

Implementing Rule Checking Early in the Design Cycle to Reduce Design Iterations and Verification Time

Dominic Lucido
Field Applications Engineer
Mentor Graphics





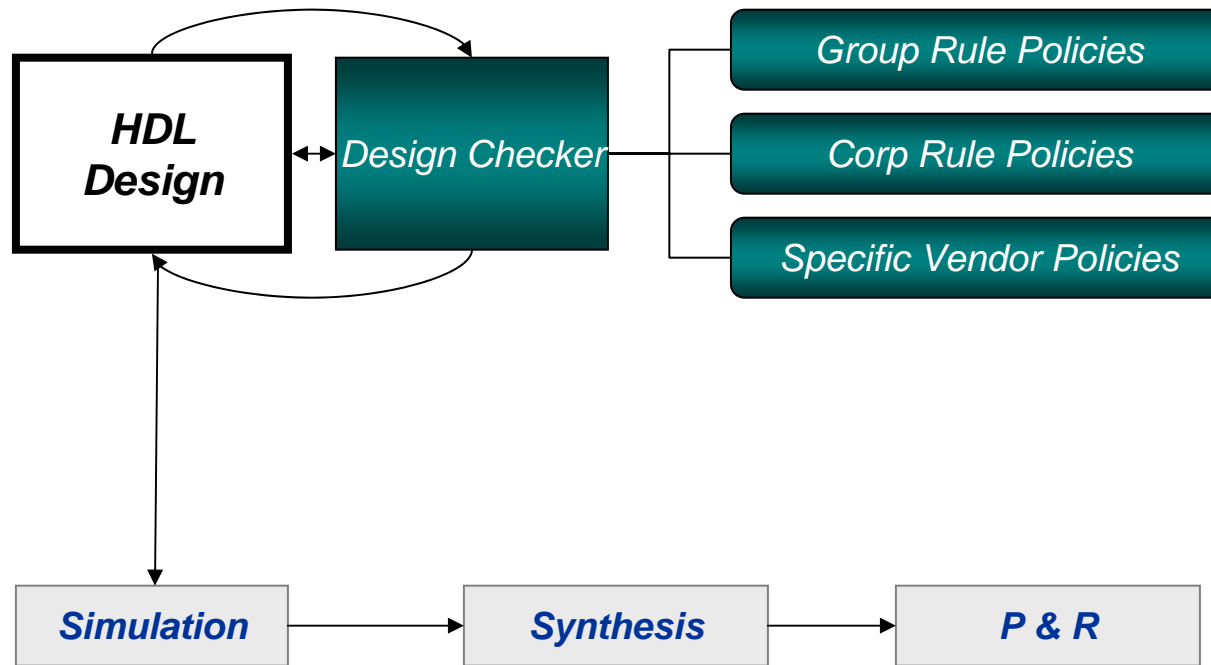
Design Checking:

Evaluating RTL code to determine if any violations exist based on a set of coding rules

What Problems Do Design Checkers Solve?

- ✓ **Verification consumes 50-70% of FPGA design cycles**
 - Design checking finds and fixes errors earlier in the design process to reduce costly verification and other downstream tool iterations later
- ✓ **Design reuse is essential to meet FPGA design schedule constraints**
 - Design checking helps ensure adherence to HDL coding guidelines essential for efficient reuse

