: a bit of MATLAB here, C there, and RTL elsewhere.

Use bottom-up verification methods such as checker-boarding, cosimulation, and verification model extraction for i) stable modules ("legacy code") already implemented, ii) implementation code that must be hand optimized, and iii) for RF and analog circuitry (where limit is physics, not complexity, and where, therefore, automatic code generation has limited applicability).

⁵ http://www.mathworks.com/company/user_stories/userstory6356.html

5: Use Migration to Model-Based Design as a Learning Opportunity

A surprising side benefit that "first generation" Model-Based Design users discovered is that it helped them learn what really happens in their existing process. These valuable insights showed how to improve. "Second generation" adopters would do well to deliberately target the capture of this type of knowledge. In order to capture this benefit, it is important not to outsource the translation and transition; otherwise this knowledge will be lost to you. Of course it is entirely appropriate to solicit help on process and tools (see best practice #10 below), but be sure to do the translation yourself: it is a tremendous learning and quality improvement opportunity, even in organization that were efficient before adoption.

6: Focus on Design Not Coding

It is a truism that creative work adds value, while mechanical work merely adds cost. An example is conversion of a floating point algorithm to fixed-point. Model-Based Design tools free up time from mechanical work, and enables engineers to migrate to more creative (and fun!) tasks. Texas Instruments for example, has developed metrics for design time that are separate from metrics for the implementation time⁷. In this way design is judged on its own merits making evaluation of tools more straightforward.

Based on their pilot experience, Agilent pointed out that providing linkage to tools that handle the other parts of the job, like implementation of high performance, concurrent multirate algorithms on FPGA hardware, was a key factor in their successful adoption of Model-Based Design and Simulink. The difficulty of integrating design tools with the existing RTL flow is one of the problems that have slowed down the adoption of ESL methods. Agilent has focused on keeping as much as possible of the downstream methods while increasing its design efficiency. Doing so has helped them to clearly identify the requirements of the integration process.

“Using MathWorks tools for Model-Based Design, we improved productivity and completed the design in about half of the time that I would have expected.”⁶

**—Idan Bar-Sade,
BridgeWave
Communications**

⁶ http://www.mathworks.com/company/user_stories/userstory11062.html

⁷ http://www.mathworks.com/company/user_stories/userstory2356.html

7: Integrate Tools into Development Process

It is important to develop comprehensive plan to integrate tools into the development process. A good rule of thumb is the 25/25/50 rule: 25% of the investment is in tools, 25% in training, and 50% in integration of tools and process. Consider processes and supporting tools that standardize and enforce modeling style, configuration management, and requirements management.

That which is measured is that which improves: choose your metrics carefully.

You'll need to develop new metrics to track effectiveness of these new processes because the old metrics such as "lines of code per engineering-month" or "defects per line of code" will no longer be relevant. "That which is measured is that which improves:" chose good metrics. A companion article "Measuring Productivity and Quality in Model-Based Design"⁸ is available that covers this part of this best practice in depth.

8: Enroll a Champion That Has Influence and Budgetary Control

The champion must have the influence and budgetary control to assign priority, assign people, and acquire tools, equipment, and services. These tasks are particularly challenging in the early (pre-pilot) days when the benefits are promising, but the costs are real.

The champion must be a consensus builder, because it is unlikely they will have direct reports in every department that is impacted by the adoption.

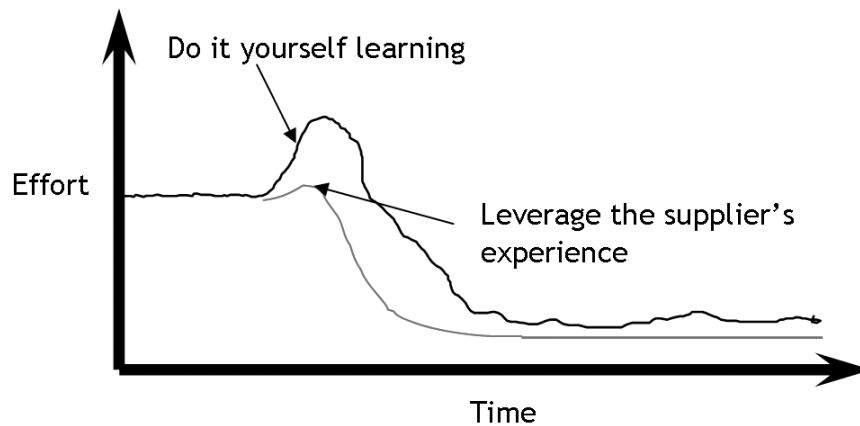
9: Have a Long-term Vision and a Plan to Move Toward It

Having a long term view is consistent with Best Practice #1 (Demonstrate Return on Investment (ROI) early on) because, although some parts of adoption will offer the "low hanging fruit," it is also true that whole is greater than the sum of the parts, and that the full transition from paper-based specifications (or hand coded, textual languages) takes 2-3 years to fully implement in a production organization. In contrast, research organizations often have fewer constraints and less legacy code, and can move faster.

⁸ <http://www.mathworks.com/company/newsletters/digest/2006/mar/measuringprod.html>

I0: Partner with Your Tools Suppliers

By partnering, you'll leverage your suppliers' cross-industry experience, avoid common pitfalls, and accelerate your breakeven point. By overcoming the learning curve you'll quickly achieve productivity and quality goals.



Conclusion

The benefits from using Model-Based Design and Simulink will vary depending on the goals each company sets. But the benefits obtained by Texas Instruments are typical. They have shortened the design time, which they measure separately from the implementation time, by a couple of months, have realized improved profits from a shorter time to market for each product, and have gained from improved product quality as measured in yield from the factory. Lockheed Martin builds large systems that often include both hardware and software components. They have found that their development costs have decreased by using Model-Based Design and Simulink.

It is clear that it is possible to express the benefits of using Model-Based Design and Simulink in financial terms. But the key is to design and manage a pilot project that highlights the contributions of the proposed methodology and also indicates opportunities for improved quality and productivity.

The benefits of Model-Based Design will not be limited to those proved during the pilot project. As the entire organization internalizes the new methods, other opportunities for improvements will be found and justified.

