Policy-Based RTL Design

Bhanu Kapoor and Bernard Murphy
bkapoor@atrenta.com
Atrenta, Inc., 2001 Gateway Pl. 440W San Jose, CA 95110

Abstract

The design decisions, made early in the design process, effect the entire chip design process. To manage this effectively, the RTL design process must be reliable in a way that the tools carrying out stepwise refinement of the RTL code towards the final layout become predictable in achieving the desired goals. We present a new methodology to accelerate the design of complex ASICs and SoCs through predictive analysis and policy-based RTL code development. The goals of policy-based RTL design, as outlined in this paper, include creation of a system that addresses RTL design using policies and guides the design process efficiently towards design goals under given design constraints.

1 Introduction

Silicon technology now allows us to build chips consisting of tens of millions of transistors. While this technology promises new levels of system integration onto a single chip, it also presents significant challenges in terms of figuring out a design methodology for effective use of available transistors on a single chip. There is a common set of problems facing everyone involved in large-scale designs:

- Time-to-market pressures demand rapid product development cycle
- Quality of results, in performance, area, testability, and power, are key to market success
- Increasing chip complexity makes design verification complexity grow at a much faster rate
- Deep sub-micron issues make timing closure more difficult
- Development team with different levels of expertise, multiple design teams, often extended geographies with different design cultures
- Lack of the ability to reuse work from previous similar design projects
- Being able to plan and execute a predictable project schedule

Each of the point-solution CAD tools has been effective in solving a particular area of problem. For example, recent productivity gains in a designer's ability to generate gates have stemmed form the advances and widespread acceptance of synthesis technology into today's design flows. In fact, design productivity has risen tenfold since the 1980s in terms of number of gates designed per day but such a claim cannot be made towards the ability to verify these designs. The goal of taking an RTL description of a large design and effectively using it for various goals towards the completion of the design has remained a challenge. As a result, we follow the step-by-step refinement process and encounter problems in later stages of the design cycle that require large resources and more than offset any gains made by the individual tools.

The net effect of these issues shows up in inefficient project planning and schedule development. Some of the large commitments for the project are made early in the design cycle when the design knowledge is at a minimal level as shown in Figure 1. This eventually shows up in schedule slips and budget overruns.

The design decisions, made early in the design process, effect the entire chip design process. To manage this effectively, the RTL design process must be reliable in a way that the tools carrying out stepwise refinement of the RTL code towards the final layout become predictable in achieving the desired goals.

The ability to address potential timing, power, testability, and size issues early in the design cycle is critical to achieving high productivity in the design process. The ability to address these issues during the RTL code development not only results in more optimised designs but also helps improve efficiency of other tools being used in the design process. In addition, there is a lot to be gained from having a truly golden RTL code for the design not only in the current design but also in the future generation designs and systems.

We present a new methodology to accelerate the design of complex ASICs and SoCs through predictive analysis and policy-based RTL code development. This approach performs detailed structural analysis on RTL in order to check coding styles, RTL-handoff, design re-use, clock/reset requirements, verification, timing, design for testability, low-power guidelines and much more. It requires a "look-ahead" engine that is based on fast-synthesis technology and a fast in-built cycle-based simulator to carry out such analysis during the RTL code development process.

The goals of policy-based RTL design, outlined in this paper, include creation of a system that addresses RTL design using policies which guide the design process efficiently towards design goals under given design constraints. Specifically, the system performs many functions including:

- Policy Application
- Policy Creation
- Policy analysis and report

The rest of the paper is organized as follows: Section 2 describes the basic model for policy-based RTL design, elements of policy enforcing engine, and the policy management process. Several examples of policies are described in section 3. We then conclude the paper with a summary and a list of references.

