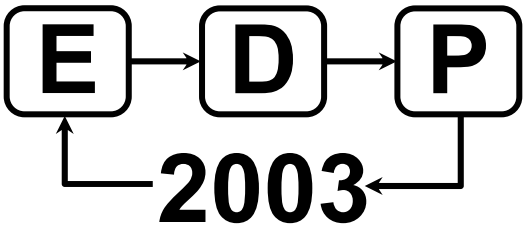
IEEE COMPUTER SOCIETY

Workshop notes

Design Automation Technical Committee
DATC



The Tenth IEEE/DATC Electronic Design Processes Workshop

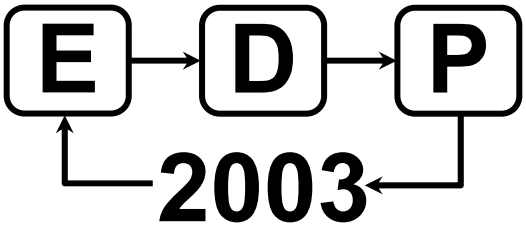
Monterey Beach Hotel, Monterey, California
April 13-15, 2003

Sponsored by

IEEE Computer Society DATC

In cooperation with

ACM Special Interest Group on Design Automation (SIGDA)



Foreword

Welcome to the Tenth IEEE DATC Electronic Design Processes Workshop (EDP 2003), an international meeting held under the auspices of the IEEE Computer Society and the Electronic Design Process Subcommittee (EDPS) of its Design Automation Technical Committee's (DATC).

Since you have chosen to participate in this year's workshop, I'm sure you recognize that the days in which one could gather a set of good point tools and quickly put out a high quality electronic design are behind us.

Today's technologies and system requirements are introducing many new problems to the design process and are causing greater amounts of interaction between previously independent design domains. As a result, efficiently achieving a high quality design demands as much if not more focus on the overall design process as on the capabilities of tools in specific domains.

Our focus this year is still on methodology, and we have sessions dealing with the new tools and methodologies automating the Power User design flow and also we are taking a look at the API vs. Data Model vs. Data Base issue for the new IC Implementation Tool Sets.

The importance of process and methodology is our industry growing, and we think that the importance of a workshop like this will only increase in the future.

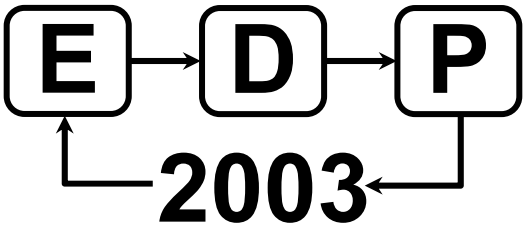
We hope you will take back the insights you may gather here, apply them in your own work, and come back next year with new insights to share in this important area.

So, thanks for your participation, and enjoy the workshop.

Gary Smith
EDP 2003 Chair

Table of Contents

Non-presented papers



Organizing Committee

General Chair	Gary Smith (*Gartner/Dataquest*)
Technical Program Chair	Juan-Antonio Carballo (*IBM Research*)
Publicity Chair	Steve Grout

Steering Committee

Juan-Antonio Carballo (*IBM Research*)
Aparna Dey (Cadence)
Steve Grout
Dwight Hill (*Synopsys*)
Andrew B. Kahng (*UCSD*)
José Augusto D. F. Lima (*Univ. of Minho*)
Stefanus Mantik (*Cadence*)
Gabe Moretti (EDN)
Naresh Sehgal (*Intel*)
Gary Smith (*Gartner/Dataquest*)

Keynote 1:

Communications 101 for EDA

Steve Schulz (*SI2*)

Communications 101 for EDA

Steven E. Schulz
President & CEO
Silicon Integration Initiative

April 14, 2003
Electronic Design Process Conference
Monterey, CA

'Processes are for Beaurocrats!'

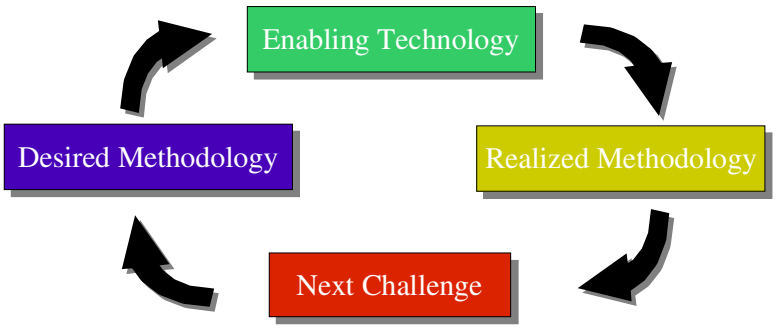
- **'Processes are just excuses used by beaurocrats to avoid doing real work!'** **-- Jason Jennings, from *Less is More***
  - Oh well, he's just a journalist
- **Processeses and methodologies are, in truth, our 'corporate memory' for achieving consistent results**
  - Thus, all businesses need them to varying degrees
  - They provide a recipe for Output=F(Input)
- **The greater the dependencies and interactions, the more complex the resulting process**
- **IC design flow ependencies also demand complex communication among tools**

Design Flow Methodology

- **Lots of abstractions: Math, Event, Logic, Boolean, Physical, Polygon, Optical/Chemical (physics)**
  - 80% / 20% rule favors front-end focus, but...
  - Attention returns to physical/physics challenges at each new node, must revise old methodologies
- **Top-down intent meets bottoms-up physics at each level**
  - Implementations (solutions) always bring bottoms-up constraints
  - Knowledge of intent useful (or critical) downstream
  - Need to pass information upstream, too
    - Is this just a “model”? What about dynamic dependencies?

Enabling Methodology Choices

- **IC design and manufacturing methodologies are becoming inter-dependent**
- **Methodology choices are constrained by implementations**



- **The trick to enabling new IC flow methodology choices is better *communication* among the tools**

130nm Frustrations Only the Beginning

News

Trouble at 130-nm node causes finger pointing

By Stephan Ohr and Ron Wilson
EE Times
June 14, 2002 (5:33 p.m. EST)

What about 90nm?

(SNIP)

Some designers, meanwhile, voiced their frustration over bottlenecks in design flows and pointed a finger at EDA vendors. The design flows are broken, said users like John Szetela, manager for tools integration at Advanced Micro Devices Inc., and Hilton Kirk, manager of physical design at Philips Research Labs. "Late changes in the design flow are inevitable ☐ and the automated design flow is too slow to accommodate them," said Szetela. Point tools for analyzing postlayout parasitics ☐ IR drop, electromigration, antenna effects, signal integrity ☐ conflict with timing-analysis tools, he said.

(SNIP)

90nm: Chip Design At The Crossroads

- **The challenges predicted for 90nm design are now upon us**
  - Very compelling, and substantially worse than 130nm design*
- **Many file formats have reached the breaking point**
  - 2GB / 4GB file size limits on SPEF, SDF, LEF/DEF, GDSII,...
  - 32-bit no longer sufficient for large die area and small features
  - Total (SoC) chip design data >80GB @ 130nm (w/ current flows)
    - Predicted to exceed 200GB @ 90nm (w/ current flows)
- **Co-dependent processing required among multiple concerns**
  - No single supplier solves 100% of flow needs
- **File format conversions already wasting too much time**
  - Ex.: 110 min. for PDEF R/W with commercial layout tool*
  - Direct access is critical to avoid unacceptable project delays

Resolving Barriers in IC Design Flows

- **Exploding Data Sizes:**
  - Eliminate data redundancy
  - Smarter data structures
- **Increasing Cycle Times**
  - Avoid flow iterations
  - Faster access to required data
- **Design Capacity, Performance Limitations**
  - More efficient storage
  - Direct access to specific data
  - Best-of-breed tool algorithms
- **Co-dependent Processing Steps**
  - Fine-grained data interaction
  - Means for shared session control
- **Multi-Vendor Integration**
  - Common, extensible, high-bandwidth interface
- **File Format Interface Revisions**
  - Separate stable interface from evolving content

EDA Tool Comunications 101

- **What do tools need to communicate with each other?**
  - Shared protocol
  - Shared semantics
  - Access to (only) desired data
  - Awareness of change to data
  - Persistence
- **EDA has lacked a common 'language' for tools in flows to communicate... until now**
- **But wait, it's even more challenging!**
  - Mask-level interdependencies with design now demand new methodologies that require communication across disciplines

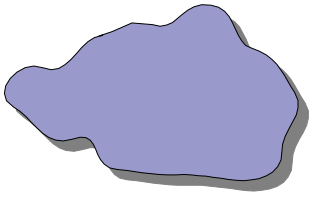
What Makes This So Hard?