Early Checking Design Flow



Design Checking Tenets



Encapsulate Knowledge

- Best Practices
- Reuse Techniques
- Known Issues



Apply Knowledge

- Understand Design Issues
- Detect & Fix Issues

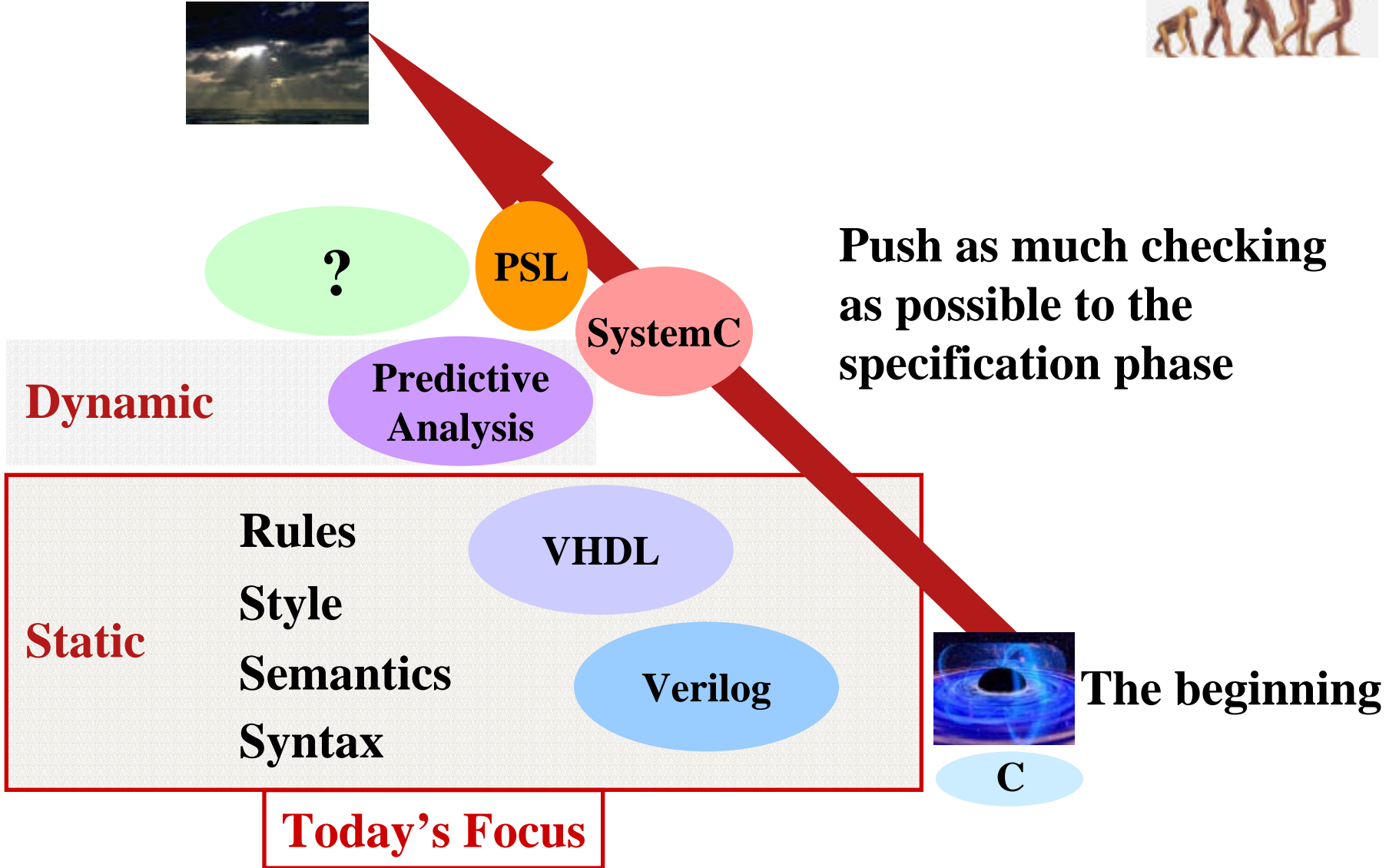


Share Knowledge

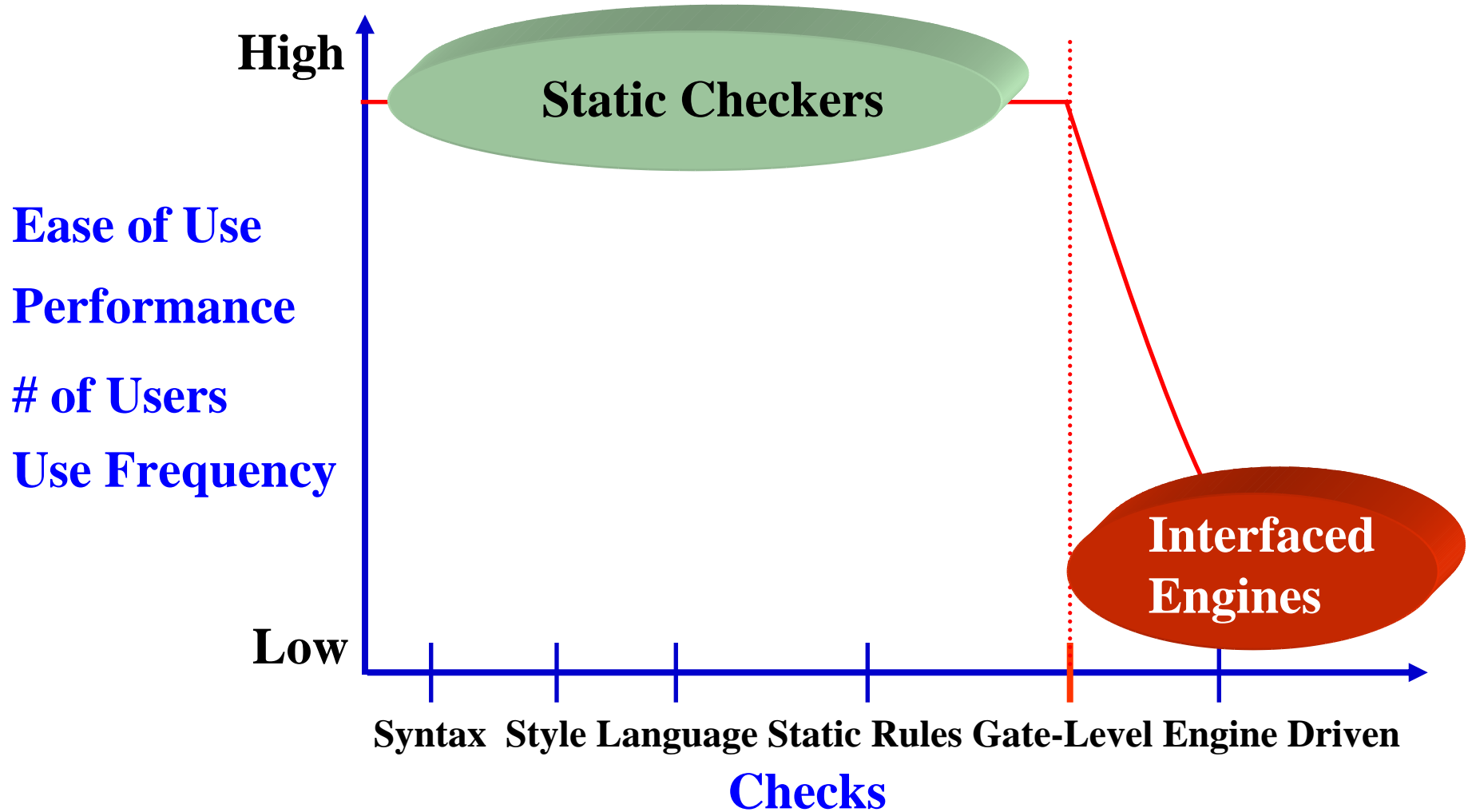
- Team Access
- Issue Reports

**HDL
Checking
Tool**

Checker Evolution



Why Focus on Static Checks?



How Fast is Fast Enough?



Design	Lines	Sec	Rules	Checks/Sec
Ethernet	9,823	9	160	174,631
Leon uP	14,860	10	160	237,769
PicoJava	60,751	80	160	121,502
MicroSparc	139,080	290	160	76,734

- Tests run on a PC: 2 GHz Pentium 4, RAM 1 G, Windows 2000
- Used same set of rules in each test including
 - HDL syntax and semantics
 - Reuse Methodology Manual ruleset