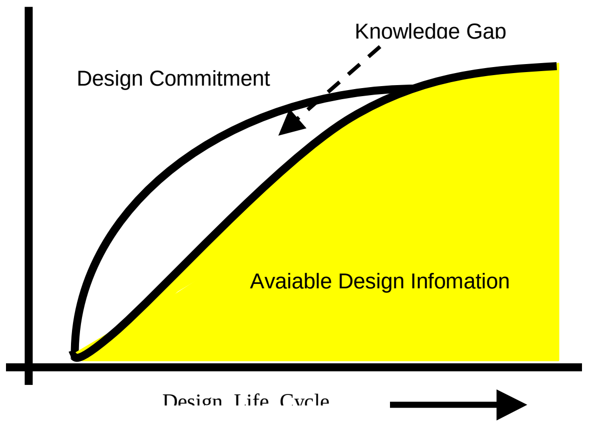


Figure 1: Challenges in Product Development

2 Model for Policy Management

What is needed is a comprehensive, policy-based system that will allow designer to define, in a succinct and organized fashion, design policies that automatically point out time consuming downstream issues during the RTL code development process. The end result is that time-to-market goals are met and predictable schedules become a reality.

Basic elements constituting policy-based RTL design are as follows:

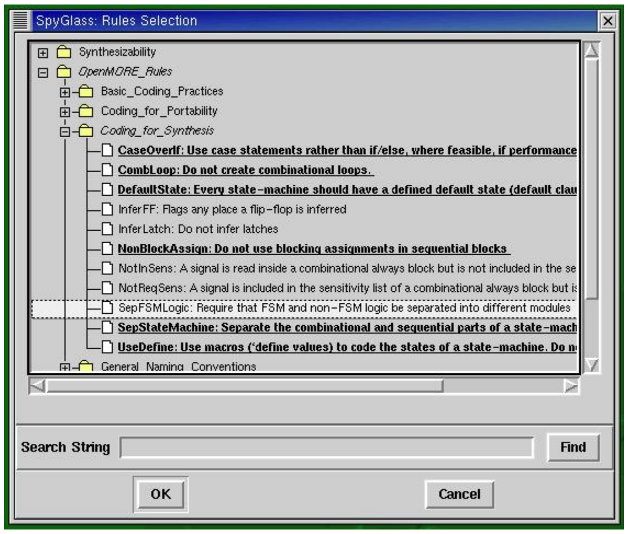
2.1 Policy

A policy is a collection of rules for specific purpose such as rules associated with certain standard or certain design tool. Policies extend user extensibility, allowing you to develop and manage you customized groupings of rules more easily. The design methodology includes a set of policies that can be selected by the designer during the process of RTL code development. Examples of such policies are as follows:

- Lint
- Reuse
- Verification
- Area
- Timing
- Testability
- Power
- Clocks and Resets
- Vendor tools

2.2 Rule Groups

A collection of rule is termed as a group. Typically, groups consist of rules addressing a particular area of interest in the RTL code. Groups are hierarchical: that is, a group can contain other lower-level rule groups as well as individual rules. A group provides additional level of modularity in applying policies to a given RTL design.



```
⊞ Synthesizability
⊟ OpenMORE_Rules
   ⊞ Basic_Coding_Practices
   ⊞ Coding_for_Portability
   ⊟ Coding_for_Synthesis
      CaseOverIf: Use case statements rather than if/else, where feasible, if performance
      CombLoop: Do not create combinational loops.
      DefaultState: Every state-machine should have a defined default state (default clau
      InferFF: Flags any place a flip-flop is inferred
      InferLatch: Do not infer latches
      NonBlockAssign: Do not use blocking assignments in sequential blocks
      NotInSens: A signal is read inside a combinational always block but is not included in the se
      NotReqSens: A signal is included in the sensitivity list of a combinational always block but is
      SepFSMLogic: Require that FSM and non-FSM logic be separated into different modules
      SepStateMachine: Separate the combinational and sequential parts of a state-mach
      UseDefine: Use macros ('define values) to code the states of a state-machine. Do n
   ⊞ General_Naming_Conventions
```

