

DCDL

The Design Constraints Description Language

An Emerging OVI Standard

 cadence

 Mentor
Graphics



 VHDL
INTERNATIONAL



 XILINX



 VSI
Alliance

 IBM

■ Key Points

- This is an OVI standard
- VSI and SLDL have endorsed the OVI effort
 - Virtual Socket Interface Alliance
 - System Level Design Language Initiative
- Cadence, IBM, Mentor are part of the ASIC demo
- Exemplar, Xilinx, Mentor are part of the FPGA demo



What Is DCDL?

- A new language for representing design constraints
 - Timing, area, power, cost, ...
- An OVI standard under development
 - Timing domain draft expected by end of 1999
- A cornerstone for interoperability in next-generation flows

■ Key Points

- Standard will address each constraint domain in turn, starting with timing
- Drafts go to OVI board for approval
- Approved OVI standards generally go to IEEE
- Constraints are one of the last areas not covered by an open standard. Already have
 - function (Verilog, VHDL)
 - library (ALF, OLA)
 - test (STIL)



How Is DCDL Used?

- For Initial Constraint Entry
 - TCL-compatible syntax allows constraints to be embedded in application control scripts
- For Constraint Interchange
- For IP Authoring
 - Tool-independent constraint scripts

■ Key Points

- Synthesis scripts are best example of application control scripts where it makes sense to embed constraints
- Chose TCL because it's becoming the dominant scripting environment
- Constraint interchange is key to interoperability
- Constraints are needed for internals of soft (RTL) and firm (RTL + floorplan) IP blocks, to complete implementing them



What Are We Demonstrating?

- Timing Domain
 - Clocks, arrival times, required times, false paths, multi-cycle paths, input slew times, output capacitances, and operating conditions
- Syntax and Use Models
- Vendor Participation
 - Cadence, Exemplar, IBM, Mentor, Xilinx

■ Key Points

- Timing domain was chosen as the starting point because it's
 - well-understood
 - complicated
 - most urgent
- We'll show syntax examples later
- Two use models
 - DCDL embedded in a synthesis script
 - Pure DCDL exchanged between tools
- Number of vendors participating shows how serious the industry is about defining and supporting the standard



Initial Constraint Entry

- Goal:
 - Enter top level constraints once in the most convenient way, use them in the rest of the flow
- Constraints can be embedded using DCDL syntax in control scripts for tools such as synthesis
 - Tools read the constraints, apply them, then write them out as pure DCDL for use by downstream tools
- Constraints will also be embedded using DCDL syntax in the System Level Design Language

■ Key Points

- Can also capture top level constraints in a GUI
- Can generate lower level constraints through budgeting
- Pure DCDL doesn't use any TCL syntax or application-specific commands



Embedded DCDL Example

```
##
## Check definitions
##
waveform -name master_clk -period 18.0 -edges {0 9.0}
waveform -name inverted_clk -period 18.0 -edges {1.0 10.0}

clock -waveform master_clk Clk

##
## Input section
##
## expanded here
data_arrival_time -waveform master_clk -lead -early -rise
[find -port MemData*]

data_arrival_time -waveform master_clk -lead -late -rise
[find -port MemData*]
```

■ Steps

- With the mouse in the text window, use the right arrow key to step through different sections in the file

■ Key Points

- The demo is based on a subset of the full constraints expected in DCDL
- The embedded example shows how the Ambit "find" command can be used in combination with the DCDL standard to make entering constraints easier and more familiar