- $\text{Checks/Sec} = (\text{Lines} * \text{Rules}) / \text{Sec}$ (Lines include comments)

What Types of Checks can be Performed?

- ✓ **Standard language & syntax checks**
- ✓ **Good coding practices**
- ✓ **Format & readability**
- ✓ **Downstream tool checks**
- ✓ **Portability/reuse checks**
- ✓ **Cross-language compatibility**



Example Rule Categories

- √ **Allow**
- √ **Assignments**
- √ **Clocks & Resets**
- √ **Complexity**
- √ **Conditions**
- √ **Configurations**
- √ **Declarations**
- √ **Directives**
- √ **FSM**
- √ **Gates**
- √ **HDL Syntax & Semantics**
- √ **Instances**



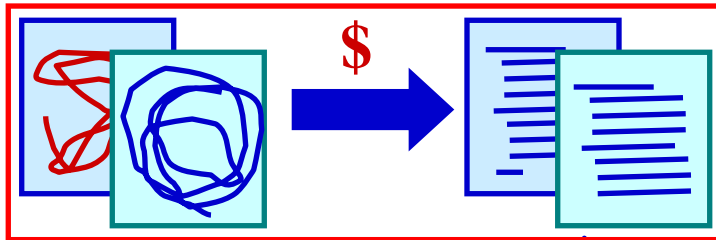
- √ **Labels**
- √ **Naming**
- √ **Order**
- √ **Partitioning**
- √ **Race Conditions**
- √ **Ranges**
- √ **Registers**
- √ **Sensitivity**
- √ **Style**
- √ **Subprograms**
- √ **VITAL**