- Efficiency -- How to minimize access latency?
- Granularity
  - Design data sets are huge, multi-dimensional
  - Tool algorithm access needs varies widely, and are NP-complete
- Capacity – Indexing, compression w/ varying design data structures
- Robustness -- Represents data needed by current tool algorithms
- Concurrency -- Shared runtime memory access, synchronization
- Multi-Processing – Leverage parallel HW for big design tasks
- Flexibility – Adapts easily to new data requirements, custom tools
- Versioning – Everything must evolve but still retain interoperability
- Cross-disciplines – Different worlds and semantics make it harder

Tool Communication Architectures

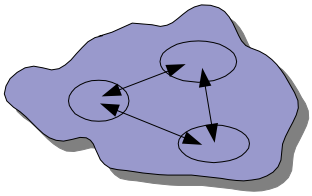
I. Single executable process (e.g. one big tool)

- (+) Shared memory, optimized
- (+) Direct data access
- (-) Fixed, rigid methodology
- (-) Lacks customization flexibility
- (-) Forces sequential processing



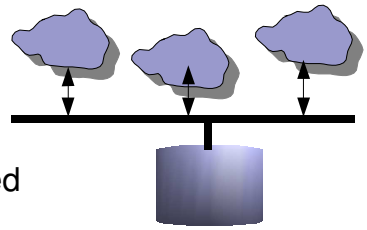
II.Multiple executables, direct MMAP sockets

- (+) Shared memory
- (+) Direct data access
- (+) Loads only algorithms / structures as required
- (-) Tool set compiled as single, fixed cluster
- (-) Cannot support distributed processing



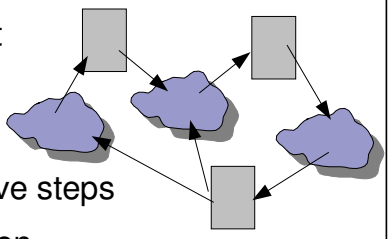
Tool Communication Architectures

III. Multiple executables, defined protocol to common interface layer



- (+) Supports shared memory (and persistence)
- (+) Direct data access
- (+) Loads only algorithms / structures as required
- (+) Methodology and customization flexibility
- (+) Supports multi-threading, distributed processing
- (-) Interface implementations must be consistent

IV. Separate file formats between executables



- (+) Relatively simple to define
- (-) Poor performance on large data sets, repetitive steps
- (-) Very poor performance with coupled interaction