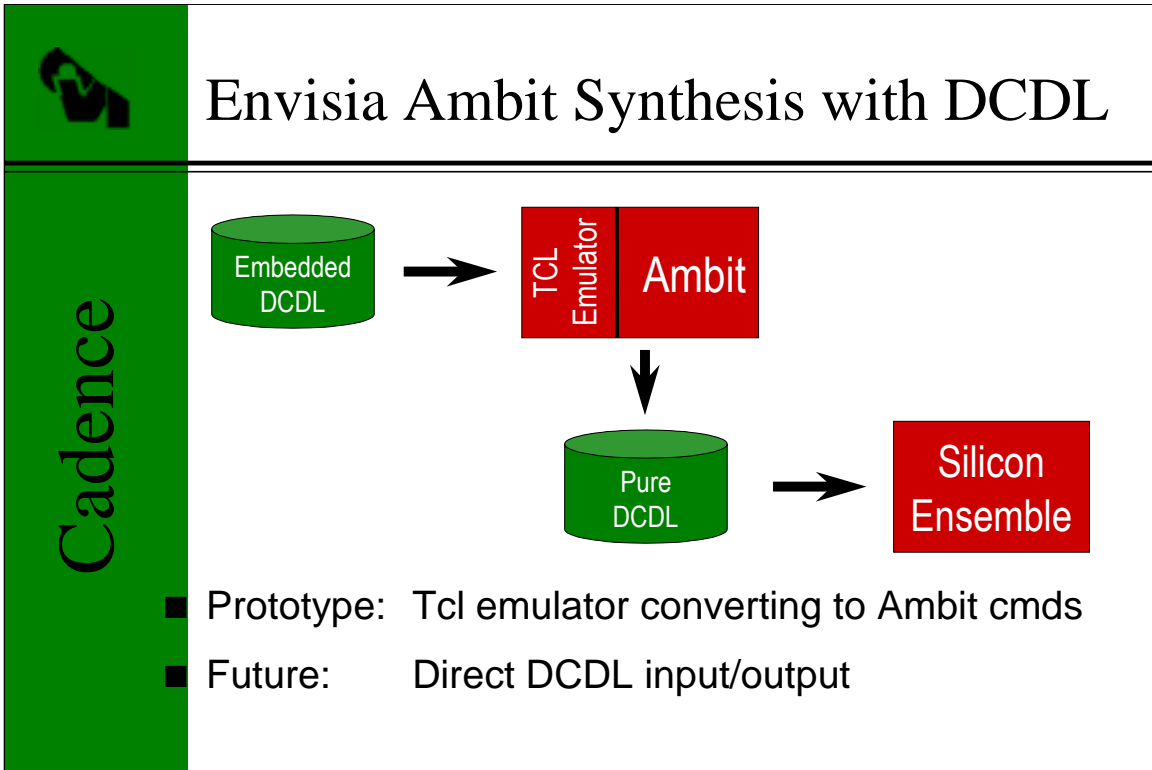
Envisia Ambit Synthesis

Cadence

- A high performance, high capacity, full-chip synthesis solution with fully integrated signoff timing analysis
- Cadence supports DCDL in the front-end as the next-generation solution to timing convergence and interoperability
 - Consistent constraints are a key part of convergence
 - ✦ Inconsistencies lead to cycles worth of inaccuracy

■ Key Points

- With the Cadence Envisia Synthesis tool, you can do top-down design with over 1M gates design and do timing analysis in the same environment.
- DCDL is based on the Ambit constraint language as a strawman
- Synthesis is one of the main entry points for entering, refining constraints
- Even with the best tools, timing analysis and synthesis results are only as good as the input constraints
- Interoperability based on syntax alone is insufficient; DCDL standard will have fully address semantics
- As an entry point in the ASIC design flow, it is important to capture and pass constraints in a consistent manner to safeguard design accuracy and convergence. That's why we implemented DCDL support.