Reuse Methodology Manual Examples

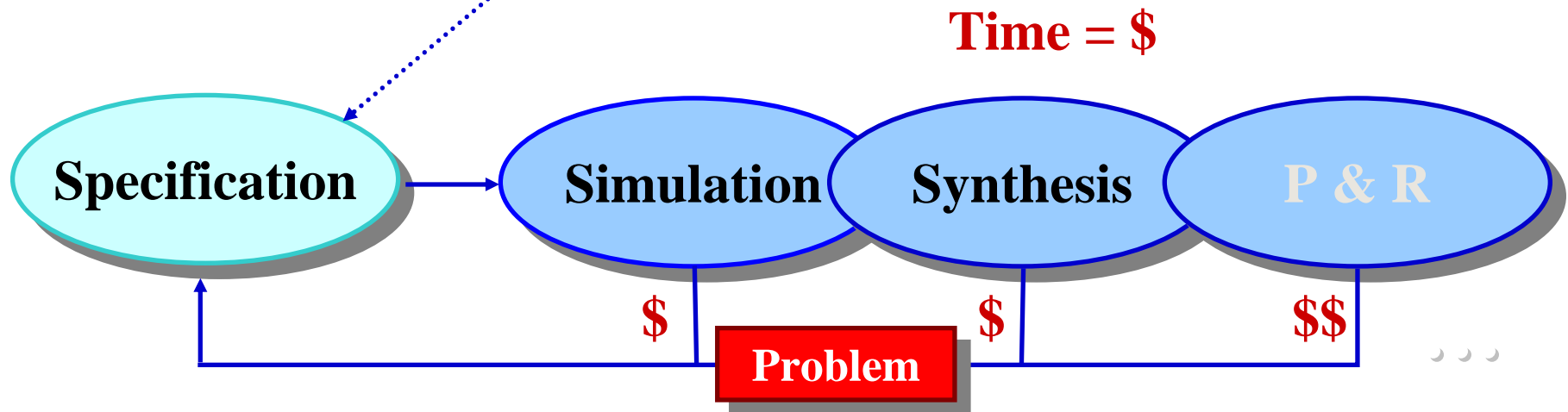
Clocks & Resets	5.4.8.1	Single-Bit Synchronizers	Use 2 FF stages to transfer single bits between clock domains. Label the FFs with distinctive names. Do not connect combinatorial logic between clock domains.
Clocks & Resets	5.4.9.1	Multiple-Bit Synchronizers	Do not use multiple single-bit synchronizers to transfer multiple bit fields between clock domains. Use handshake circuit or multibit coding schemes (eg Gray code).
Synthesis Coding	5.5.1.1	Infer Registers	Use technology-independent RTL style to infer registers (FFs) for sequential logic. Use the master reset to initialize the registered signals. Do not initialize in the declaration (VHDL) or within an INITIAL block (Verilog).
Synthesis Coding	5.5.2.1	Avoid Latches	Avoid inferring any latches unless as instances of technology-independent D-latch components (eg for register files, memories, FIFOs etc).
Synthesis Coding	5.5.2.2	Avoid Latches	To avoid latch inference: 1) VHDL-assign default values at start of process, 2) Verilog-assign outputs for all input conditions, 3) VHDL-ensure final branch is "else" not "elsif".