EDA Interoperability Geneology

Process level integration

Design to Manufacturing Data Sharing

Multi-vendor integrated systems

Open Data API

2002

Single vendor integrated systems

Proprietary Data APIs

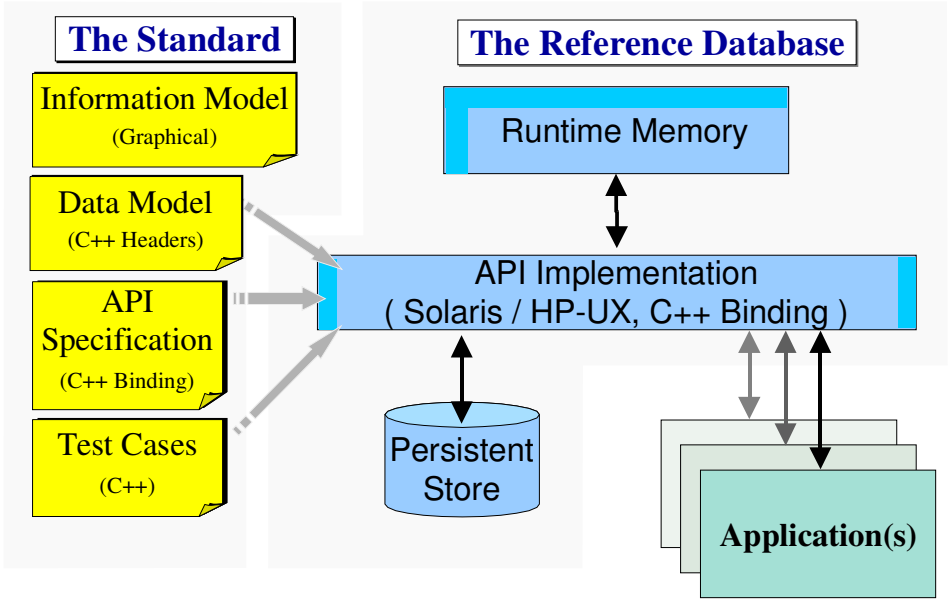
Loosely coupled systems

Open File Formats

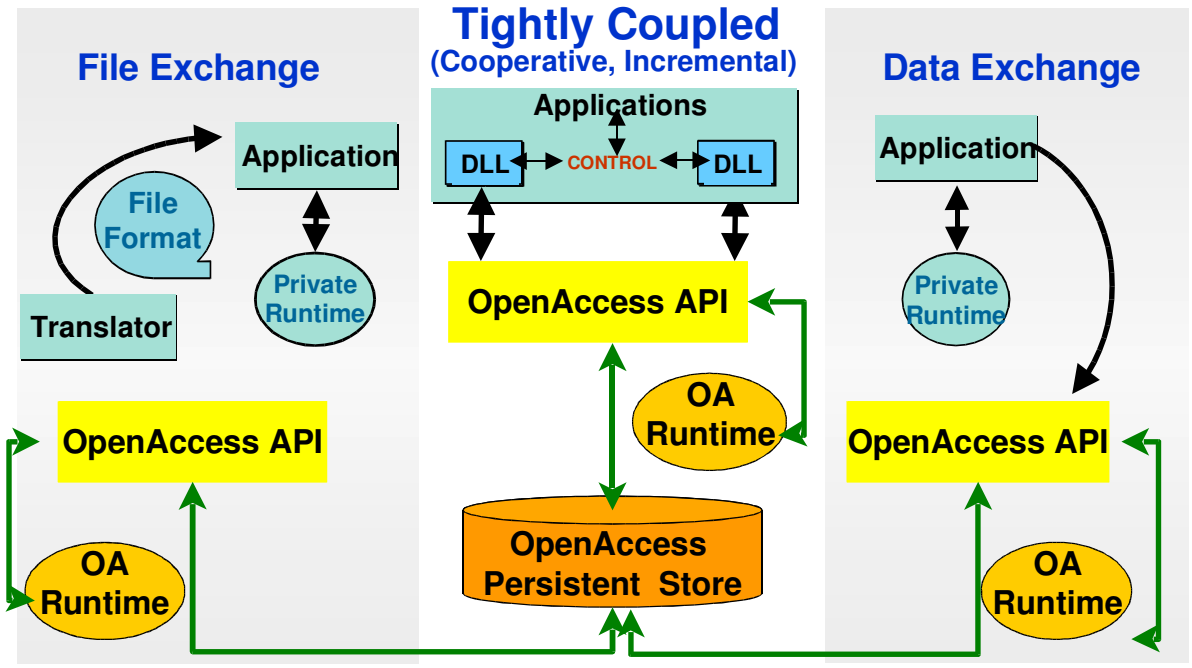
Tool-Data Binding

Proprietary File Formats

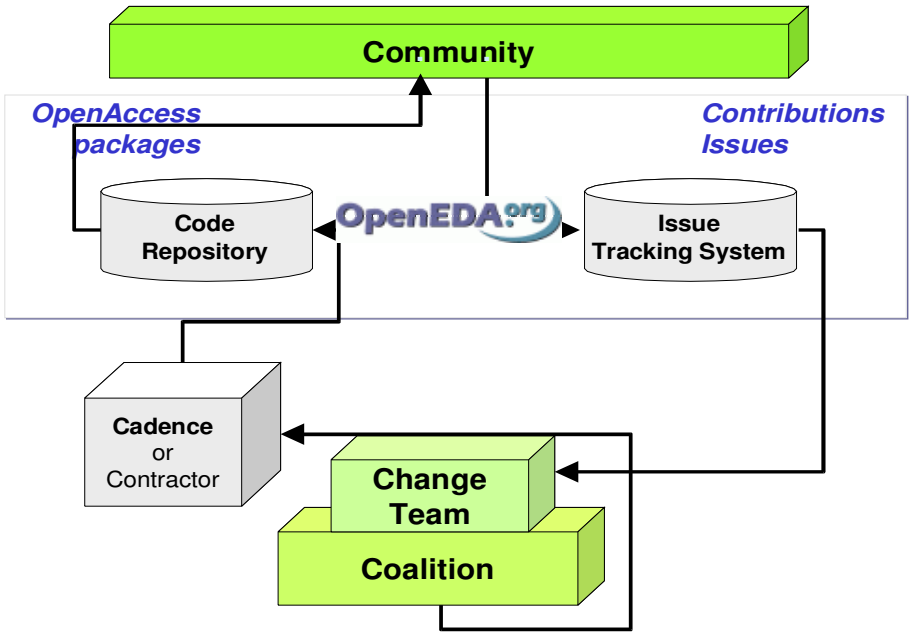
Open Access Deliverables



Application Integration



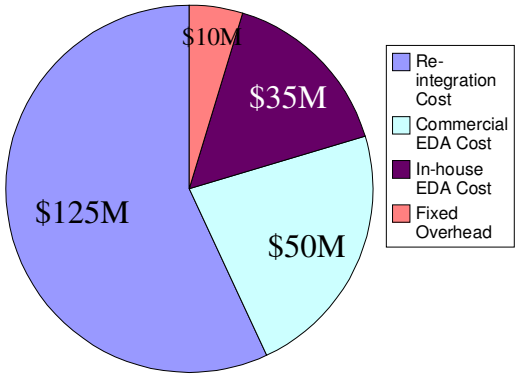
OpenAccess Process



Improved Cost Efficiencies

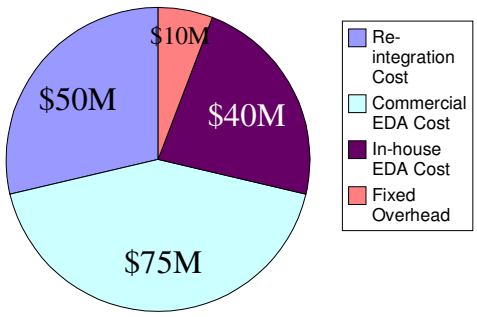
- Customers waste less on re-integration costs
- More potential for commercial EDA tool purchases
- Example: Corporate EDA budget of $220M

Current Situation



Total: $220M

With OpenAccess



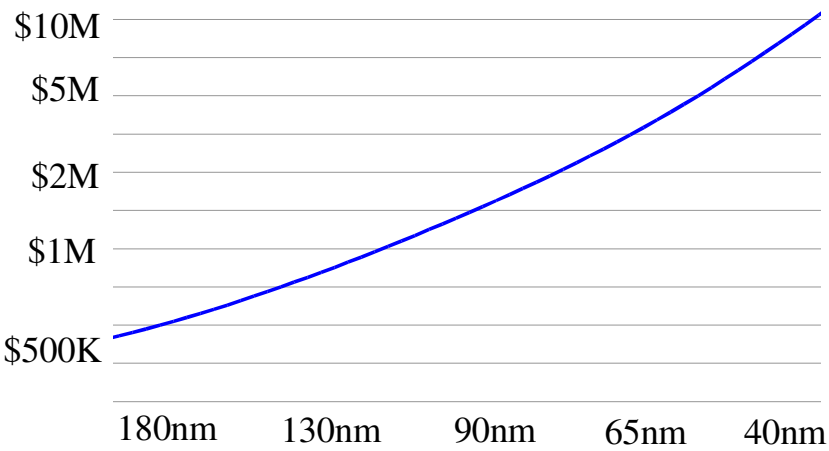
Total: $175M

Directions Ahead: OASIS, UDM

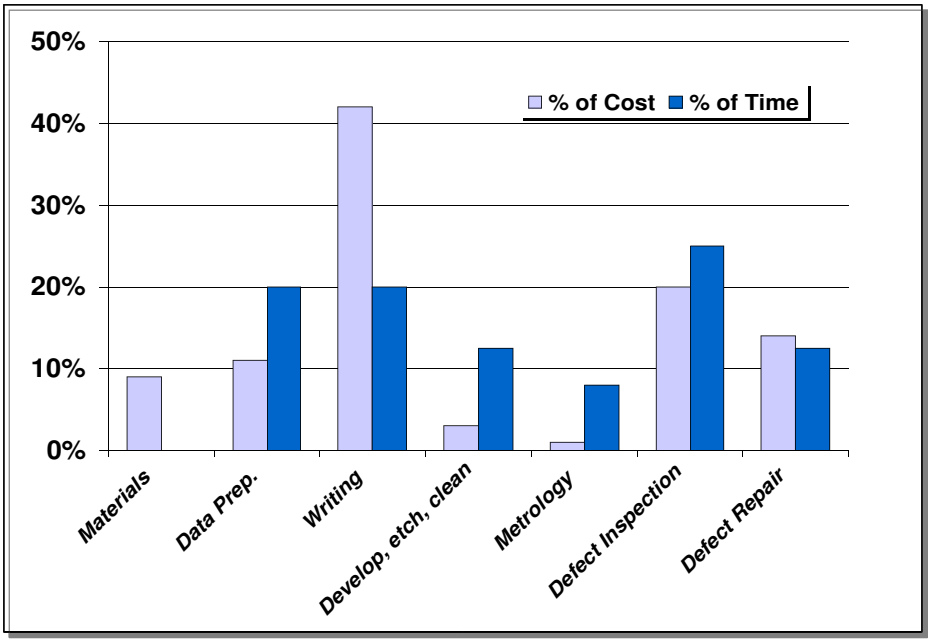
- **Use of OPC / PSM compounding the data explosion problem**
  - larger die area ∩ smaller features ∩ more accuracy ∩ RET
- **IC mask costs rising out of control, threatens industry growth**
- **Advanced data handling for new mask tools (laser, e-beam)**
- **What's Needed:**
  - Better data efficiency (OASIS ~ 1/10th GDSII data size)
  - Higher bandwidth, direct access (OA)
  - Smarter data representations (OA)
  - Bi-directional design intent exchange (OA)
- **Potential for $4B-$6B industry savings if solved (source: SEMI)**
- **SEMI has endorsed OpenAccess as their technology of choice**
  - UDM = “Universal Data Model” (mask, test, yield improvement)

Mask Set Cost Escalation

- **Mask cost is becoming a major impediment for low volume ASICs**
  - At 90nm: $1.5M for typical mask set (up 85% over 130nm)
  - 65nm predictions are far worse...



Mask Cost and Cycle Time Drivers



Source: TSMC

Summary of Major Mask Problems

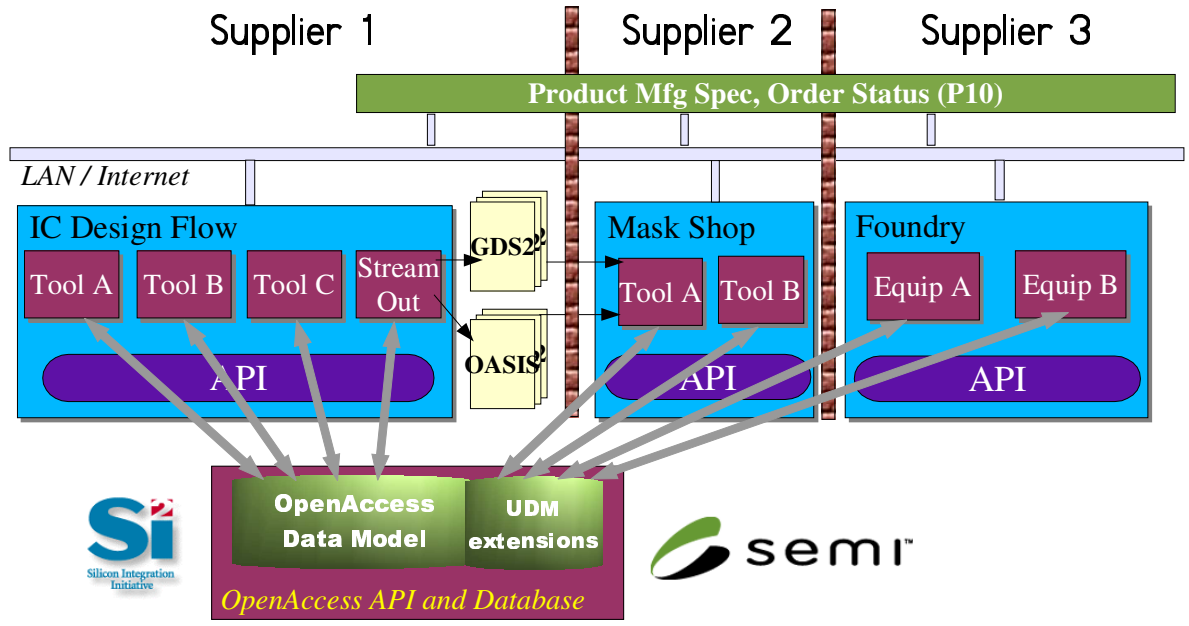
- [Exchange] file sizes too big
- [Exchange] file lacks extensibility/Adoption to change too slow
- Lack of manufacturing feedback to design
- Stream file looses too much design intent
- Ambiguous data model
- Lack of "Best Practices" Guide
- Too many [tool] formats