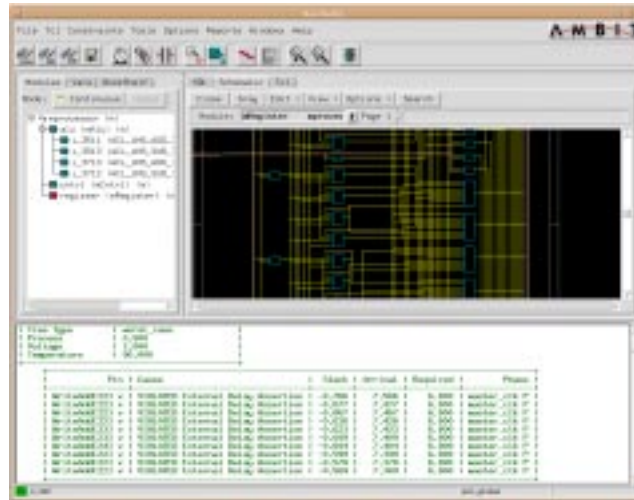
■ Key Points

- Embedded DCDL can make use of the full power of TCL and the application commands, such as find
- TCL emulator is a set of TCL functions that correspond to DCDL statements
- Each emulator function interprets pure DCDL syntax
 - The TCL shell expands variables and command invocations before calling the emulator function
- Each expanded DCDL statement is converted into existing BuildGates constraint commands
- The existing BuildGates write_assertions output is post-processed to create the pure DCDL file



Envisia Ambit Synthesis Interaction

Cadence



■ Steps

- The demo startup will leave BG with the design already loaded. A zoomed, schematic view of the top-level of the design should be displayed.
- Issue the command “demo” on the command line to load the DCDL constraints
- Perform a timing analysis by selecting:
Reports->Timing... then click the Report button
- Close the Path display window before moving to the next step in the demo

■ Key Points

- Once the DCDL constraints are read into Ambit using the synthesis script, all of the normal operations in the tool will make use of them
- We intentionally synthesized at a different operating point to illustrate how the signoff timing analysis in BG can be used to detect post-layout violations, then fix them automatically (although we're not actually doing this step)



Pre-Layout Timing Analysis

- DCDL provides inputs for delay calculation ...
 - Input slew and output capacitance
 - Wire load model selection
 - Operating conditions
- and for timing analysis
 - Clocks, arrival times, required times, false, multi-cycle paths

■ Key Points

- Pre-layout timing analysis is a key step in the traditional ASIC design flow
 - Want to verify that the design has no timing violations before proceeding to place & route
- DCDL can help prevent mismatches before and after handoff



Pure DCDL Example

```
waveform -name master_clk -period 18.0 -edges {0 9.0}
waveform -name inverted_clk -period 18.0 -edges {1.0 10.0}

clock -waveform master_clk Clk

data_arrival_time \
  -waveform master_clk -lead \
  -early -rise 6.0 \
  { MemData/0/ MemData/1/ MemData/2/ MemData/3/ \
    MemData/4/ MemData/5/ MemData/6/ MemData/7/ \
    MemData/8/ MemData/9/ MemData/10/ MemData/11/ \
    MemData/12/ MemData/13/ MemData/14/ MemData/15/
    MemData/16/ MemData/17/ MemData/18/ MemData/19/
    MemData/20/ MemData/21/ MemData/22/ MemData/23/ }
```

■ Key Points

- In the pure DCDL file, the design objects that were identified using the Ambit find command have been explicitly specified



IBM ASICs and EinsTimer

IBM

- As design complexity, size, timing demands and number of tools explodes, standard methods for exchanging chip design data grows in importance
- IBM ASICs is frequently asked by customers to support various timing tools
- IBM ASICs requires EinsTimer for timing sign-off and "back-end" processing
- DCDL facilitates interaction between numerous styles of design flows and methodologies
- IBM is committed to the development of industry standards

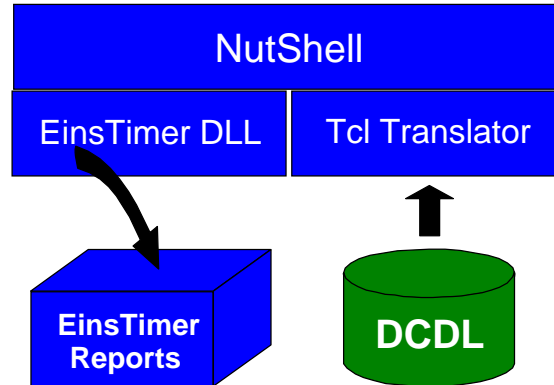
■ Key Points

- EinsTimer (Static Timing Analysis Tool) required for Sign-off process at IBM ASICs
- Requests for Pre-Layout Sign-Off using external tools.
- Process of translating design constraints is tedious and error prone
- DCDL will address such interoperability issues