Value of Early Detection

Reuse Cost

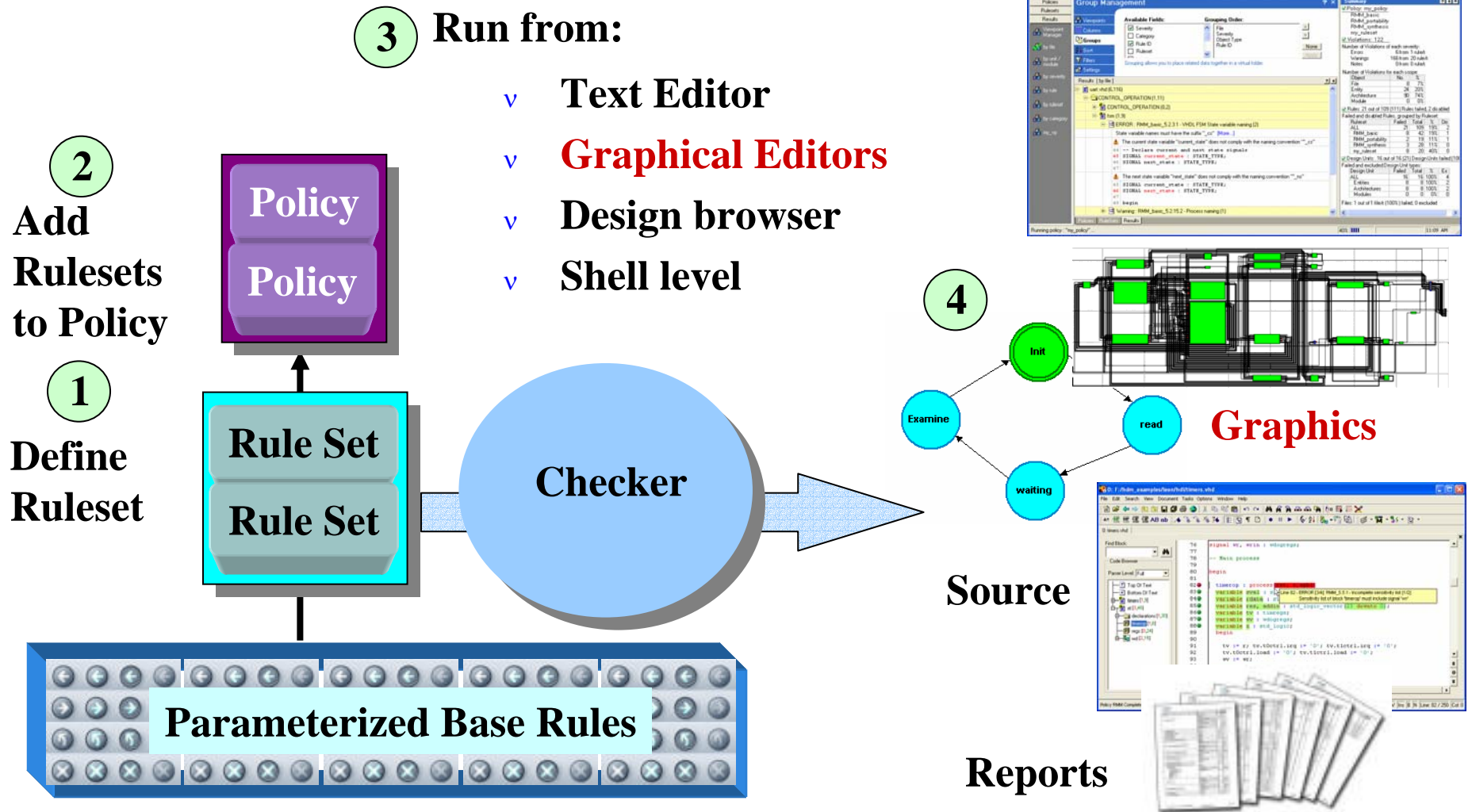


- ✓ Cost of error detection increases further downstream
- ✓ Cost is multiplied if code must be edited and re-verified for subsequent designs



$$\text{Cost} = [(T_{\text{fix}} + T_{\text{reiterate}}) * \text{Cost}_{\text{Engineer}}] + T_{\text{market}}$$