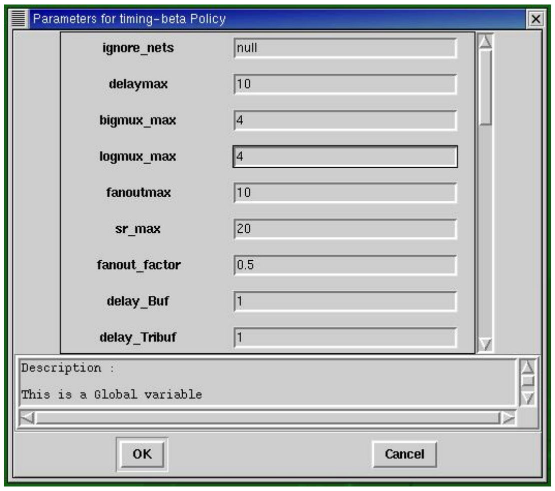
Figure 2: Rule and Group Selection

2.3 Rules

A rule is the most fundamental element in the policy-based management system. It describes a set of conditions when triggered by the policy engine result in an indication of a specific issue with the RTL code. These rules allow standard analysis of the RTL code. A simple simple of a rule could be: *There should be no stranded states in the FSMs described in the RTL code.* Figure 2 shows an example of group and rule selection during the policy-based RTL design process.

The result of applying policies to an RTL design is s set of rule violations with different severity levels, cross-linking of these violations with RTL code and schematic, and a process of suggested corrections and/or automated correction of the issue in the RTL code. Each policy can be tailored for application with a list of parameters. Figure 3 shows an example for the timing policy that includes parameters such as levels of logic, fanout, and the allowed size for multiplexers.

Policy management involves the selection of rules, groups, and policies, application of policy parameters, creation of new rules and policies, and policy analysis and report generation.



ignore_nets	null
delaymax	10
bigmux_max	4
logmux_max	4
fanoutmax	10
sr_max	20
fanout_factor	0.5
delay_Buf	1
delay_Tribuf	1

Description :
This is a Global variable

Figure 3: Parameters defining a timing policy

2.4 *Policy Engine*

Policy enforcement is enabled through an engine consisting of fast synthesis, fast cycle-based simulation, and a fast design database traversal working at a higher-level of abstraction.

Policy engine accelerates electronic product development by enabling development teams to capture, aggregate, distribute and apply constraints and requirements early in the development cycle. The fast synthesis engine internally creates the design structure and foresees downstream issues early in the development cycle, thereby eliminating errors at the earliest possible stage. Such an engine ensures that designs are always compliant with value chain constraints by identifying, advising and correcting violations as the design progresses towards completion. The engine preserves the correlation of RTL to the fast-synthesis netlist so that any errors detected there can be traced back to the origin for quick problem identification and correction.

The evaluator in the Policy engine is effectively a zero-delay cycle-based simulator used to resolve functional design constraints as well as carry-out simulation required to set up the design for testability analysis.

Policy creation requires a traversal engine that works on the RTL netlist resulting from fast synthesis and provides user with basic primitives to implement rules requiring traversal of the design. The connectivity information coupled with traversal primitive allow for creation of rules that look for violations across the design hierarchy.

3 Policy Examples

The design decisions, made early in the design process, effect the entire chip design process. To manage this effectively, the RTL design process must be reliable in a way that the tools carrying out stepwise refinement of the RTL code towards the final layout become predictable in achieving the desired goals. A detailed structural analysis on RTL in order to check coding styles, RTL-handoff, design re-use, clock/reset requirements, verification, timing, design for testability, and low-power guidelines can be applied during the RTL design process to highlight issues which will remain after detailed synthesis and refinement are carried out by other tools. Figure 4 shows a set of policies that can be applied during RTL development.