Source: SEMI EDA-Mask Taskforce 11/6/2001

OpenAccess Features: Benefit to Manufacturing

Feature	Benefit
· Full access to design intent	· Aids diagnosis, analysis and repair
· L/P hierarchies preserved	· Reduces memory
· Identification of OPC	· Adds intelligent for analysis
· Incremental R/W access	· Very high-performance
· Region selection	· Reduces processing
· Thread safe	· Supports parallel processing
· Fully extensible	· Longevity
· GDSII/OASIS translation	· Eases migration, allows coexistence
· Community source model	· Controlled by all partners but without dependence on any

SEMI/Si2 Join Forces To Realize OA-UDM



Open Source, *Plus* Shared Control

EDA Standards: License Model Comparison

Classification of Standard	Rationale	Access		Distribution		Control	
		Rights	Restrictions	Rights	Restrictions	Rights	Restrictions
Open Source	Allow freely derived works, rapid evolution	Free, open use of source or binary code	Unrestricted access to source (1)	Source or binary, all versions of all changes	Changed source must be available, and allow derived works (1)	None	Undefined
Open Evolution ("Open Community")	As above, plus shared control process ensuring stable, converged evolution of standard	As above, with single public source download site	Unrestricted, royalty-free access to source (2)	Binary, including bug fixes and incorporated API changes	For distribution, modified source must be contributed within 30 days (2)	Elected change team with equal voting rights (2)	No single company can control evolution of standard (2)

(1) see opensource.org for complete description
(2) see openeda.org for complete description

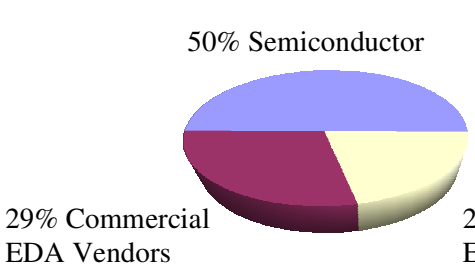
OpenAccess Coalition Members

Customers

Suppliers

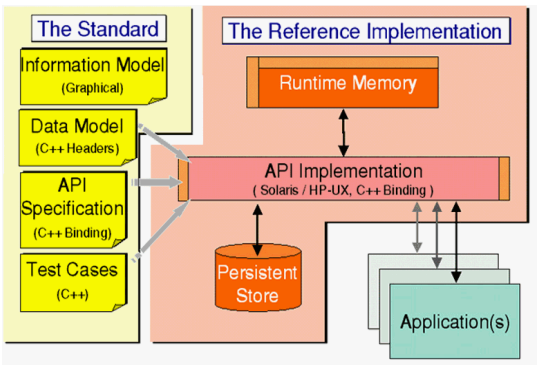
Si2 Mission

Achieve industry adoption of collaborative technology and services that deliver higher levels of silicon design integration, enabling compelling advantages for our members through reduced costs, faster time to market, and improved IC design capability.



Our focus is on the **implementation** of ***collaborative solutions***

Summary



- **Methodology + new technology drives improved IC flows**
- **Improved tool communication is required for new methodologies**
- **Common, open API architecture addresses both performance and flexibility needs**
- **OpenAccess is the *only* open source API standard / database**
- **Si2 and SEMI are partnering to tear down the design-mfg 'wall'**
- **Now is the time to actively support OA and OA-UDM**

OpenAccess: A New Era in IC Design Has Begun

Session 1:

Impact of Design-manufacturing Interface I

Moderator: Juan-Antonio Carballo
(*IBM Research*)

The X Architecture: Roadmap for Design for Manufacturing Methodology

Presentation to Electronic Design Processes
April 14, 2003

Ken Rygler, President
Rygler and Associates, Inc.

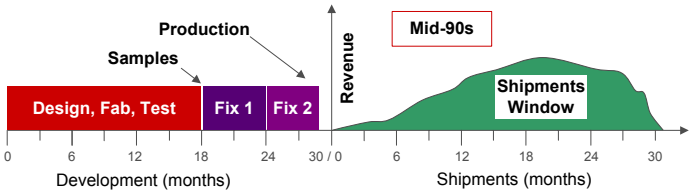
Agenda

- Background
- Industry Challenges
- X Architecture
- X Architecture Supply Chain - DFM
- Test Chip Results
- Summary

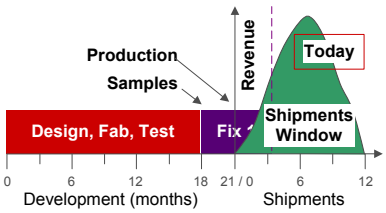
X Architecture DFM History

- Initial customer response positive
  - But can masks be made?
  - What about image fidelity? OPC? CDs?
- DuPont Photomasks produces first 180nm masks
- ASML prints 180nm silicon
- Numerical Technology implements OPC
- 130nm flow developed
- STM announces first test chips

Business Challenge: There is No Market for a Second-to-Market

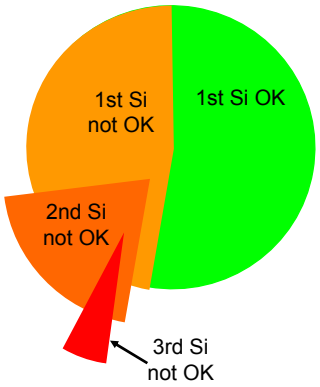


- **Mid-90s: 6 month late → ~31% earnings loss[1]**



- **Today: 3 months late = lose the market!**

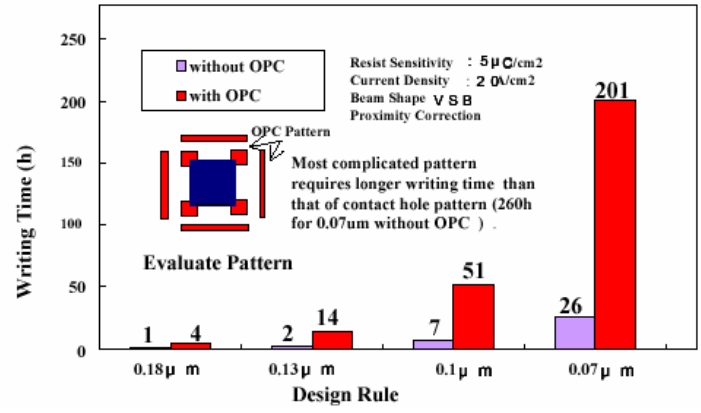
Silicon Failures Are Increasing



- 48% fail on first silicon[1]
- 20% still fail on second spin
- 5% still fail on third spin