Understanding the Process



Building Custom Rulesets

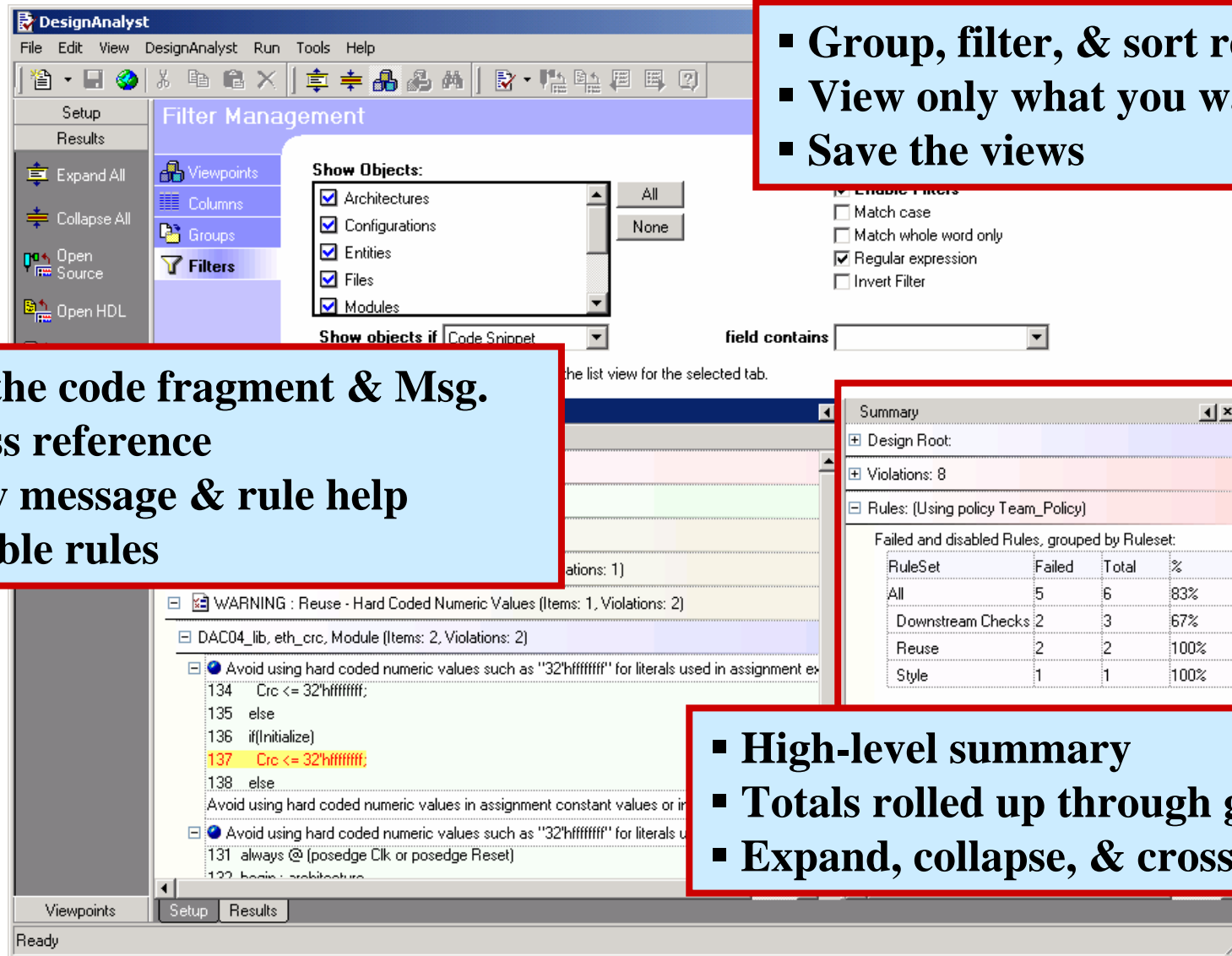
The screenshot shows the DesignAnalyst software interface. The left pane displays a tree view of folders including Policies, RuleSets, and Base Rules. The middle pane shows the 'Content of RuleSet : Clocks & Resets' with a list of rules and their categories. The bottom pane shows the 'Parameters of Configured Rule : Clock Domains' with a table of parameters and values.

Name	Base Rule Category	Base Rule Name
<input checked="" type="checkbox"/> Clock Boundaries	Clocks & Resets	Clock Boundaries
<input checked="" type="checkbox"/> Clock Domain Names	Clocks & Resets	Clock Domain Names
<input checked="" type="checkbox"/> Clock Domains	Clocks & Resets	Clock Domains
<input checked="" type="checkbox"/> Conditional Resets	Clocks & Resets	Conditional Resets
<input checked="" type="checkbox"/> Edge Detection	Clocks & Resets	Edge Detection
<input checked="" type="checkbox"/> Flip Flop Clocks	Clocks & Resets	Flip Flop Clocks
<input checked="" type="checkbox"/> Internally Generated Clocks	Clocks & Resets	Internally Generated Clocks
<input checked="" type="checkbox"/> Internally Generated Resets	Clocks & Resets	Internally Generated Resets
<input checked="" type="checkbox"/> Mixed Clocks Resets	Clocks & Resets	Mixed Clocks Resets
<input checked="" type="checkbox"/> Reset Logic Function	Clocks & Resets	Reset Logic Function

Parameter	Value
Name	Clock Domains
Severity	Error
Language	VHDL Any, Verilog Any
Hint	Minimize the number of clock domains within the design.
Short Description	Checks the number of clock domains in the design. [Design-wide]
Allowable Number Of Domains	1
Hierarchy Level	All Levels

- ✓ Create ruleset(s)
- ✓ Use search to find a base rule
- ✓ Access online help
- ✓ Drag & drop base rulesets & rules into your own rulesets
- ✓ Change rule parameters
- ✓ Create policies that contain rulesets
- ✓ Disable rules
- ✓ Lock down & share rules with the team