EinsTimer, NutShell and DCDL

IBM



- Prototype: Tcl Translator running under NutShell
- Future: A DLL for reading DCDL

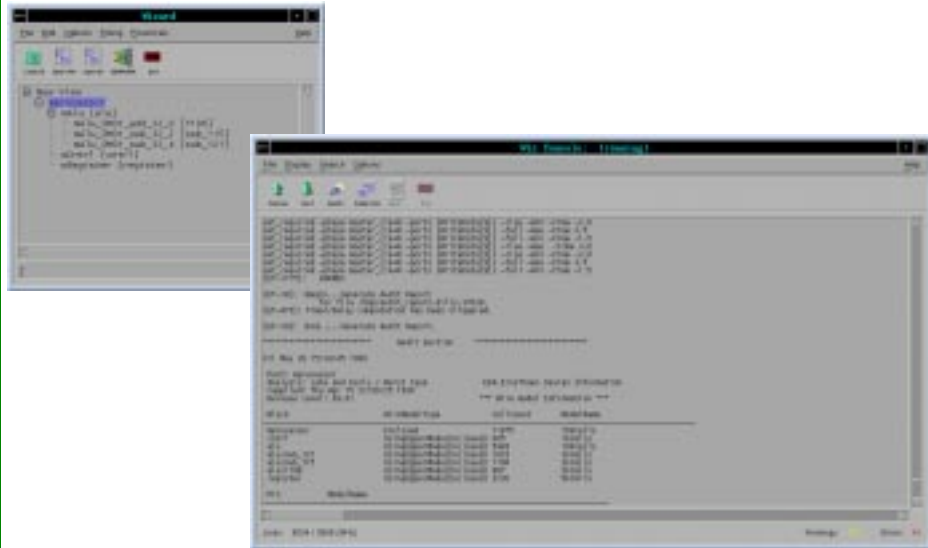
■ Key Points

- Expanded DCDL Design Constraints converted to generic timing database
- Set of TCL procedures read database to write out EinsTimer commands
- TCL based translator runs under Nutshell
- Envisioned to be a separately loadable DLL



EinsTimer Interaction

IBM



■ Steps

- Place cursor in lower bottom of the Wizard window
- Type "load_design"
- Type "demo"
- Move Cursor in BCONSOLE window to the start of the Audit Report Section

■ Key Points

- Timing analysis constraints read from DCDL
- Point to special paths section in Audit Report
- Point to Phase Definition in Audit Report
- Point to Required Arrival times and Cap data in Audit report