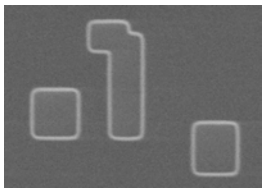
1: Source: Collett International – 2000

OPC Driving Up Photomask Costs

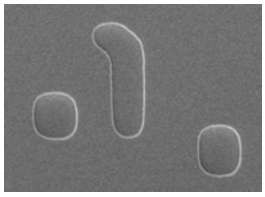


Source : July 2001 Sematech EDA Workshop, Toshiba

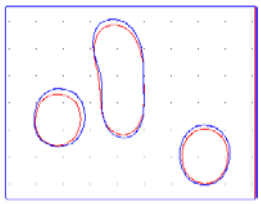
Photomask Patterning Impacts Mask Cost



Ebeam $$$$$

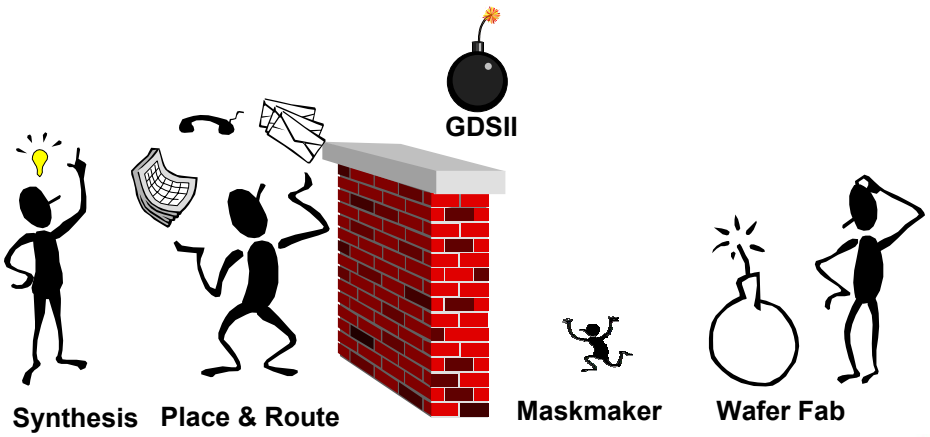


Laser $$$

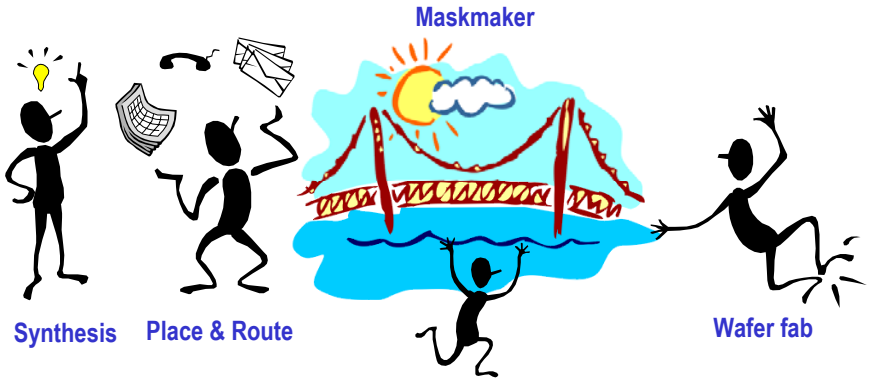


Source: DPI

IC Design and Manufacturing Process Circa 1995

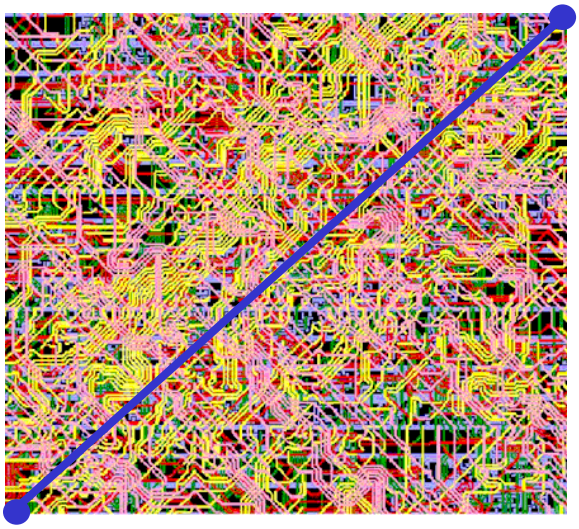


IC Design and Manufacturing Process, ca 2000



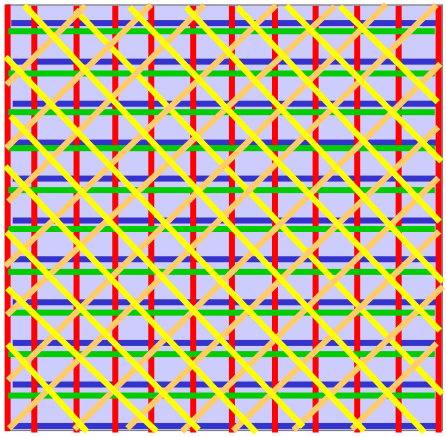
X Architecture: Large-Scale and Small-Scale Diagonals

www.xinitiative.org



- 20+% less interconnect
- 30+% fewer vias

The X Architecture



Metal 5

Metal 4

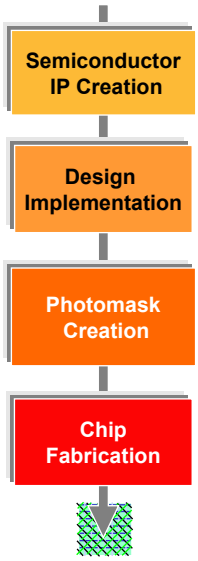
45°

Metal 3

Metal 2

Metal 1

The X Architecture: Roadmap for Design for Manufacturability



	First Availability	Expansion
Semiconductor IP Creation	✓ Confirm compatibility	
Design Implementation	✓ Physical Design ✓ Extraction	Test chips Design methodology
		OPC
Photomask Creation	✓Mask fracture & writing ✓Mask CD metrology ✓Mask inspection	Raster-based Vector-based Inspection
Chip Fabrication	✓ Design rules ✓ Lithography	Design rules Lithography

Technology nodes

130nm Mask-to-Wafer Supply Chain

ETEC
An Applied Materials Company

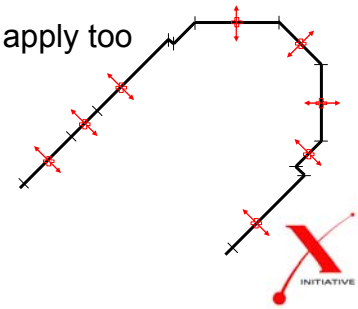
DUPONT
DuPont Photomasks, Inc.

KLA-Tencor

Nikon

Application of OPC on X Architecture 130nm Design

- Used Numerical iN-Tandem hybrid OPC
- Defined a “custom shape” for octagonal line end
  - Ensures limited data volume increase
  - Limits mask complexity and mask cost
  - Other OPC approaches would apply too aggressive OPC with possibly worse corrections



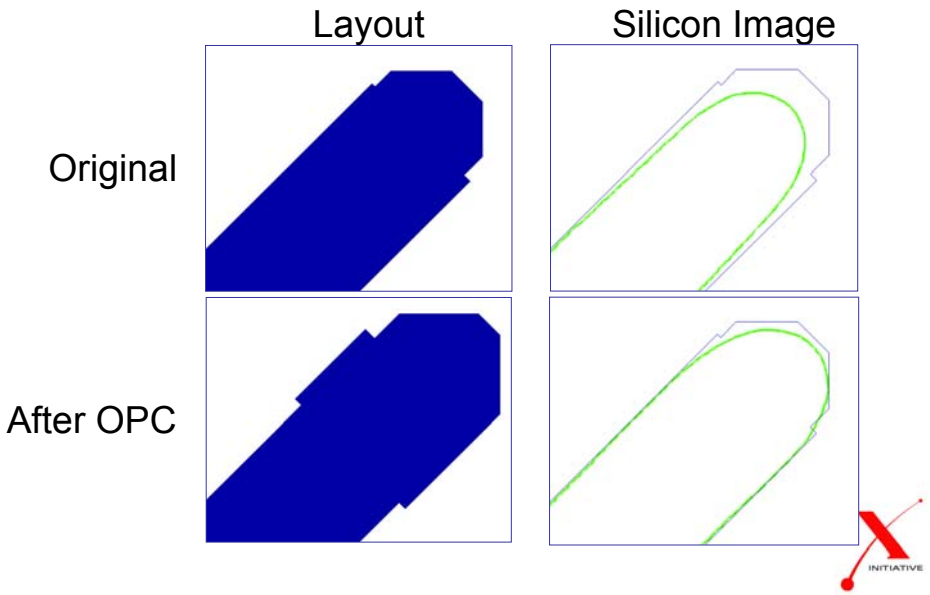
OPC Results

	Layer	Input (MB)	Output (MB)	Run Time (Hrs)	Estimated Production Run Time*
Bright Field	Metal 4	1.58	3.50	6.2	25 min
Dark Field			1.49	2.2	10 min
Bright Field	Metal 5	0.68	1.55	4.2	15 min
Dark Field			0.65	2.3	10 min

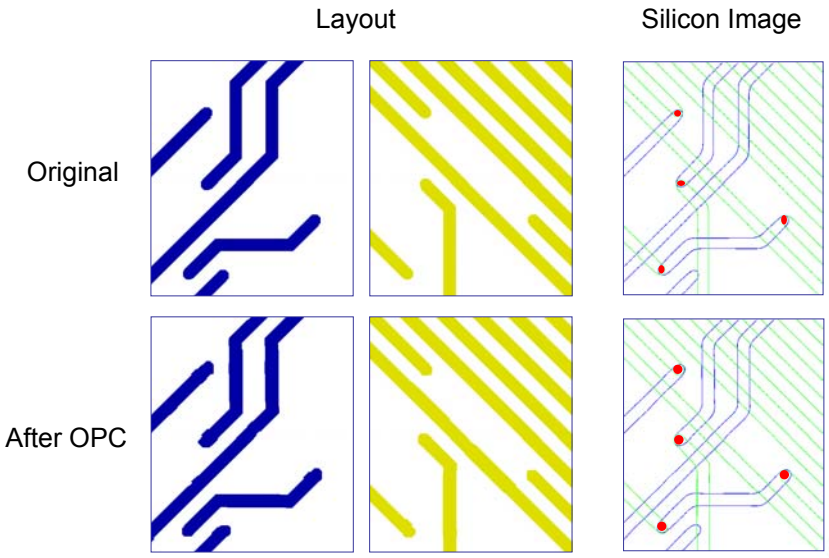
- Run times and output file sizes are well within normal ranges