Examining Results



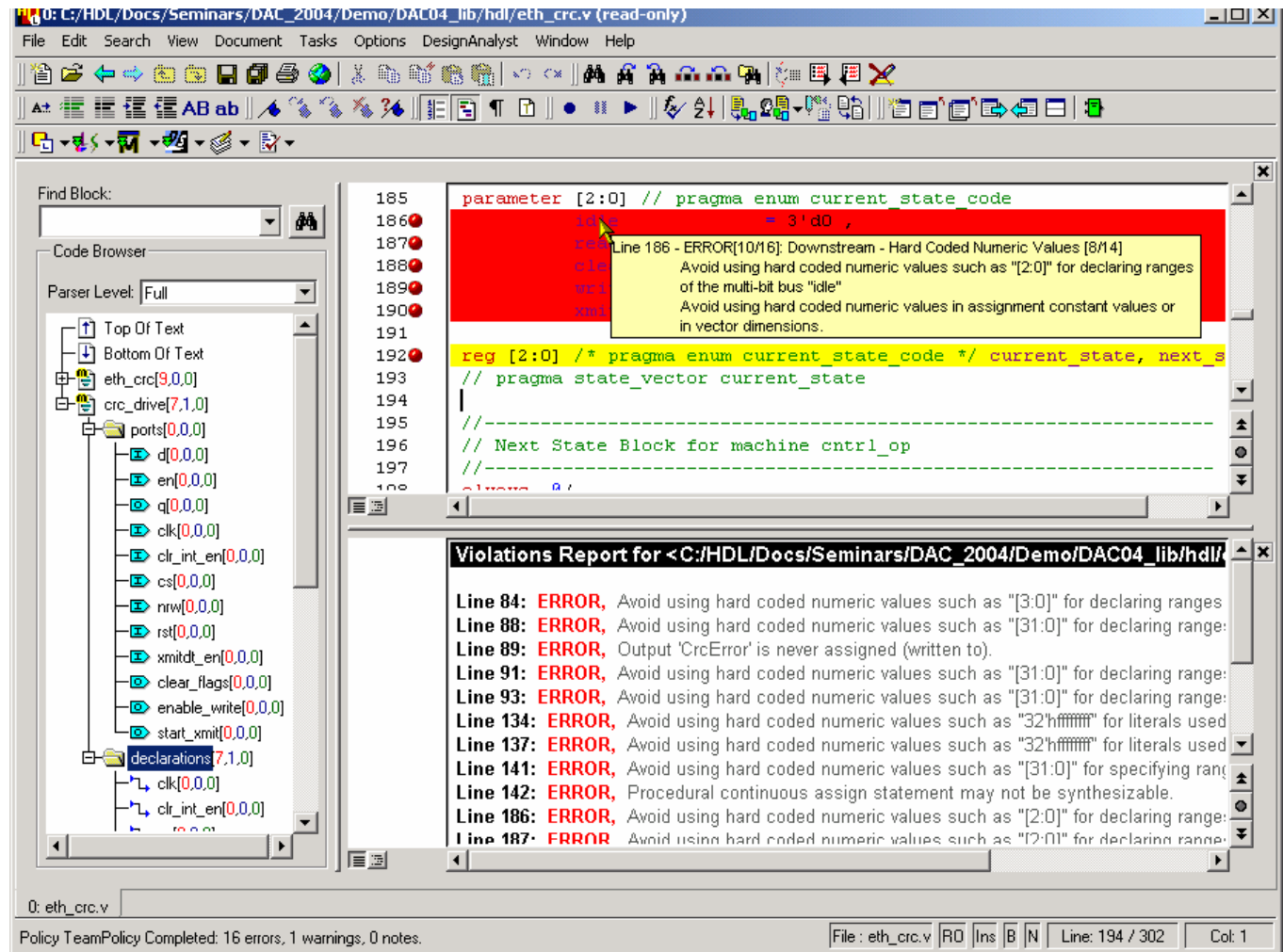
- Group, filter, & sort results
- View only what you want
- Save the views

- See the code fragment & Msg.
- Cross reference
- View message & rule help
- Disable rules

- High-level summary
- Totals rolled up through groups
- Expand, collapse, & cross-ref

Analyze Results within Text Editor

- ✓ Code Browser indicates errors & warnings
- ✓ Violation report provides navigation
- ✓ Code lines highlighted
- ✓ Hover help for each violation
- ✓ Step through errors
- ✓ Show rule/rule help
- ✓ Rerun analysis