3.1 *Lint and Reuse*

Many of the techniques for design reuse are just good design techniques such as good documentation, structured code, thorough commenting, and properly organized code. These are captured in approximately 200 rules in the lint policy. Lint provides a powerful method for checking the RTL for violations of coding guidelines and other kinds of coding errors not reported by the RTL compilers. This policy is typically used throughout the RTL design process since it is the fastest means of catching the most basic issues with the RTL code. Some of typical lint issues encountered during the course of RTL code development include issues such as:

- Not all cases are covered in a case statement.
- Variable specified in the in the sensitivity list but not contained in the block of code.
- Multiply driven net during the same clock cycle

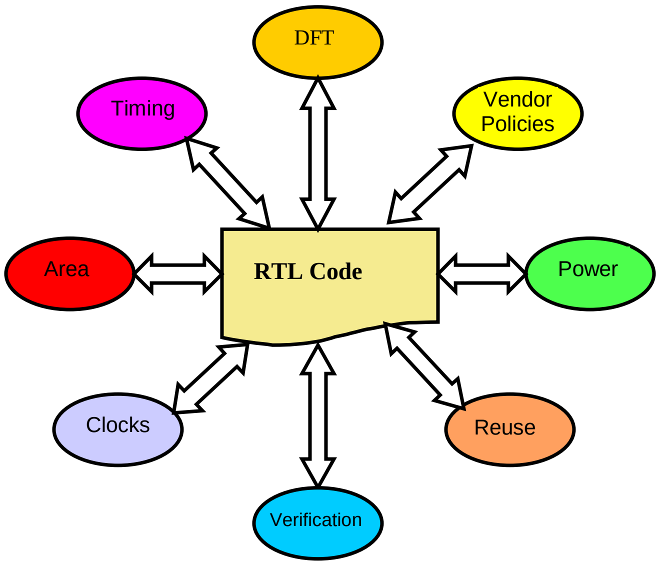


Figure 4: Example of policies in Policy-based RTL Design

The design for reuse [1] philosophy implies RTL code development that is designed to solve a general problem, for use in multiple technologies, targeted for efficient use of simulators and synthesis tools, and follows recommended design practices. The OpenMORE [5] policy with approximately 150 rules and guidelines that address issues associated with the reuse of soft IP cores and 90 rules and guidelines of hard IP. Another policy for reuse is defined by the Japanese Semiconductor Technology Academic Research Center (STARC) [4] initiative. All design projects have the basic underlying goal to develop RTL code that is simple and regular. Simple and regular RTL structures are easier to design, support, verify, synthesize, and reuse as compared to the code developed in the projects that do not follow design guidelines. A typical design project should provide the engineers with a set of guidelines that are likely to streamline synthesis and other implementation issues down the road. Some typical guidelines used in projects include practices such as:

- Use "case" statements instead of "if else" for performance
- Combinational loops are not allowed
- No stranded states in the finite state machines
- Blocking assignments in the sequential blocks must be avoided
- Separate the combinational and sequential parts in the FSM
- Use define macro to code the states of an FSM; no hard coded states

3.2 Verification

Enabling RTL code that follows a verifiable style serves both for the success of EDA tools as well as enhances the productivity of less experienced engineers who participate in the design and verification process. The verification policy address verification from several angles including coding for verifiable RTL, coding for functional and code coverage, coding for verification reuse, coding for simulation, and coding for formal verification success. Some of issues such a policy can help with are:

- Synchronous and Asynchronous race conditions at time zero during simulation
- Avoid blocking assignments in sequential code inferring flops
- Avoid multiple drivers to clock, data, and reset inputs of sequential devices

3.3 Area and Timing

An early estimate during the RTL code development on the size and performance issues of the design can potentially save significant design time in the later stages of the design cycle. Using the area policy, one can get the partition size information and the timing policy looks into issues such as large fanout nodes and the number of levels of logic in the design to give designers an early estimate for potential issues down the road.