** Times are estimated based on a 20-CPU distributed processing production environment*

OPC Analysis: Octagonal Line Ends



OPC Analysis: Via Enclosure



Numerical Technologies OPC Conclusions

- Feature sizes below 250nm require resolution enhancement techniques
- OPC is a requirement for manufacturing of X Architecture metal layers with octagonal features
- Appropriate OPC approach and tools can produce run times, and output sizes consistent with Manhattan designs
- Analyses confirm manufacturability and yield

DuPont Photomasks Produces First 130nm Masks

- **Results confirm manufacturability**:
  - Confirms Etec's ALTA 4000 raster laser writer
  - Confirms optimization of inspection

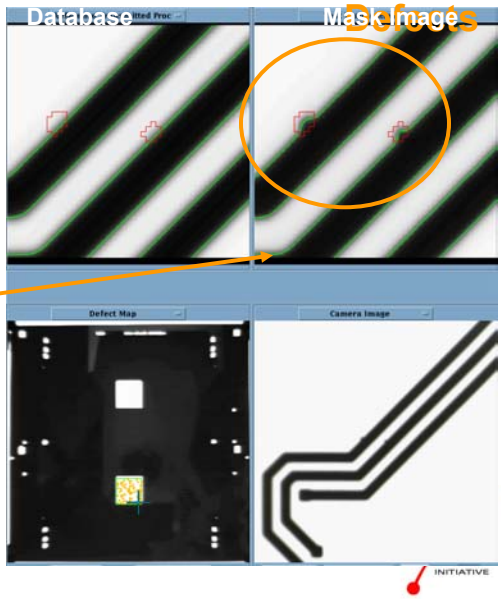
Process Step	Capability Indicator
Data fracture	Green
Data volume	Green
Write	Green
CD Measurement	Green
Defect Inspection	Green

KLA-Tencor TeraStar Inspection Conditions

- Two masks
  - 250nm DR mask
  - 130nm DR mask
- Inspection mode
  - Die-to-database
  - TeraStar standard algorithm – XPA
- Criteria: Monitor real and false defect counts

250nm DR Mask: Examples of Real Defects and No False

- Example of two contamination defects found on angle geometry edges.
- Note – no false defects at “elbows” as seen by previous generation inspection system.



KLA-Tencor Inspection Summary

The TeraStar F77 successfully inspects the 250nm DR and the 130nm X Architecture masks

- 250nm DR mask - Most defects were large contamination – due to no pellicle
- 130nm DR mask – Handled the OPC, few or no false defects

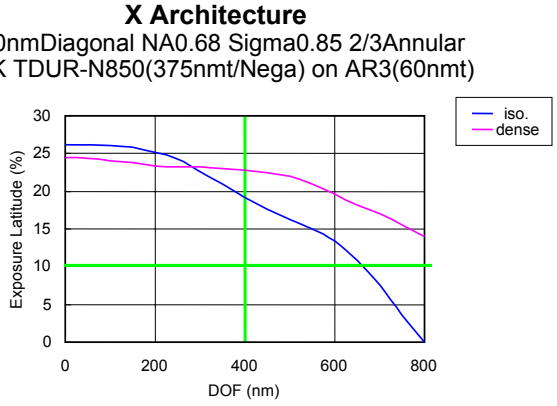
INITIATIVE

Nikon Experimental Conditions

- Exposed wafers using DuPont Photomasks 130nm generation masks with X Architecture data (metal 4 and 5 layers)
- Nikon used its NSR-S205C DUV scanner to expose the wafers
  - For 220nm X, NA0.68 s0.85 2/3 annular / negative resist
- Criteria:
  - Process window
  - ΔCD

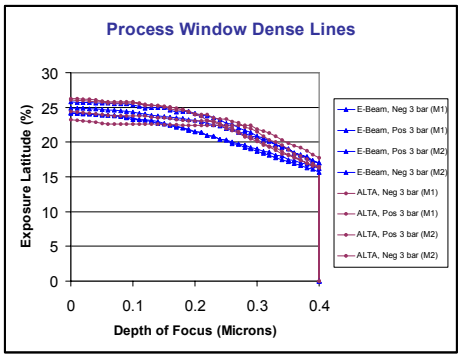
Process Window Results

- More than acceptable process latitude with X Architecture

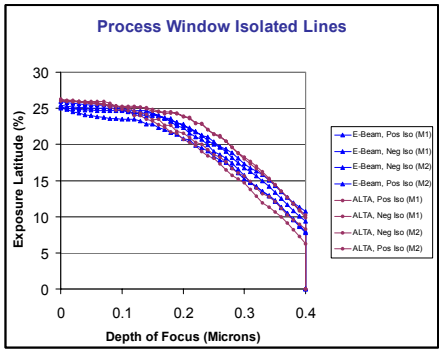


ASML X Wafers Results
E-Beam & ALTA comparison

X Technology - ebeam mask (Toshiba Machine EBM-3500)
X Technology - laser mask (Etec ALTA-3700)



L/S (3 Bar Structure - Dense Lines)

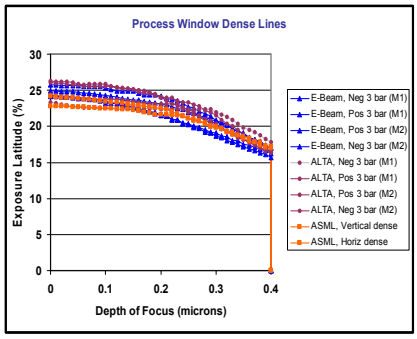


Isolated Lines

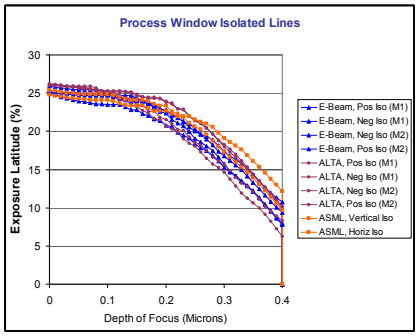
- No difference is observed comparing e-beam and laser generated X Technology masks

ASML X Wafers Results

Manhattan geometries
X Technology - ebeam mask (Toshiba Machine EBM-3500)
X Technology - laser mask (Etec ALTA-3700)



3 Bar Structure - Dense Lines



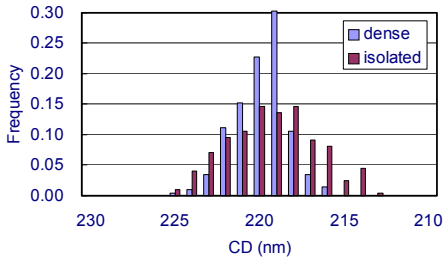
Isolated Lines

- No difference in manufacturability observed between Manhattan and X Technology non-critical dimension interconnect layers

CD Uniformity Results

- More than acceptable CD uniformity with X Architecture

X Architecture



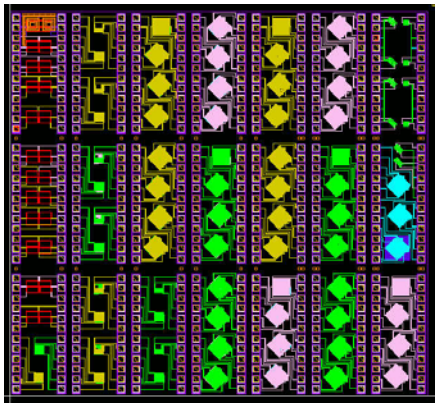
Dense 3s=4.6nm (2.1% of CD)
Iso. 3s=7.7nm (3.5% of CD)
Note: for wiring, up to +/-15% of CD is acceptable

Nikon's Conclusions:

- Using existing equipment and technology, the X Architecture is manufacturable today with sufficient process latitude and CD uniformity
- The X Architecture is a good additional test criteria for calibrating scanners for new technology nodes

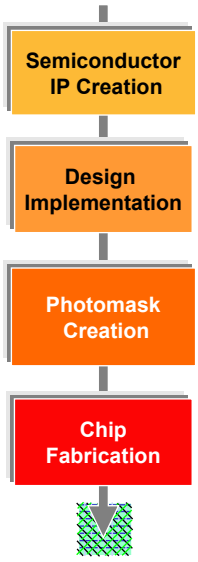
X Architecture 130nm Testchip Results

- 25mm^2 test chip with combs, serpentines, via chains at minimum pitch
  - all results OK on 59 dice!
  - We are pretty confident that we can manufacture X at minimum pitch in 130nm!
- Next test chip in 90nm soon



Jean-Pierre Schoellkopf, Central R&D, STMicroelectronics

The X Architecture: Roadmap for Design for Manufacturability



	First Availability	Expansion
Semiconductor IP Creation	✓ Confirm compatibility	
Design Implementation	✓ Physical Design ✓ Extraction	✓ Test chips ✓ Design methodology ✓ OPC
Photomask Creation	✓Mask fracture & writing ✓Mask CD metrology ✓Mask inspection	✓ Raster-based ✓ Vector-based ✓ Inspection
Chip Fabrication	✓ Design rules ✓ Lithography	✓ Design rules ✓ Lithography

Technology nodes

Summary

- X Initiative developed “soup to nuts” design-to-lithography flow to demonstrate manufacturability of diagonal routing
- Accumulated more than 30 members including leading semiconductor and photomask equipment suppliers, maskmakers, EDA companies, and semiconductor companies
- How can we leverage some of the lessons learned from this initiative to the broader DFM space?