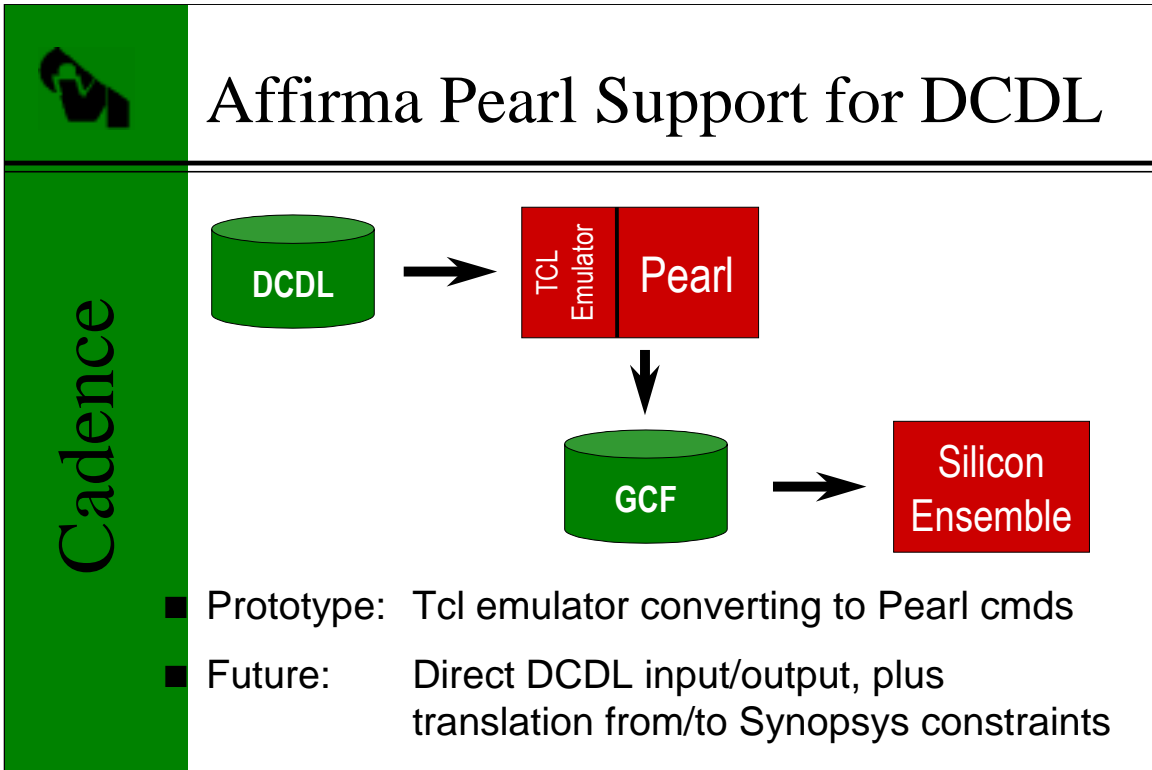
Affirma Pearl Timing Analysis

Cadence

- Cadence's solution for detailed back-end timing analysis, highly integrated with floorplanning and place & route
- Pre-layout analysis is done as the first post-handoff step, for constraint and netlist consistency checking
- Cadence supports DCDL in the back-end as the next-generation solution to timing convergence and interoperability

■ Key Points

- Today, constraint interoperability is a key issue in the handoff to place & route
 - Constraint translators can be a source of inefficiency and errors
 - Translators aren't robust when there are semantic mismatches
- DCDL addresses both syntax and semantics
 - Eliminates the need to translate altogether
 - Ensures that back-end interpretation matches front-end



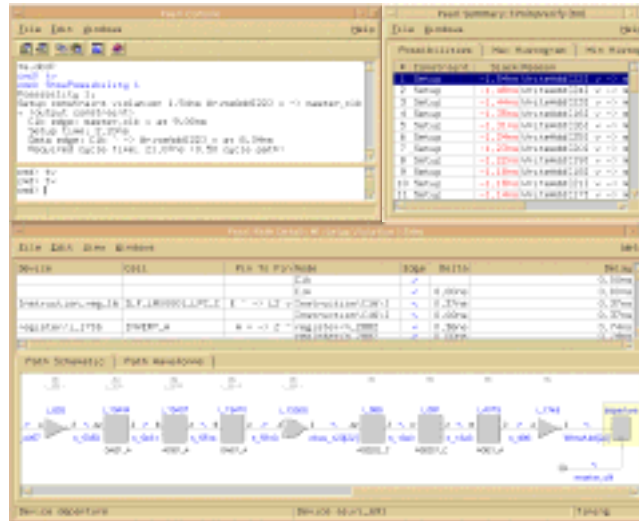
■ Key Points

- The Ambit and Pearl TCL emulators are similar
 - Each expanded DCDL statement is converted into existing Pearl constraint commands
- Pearl can support both embedded DCDL and pure DCDL
- DCDL is expected to replace GCF in the future as the standard constraint format in the Cadence back end



Affirma Pearl Interaction

Cadence



■ Steps

- Click on the Read DCDL toolbar icon (second from left)
- Select "expanded_constraints.dcdl" in the file dialog
"Pearl has just read .."
- Type "tv" in the command entry area
"Pearl's TimingVerify function is being used to analyze the design"
- The path schematic view should show an output path to WriteData[22]
- Left-mouse click on the "Path Waveforms" tab to reveal the timing diagram. The blue arrow shows that output timing requirement derived from the DCDL *data_required_time* and the clocks that are defined.