Once the RTL code is clean with respect to lint issues, initial simulation has been carried out, and project guidelines have been met, an estimate of total area of the design as well as the distribution of area for different units in the design can be a valuable data early in the design process. For designs in the networking arena, some estimate of amounts of memory being used in different sub-systems can be equally valuable in design planning.

The timing policy looks into issues such as large fanout nodes and the number of levels of logic in the design to give designers an early estimate for potential issues down the road. Large fanout nodes may indicate the need for duplication of registers to achieve higher performance. The levels of logic can point out areas where retiming across registers can help improve performance of the design. The policy consists of parameters to allow users to customize feedback utilizing their experience from the past designs.

3.4 Clocks

It is important to identify the clocks and resets in the design as well as the clock-tree network early in the code development process. A policy devoted to dealing with many challenges presented by the clocks and resets in the design. It is important to identify the clocks and resets in the design as well as the clock-tree network early in the code development process. Many designs, especially the ones in the networking domain, typically use multiple clocks domains that present us with challenging issues in integration and verification of such systems. Signals originate in one clock domain, but they may be used in other clock domains. Correct operation of a chip can only be ensured if rules governing correct use of signals across asynchronous clock boundaries are followed. The clocks policy identifies signal paths that cross clock domains, analyzes whether paths conform to selected synchronization rules, and ensures whether gated clocks are derived according to specified rules. It recognizes clocks by looking for flops, then finding the signal attached to the clock input. For Verilog, this means finding a signal used as an edge-trigger to an "always" block and not used as an asynchronous reset within that block.

3.5 Power

With the ongoing fusion of rapidly emerging technologies such as personal communications, portable computing, and high bandwidth communications, the need for low power digital and analog designs [3] is ever increasing. The opportunities for achieving low power design exist at various levels during the product development process. These levels can be identified as: 1) technology 2) layout 3) transistor-level circuit 4) gate-level circuit 5) architecture and system, and 6) algorithms. A power policy addresses the gate and architecture level issues using the RTL code. Some of issues being addressed focus on adhering to low power design guidelines such as:

- All banks of memory controlled by enabled logic cone
- All buses driven by tri-state drivers
- Check for one-hot coded state machines
- Use of gated clocks to selectively turn on/off various units

3.6 Testability

To get the design ready for testability, the testability policy has rules which address topology related issues as well as functionally dependent issues. The topology related rules depend only on part type, pins, and interconnections and functionally dependent rules involve test mode, test clock pins, parameters and circuit reset checks. This category includes rules such as combinational feedback detection and port-to-port path connection.

- Nodes in the design should be observable
- No gated clocks
- Report multiple clock domains
- No switching on both edges of the clock
- Potential race conditions

In addition, policies that make other CAD tools more efficient are equally important for the goals of policy-based designs. Some of our policies can significantly enhance user productivity when using products such as simulators and timing closure tools. A policy can be equally applicable to lower-level netlists and can include common rules checked prior to the physical design hand-off.

4 Summary

We have described a design methodology based on policy-based RTL design. The elements of policy-based RTL design have been outlined. This approach performs detailed structural analysis on RTL in order to check coding styles, RTL-handoff, design re-use, clock/reset requirements, verification, timing, design for testability, low-power guidelines and much more. The end result is that time-to-market goals are met and predictable schedules become a reality.

5 References

[1] M. Keating and P. Bricaud, "Reuse Methodology Manual: For System-on-a-Chip Designs", Kluwer Academic Publishers, 2001.

[2] L. Bening and H. Foster, "Principles of Verifiable RTL: A functional coding style supporting verification processes in Verilog", Kluwer Academic Publishers, 2001.

[3] A. P. Chandrakasan, S. Sheng, and R. W. Brodersen, "Low Power CMOS Digial Design", IEEE Journal of Solid State Circuits, pp. 473-484, April 1992.

[4] Semiconductor Technology Academic Research Center, "STARC open Design Style Guide", www.starc.or.jp/.

[5] OpenMORE, "The Industry Reference for IP Measure of Reuse Excellence" www.openmore.com/