IC Design and Manufacturing Process, ca 2005

Manufacturability Metrics and RET Tradeoffs For Physical Design and Layout

Luigi Capodieci, Ph.D.

Advanced Micro Devices, Sunnyvale, CA – USA

Abstract

In the development of technology nodes at 65 nm and below, manufacturability verification must be performed simultaneously with physical design, in order to guarantee sufficient process margins and economic viability. Three fundamental fabrication metrics are presented in this work, together with direct applications on physical design. Algorithms are predicated onto EDA software tools, which allow for the integration of design verification and process simulation capabilities. These novel methodologies enable the first practical implementation of a Design For Manufacturability flow.

1 Introduction

State-of-the-art design methodologies for VLSI electronic systems, in the last decade, have relied upon well-defined, fundamental levels of abstractions, from behavioral and functional specifications to system architecture definitions, from electronic block partitions to device design and, finally, from physical layouts to integrated semiconductor fabrication. Definitions of these design-level boundaries have been enabled and maintained by the continuous scaling rate in integrated circuit manufacturing, commonly referred to as "Moore's Law".

In the deep sub-100 nm domain (65, 45, 32 and 22 nm technology nodes), the entire design hierarchy is being challenged by non-linear effects due both to fabrication processes, and to nanoscale physical phenomena. As a direct consequence, exponential increase in semiconductor manufacturing costs for *traditionally* designed systems, severely threatens the economic feasibility of the entire electronic industry.

Interdisciplinary activities at various levels of design abstraction are converging into a proper R&D area, generally denoted as Design For Manufacturability. This paper will present the core set of these novel methodologies, and their application to the extraction of manufacturability metrics for Physical Design Layouts.

2 Design For Manufacturability

The broad field of Design For Manufacturability (DFM) addresses the fundamental VLSI engineering problem, defined in the previous section, by informing electronic design methodologies and tools with rigorous manufacturability knowledge and constraints. Its scope is not limited to the layout design and physical verification levels of the flow, but extends into the architectural and behavioral specification domains [1]. DFM consists of re-mapping the problem of **how to fabricate** a given electronic system into the problem of **how to specify and design** a *manufacturable* system.

Historically, during the development of the 180, 130 and 90 nm technology nodes, the natural (and more obvious) insertion points for DFM methodologies and techniques have been confined to the Physical Verification steps of the flow (*primary insertion point* as depicted in Figure 1). Resolution Enhancement Techniques and Optical (or, more appropriately, Process) Proximity Correction (RET/OPC) have become standard industrial practices in the development of advanced integrated devices. RET/OPC algorithms and software tools introduce local corrections to a given physical design so that economically viable fabrication processes can be employed to meet system specifications. These design modifications often conflict with original design specifications, and therefore RET/OPC applicability is severely limited, in the current traditional setting.

As a characteristic example, adoption of Alternating Phase Shift Mask (A-PSM) technology enables sub-resolution patterning, without expensive reduction in wavelength of the lithographic tools [2]. On the other hand, the use of A-PSM requires a layout to conform to specific geometric constrains. Satisfying such constrains often causes a global geometrical chain reaction, affecting the design all the way from cell synthesis to full-chip place-and-route. Electrical characteristics of individual devices (transistor's leakage currents and threshold voltages) are also directly affected. A vast class of electronic designs is currently not amenable to A-PSM, because implementation conflicts cannot be

eliminated without major (or complete) costly re-design. Furthermore no design automation methodology, nor tool, exists to evaluate the impact of changes in electrical behavior of the system (for instance critical paths, clock skew, etc.), which can propagate upstream in the chain of design abstractions, up to the architectural level.
This example highlights that *a posteriori* verification of manufacturability conditions is not an effective methodology for mapping electronic designs onto current and future fabrication processes. The appropriate engineering solution must consist of the definition of process-aware (or process-specific) design flows and the adoption of fundamentally new EDA methodologies, to be deployed during, and not after, physical design development.

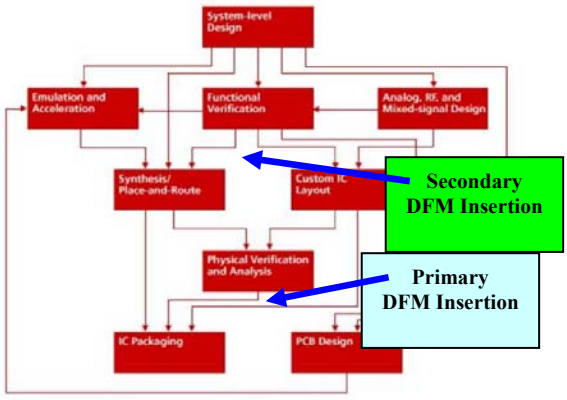


Figure 1: Flow Insertion Points for DFM.

In order to enable a more efficient *secondary insertion point* for DFM (as shown in Figure 1) quantitative manufacturability metrics must be available and integrated into automated layout synthesis and/or manual custom layout. Technical aspects of the implementation of such metrics will be presented in the following sections.

3 Functional Model for a DFM Engine

Existing RET/OPC tools provide integrated CAD layout and process simulation functionalities, which can be programmed to build virtually any DFM application. Within the scope of this study, we will describe the most general framework for Physical Design Layout applications, targeting 3 main objectives:

a. augmenting and guaranteeing layout manufacturability with respect to process latitude;
b. analyzing impact of CD variation and pattern fidelity on functional and electrical performance;
c. supporting the selection of a manufacturable Design Rule set.

Objective (c) is particularly critical in the sub-wavelength manufacturing domain. While *classical* Design Rules (DR's) for technology nodes above 130 nm were able to encapsulate most of process behavior and limitations, for 100 nm and below, non-linear effects cannot be described in crude terms of purely geometrical rules. Adoption of Model Based OPC (to replace simplistic Rule Based OPC) is the strongest indicator of the inability of DR's to accurately describe manufacturing processes. If a given fabrication process, in a specific technology, could be completely described with a set of DR's then, for each rule, an appropriate proximity correction action could also be defined. Instead, Model Based OPC (i.e. simulation based corrective actions) must be implemented to fix large classes of layout problems, not captured (nor predicted) by rules.
A further limitation of DR's is that they lack the ability to describe the entire chain of cause-and-effect from imaging to patterning distortions onto single device electrical characteristics and finally onto full chip functionalities. DFM applications are not aimed towards replacing DR's, but rather towards extending them, by integrating non-linear effects at various length scales.

The fundamental building block for a DFM engine is represented in Figure 2.

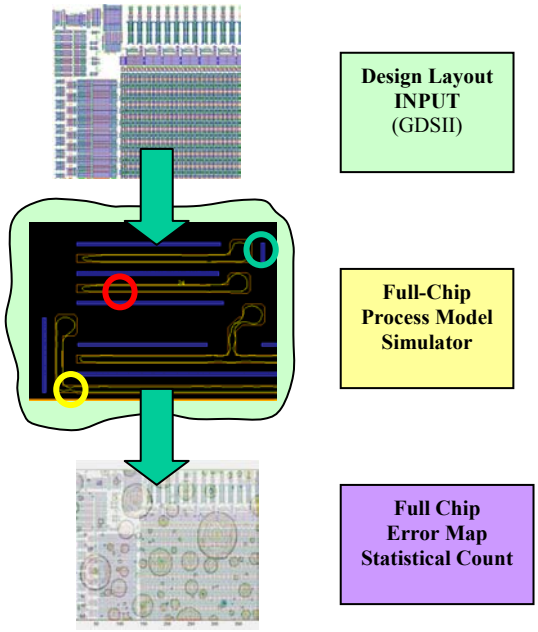


Figure 2: DFM Functional Building Block

It consists of an Image and Process Model Simulator integrated within a physical verification (DRC/LVS) software environment, built on top of a hierarchical design database. A suitable programming (or scripting) language (such as Perl, Tcl, etc.) allows for the development and implementation of specific DFM applications. Computational capabilities of the engine must include hierarchical (and flat) processing of polygonal data, fast process modeling and statistical analysis, including error counting and area (X,Y) mapping.
The ability to extract process **error statistics** at the full chip level, including exact locations for such errors is the true leverage of the DFM engine.
Extraction and categorization of process errors can be based either on edge placement (CD) or patterning and image properties (contrast, image log-slope). Both types of errors can be also evaluated with respect to process latitude, as it will be shown in the following section.