■ Key points

- Pearl is doing pre-layout delay calculation using the input slews, wireload estimation, and output loads from DCDL
- Timing analysis is using clock definitions, arrival times, and required times, as well as false and multi-cycle paths
 - Point out the required time in the Path Waveform view



Envisia Silicon Ensemble

Cadence

- Constraint-driven algorithms are the heart of the Envisia family of physical implementation and optimization tools
- Eliminating semantics differences will further improve convergence
- Silicon Ensemble was used to complete place & route of the design
- Prototype: Translation from DCDL to GCF
- Future: Direct DCDL input/output

- Key Points



Post-Layout Timing Analysis

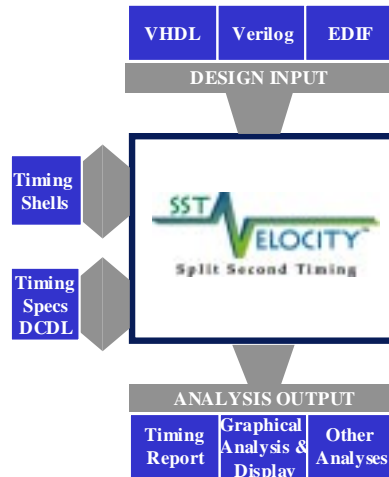
- DCDL provides inputs to delay calculation and timing analysis
 - Similar to pre-layout analysis
- Extracted parasitics are used instead of wire load models
 - Consistent constraints between logical and physical implementation help reduce or eliminate timing failures
 - Some timing violations are shown here to illustrate interactive tool capabilities

■ Key Points



SST Velocity

Mentor Graphics



- Next-generation, high performance algorithms
- Incremental analysis for improved total verification cycles
- Simple verification of complex designs
- A new standard in ease of use for static timing

■ Key Points

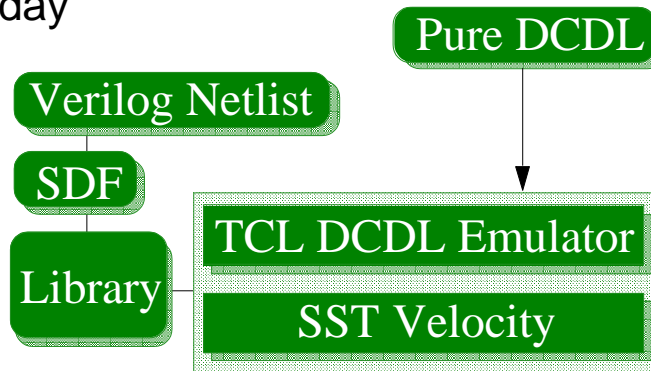
- In today's demo, we will also be using SST Velocity from Mentor Graphics to perform post-layout static timing verification.



SST Velocity Support for DCDL

Mentor Graphics

■ Today



■ Status

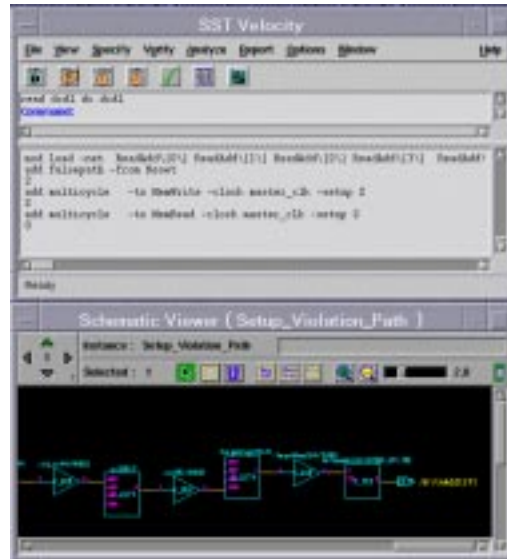
- The SST Velocity team is committed to industry standards and is actively involved in the definition of DCDL