View Reports

- Export Summary report as CSV, TSV, or HTML
- Export Result Table as CSV, TSV, or HTML
- Export rules used in ASCII

	A	B	C
1	Design Root		
2	Library:	Ethernet	
3	Primary:	eth_fifo	
4	Secondary:	eth_fifo	
5	Master Clocks	clk	
6	Master Resets	rst	
7	Depth:	Single	
8	Violations: 23		
9	Number of violations for each scope:		
10	Errors	0	
11	Notes	0	
12	Warnings	23 from 7 Rule/s	
13	Number of violations of each severity		
14	File	0	0%
15	Unknown	0	0%
		0	0%
		0	0%
		0	0%
		23	100%
		0	0%
		0	0%

Rule Severity	Severity, Ruleset and Rule	Library, Design Unit and Scope		
Warning	Warning : 2.1 - General Naming Conventions - Ethernet, eth_fifo, Module 2.1.9 - Reset name prefix		Reset signal "reset" violates naming convention: use	0
Warning	Warning : 2.1 - General Naming Conventions - Ethernet, eth_fifo, Module 2.1.11 - Descending bus order		Dimension definition "[0:DEPTH-1]", for "fifo", do	0
Warning	Warning : 3 - Portability - 3.2.1 - Avoid hard-coded numeric values	Ethernet, eth_fifo, Module	As a signal that is not a constant, "rst"	0
Warning	Warning : 3 - Portability - 3.2.1 - Avoid hard-coded numeric values	Ethernet, eth_fifo, Module		0

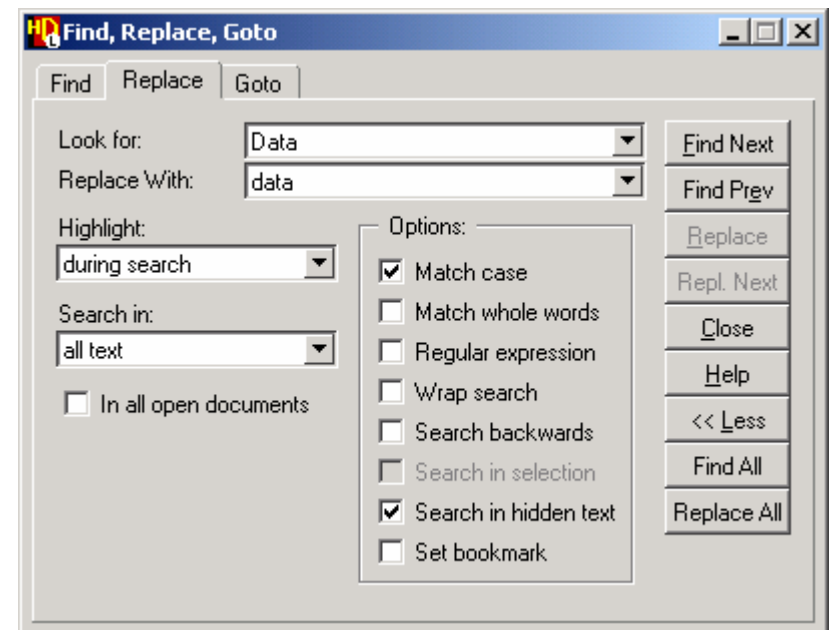
RuleSet: 4 - Clocks & Resets/4.6 - Internally Generated Resets
 Rule Name: 4.6.2 - Isolate Condition resets
 Base Rule: Conditional Resets
 Name: 4.6.2 - Isolate Condition resets
 Severity: Warning
 Language: VHDL Any, Verilog Any
 Hint: If conditional resets are used, isolate into a separate module
 Short Description: If conditional resets are used, isolate into a separate module

Assisted Violation Correction

- ✓ If a tool knows exactly what the problem is & how to fix it, then



- ✓ A good percentage of rule violations fit this scenario
- ✓ The tool should have modes of correction: “do it”, step through & change, etc.



Challenges and Limitations

- ✓ **Mapping desired design rules into checking tool**
 - Not always obvious match
 - Custom rule creation may be necessary
- ✓ **Tradeoff between run-time tool performance and depth of analysis**
 - Must be fast to be used frequently
 - Deep analysis takes longer
- ✓ **Some rule violations are not detectable by static analysis techniques**

Summary

- ✓ **HDL design checking tools save design time by identifying errors earlier to avoid costly iterations downstream**
- ✓ **Design checkers help ensure that HDL code is reusable from the start**
- ✓ **Key to frequent usage is high runtime performance and interactivity**