■ Key Points

- Today we are reading the original, pure DCDL file into SST Velocity via a TCL DCDL emulator. This emulator maps DCDL into SST Velocity commands.
- "The SST Velocity team is actively working with several standards groups, including the DC-WG to ensure tool interoperability. When DCDL is supported, the emulator will be replaced with internal code.



SST Velocity Interaction



■ Steps

- "We have already loaded the library, design, SDF, and TCL DCDL emulator to save time."
- "First we will read in the original DCDL file."
 - <At the command line, execute: read_dcdl dc.dcdl
 - The demonstrator could point out in the transcript that the DCDL is mapped into Velocity commands
- "Next, lets make sure that the constraints were applied."
 - <Execute the menu pick: Report > Timing Specifications > All
 - and scroll thru the report - pointing out the constraints>
- "Lets examine the most critical path."
 - <Execute the menu pick: Report > Timing Violations > Setup Constraints
 - Click on "Generate schematic from result" to get the schematic path.>
 - <Grow the timing report window.>
 - <Point out the slack violation>

■ Optional steps:

- <In the schematic window, zoom in on the path by clicking the + in the magnifying glass button.>
- <Right mouse click and select Object Query. This allows an info box to appear when you pass the cursor over the path elements>



Post-Layout Analysis with Pearl

Cadence

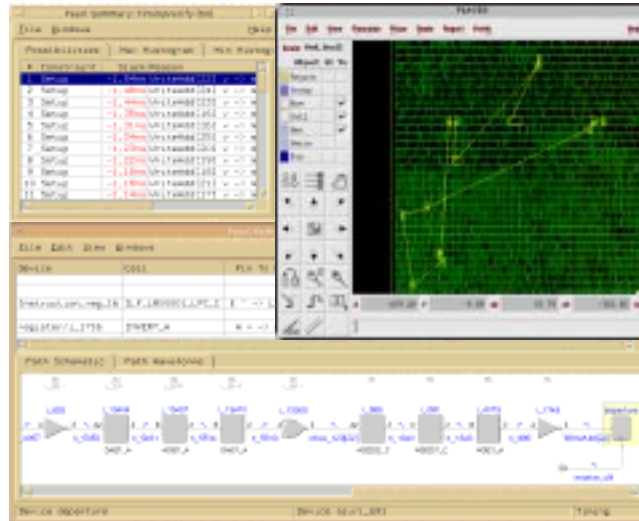
- Using DCDL as input, Pearl supports
 - Delay calculation with extracted parasitics
 - Timing analysis
 - Interactive queries
 - Cross-probing of critical paths with Silicon Ensemble

■ Key Points



Affirma Pearl & Envisia Silicon Ensemble Interaction

Cadence



■ Steps

- Left-mouse click on Possibility #1 in the Pearl Summary window; this will draw a detailed path schematic and cross-highlight into the layout display
- Click on Possibility #2 to see a new path
- Click on an instance or net in the path to see it blink/highlight in the layout display

■ Key Points

- To achieve a convergent timing flow, it is critical to analyze that timing after each step of the implementation, and to drive the next step with updated information
- This step shows multiple tools in use at a step in the implementation flow. It is critical for the tools performing the specific implementation steps to have an identical view of the constraints as the timing analyzer reporting on the results.



Summary

- DCDL provided constraint interoperability across an entire ASIC design flow
 - From RTL through post-route analysis
 - 5 different tools from 3 vendors
- The full set of DCDL timing constraints will be available as a draft by the end of 1999
- Formal tool support is expected in 2000