Two general classes of DFM applications can be built using this functional model.
A DFM *Design Analysis Application* iterates over an input set of design layout variants, given a specified process model, and computes a ranking of these design variants based on their respective distribution of process errors.
Analogously a DFM *Process Analysis Application* iterates over a set of process models, given a specified test design layout, and computes a ranking of the input process models again based on the different error distributions generated by simulating each process onto the given test design.

4 Manufacturability Metrics

The availability of the functional DFM core, as defined previously, allows for the implementation of novel applications to be inserted during the layout design phase of the flow. This section will describe 3 fundamental manufacturability metrics and corresponding algorithmic implementations. As state of the art in DFM continues to evolve, similar applications will become standard part of design flows.

4.1 Design Rule Look-Ahead

The early phases of a technology node development (e.g. 90 nm, 65 nm, etc.) consist mainly of the definition of a suitable set of Design Rules. The starting point for this set is typically a "shrink" version of a previous rule set generation. Since no physical fabrication process has actually been implemented at this stage of development, validation of the manufacturability for each individual rule can only be based on extrapolations of previous empirical data and/or simulated predictions of future processes.
Except for individual (company specific) best practices, no formal methodology is applied at this stage to generate a **quantitative** assessment of the manufacturability of each proposed design rule. As soon as the initial DR's set is released and physical designs are started, a particularly "fabrication critical" rule might therefore be extensively used and become embedded into cell libraries and macros. The discovery of such criticality will occur much later in the flow, for instance during a test chip revision, with severe economic and technological penalties.
The following *Design-Rule-Look-ahead* (DRL) algorithm provides a formal methodology for DR's evaluation. The procedure is based on the fundamental observation that design rules do not exist in abstract, but they are rather "abstractions" (or generalizations) of actual geometrical conditions occurring in real layouts.

DRL Algorithm I [Ranking and Revision]

Given a set of test layouts (TL_j) and a set of design rules (DR_k):

(1) Perform Design Rule Checks on TL_j
(2) Compute DR_k histogram (frequency count)
(3) For each DR_k
    a. Perform process simulation
    b. Extract process metrics (CD errors)
    c. Compute Manufacturability Ranking
(4) Sort DR's according to ranking

Typical results of the first two steps of the algorithm are shown in Figure 3. In this specific case 7 designs have been used to extract rule checks and their respective frequency counts (represented as histograms) out of a (synthetic) set of 10 rules.

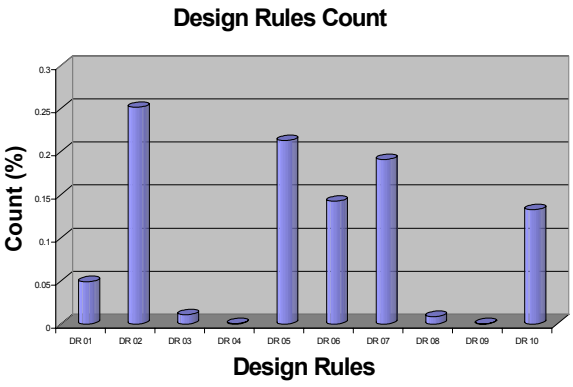


Figure 3: DR's frequencies

It can be immediately observed that DR_4 and DR_9 have a very low frequency compared to most of the other rules in the set.
As design rule occurrences are checked against the test layouts, corresponding DR_k locations are also identified. This allows process simulation to be performed at each of these locations. It is very important to notice that, in this methodology, a given design rule can be simulated several times, within the different geometrical contexts it appears in. This allows for an increased level of confidence in the modeled fabrication robustness, as a statistical model is implicit in the algorithm.
The Manufacturability Ranking (MR) value in step 3.c can be computed as a function of process errors returned by the simulation and analysis steps (3a, 3.b). Average CD error, for instance, can be typically used.
The final DR's sorting order is obtained by multiplying each DR_k frequency with its corresponding MR. This sorting constitutes also a quantitative *revision order*, which can be used to determine the cost of changing or eliminating a given design rule with poor manufacturability.
In the case depicted in Figure 3, DR_4 results to be the first candidate for rule revision, because its large average CD error (fabrication critical) and its extreme low occurrence in the input test layouts (design non-critical).

While the DRL I algorithm allows to quantitatively compare sets of design rules among themselves, thus determining what rule (or rules) degrade manufacturability, the procedure does not provide any indication about what "improvements" could be introduced into a design rule set in order to make it more manufacturable. An algorithmic variant to address this issue will be presented in the following section.

4.2 Design-Rule Analysis (Forbidden Pitches)

Any manufacturability metric is obviously not simply a function of a specific DR's set, but also of the chosen fabrication process [3]. This intrinsic dependency stands at the root of a potentially catastrophic flaw affecting all the *traditional* design-to-manufacturing flows, currently adopted in the electronic industry. Design rules tend to be defined at the very beginning of technology node development, when only vague approximations exist relative to the actual production processes of choice. The 65 nm technology node is emblematic. First lithography tools will be shipped by the end of 2003, almost two quarters later than the definition of the actual design rules. But while it is an accepted industrial practice to revise OPC treatments every time a change in the fabrication process is introduced, DR's are very seldom modified.

In order to support DR's revision based on process changes, a variant of the DRL algorithm is proposed. The procedure allows for the characterization of a **Manufacturability Matrix** MM(k,j), where each row corresponds to a given rule DR_k in a DR's set and each column correspond to a fabrication process alternative RETj. MM(k,j) values can be any of the previously defined process error metrics, such as average CD error, image contrast, process latitude, etc.
The algorithm will be applied to the analysis of fabrication sensitivity for a DR's set, with respect to the well-known phenomenon of "Forbidden Pitches" [4].
As shown in Figure 4, Critical Dimension (for gate features in this specific case) displays a strong variability, with respect to its nominal value, functionally dependent on feature pitch.

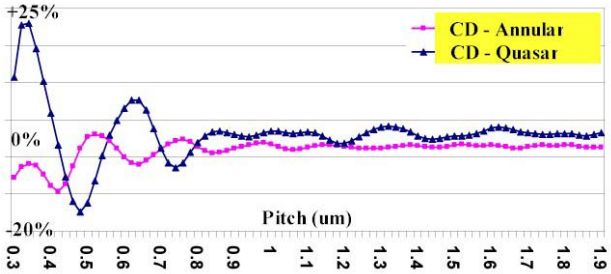


Figure 4: Forbidden Pitches for Annular and Quasar

Furthermore such dependency also varies with different patterning processes as it can be observed for Annular Illumination and Quasar Illumination, in Figure 4. This well known lithographic effect results in some CD's not to be correctable with any type of OPC, because of the extreme oscillation in their magnitude. Analogous pitch effects exist for line ends, corner rounding and in general any type of pattern fidelity measure. It can be concluded that in the sub-resolution domain these types of non-linearity cause manufacturability of layouts and design rules to be completely tied to a chosen fabrication process [5].
The first two steps of the DRL II algorithm are the same as in the original DRL I. Design rule checks are performed on test layouts in order to extract a frequency count for each rule, thus quantifying its relevance for the given technology.
A Manufacturability Matrix is then computed by modifying the inner loop and further iterating over a given set of RET processes. Manufacturability of every layout is carried out first allowing unrestricted pitches in test layouts, then selectively removing the RET specific pitches.

Values MM(k,j) are finally computed as the ratio of a given process error metric with forbidden pitches removed, divided by the same metric simulated with test layouts *including* all pitches.

DRL Algorithm II [Forbidden Pitches Analysis]

Given a set of test layouts (TL_j) and a set of design rules (DR_k):

(1) Perform Design Rule Checks on TL_j
(2) Compute DR_k histogram (frequency count)
(3)
(4) For each RET_j
    a. Evaluate set of forbidden pitches (FP_j)

(5) For each DR_k, for each RET_j
    a. Perform process simulation (all pitches)
    b. Extract process metrics (CDE_1)
    c. Perform process simulation (forbidden pitches FP_j removed)
    d. Extract process metric (CDE_2)
    e. Compute Manufacturability Ranking as ratio CDE_2/CDE_1

(6) Sort DR's according to ranking in MM(k,j)

Design rule negotiations among product design groups, process integration groups and fabrication groups can directly apply this methodology for the quantification of the trade-off between the introduction of DR's restrictions and their immediate benefit in terms of manufacturability. For instance, considering process data given in Figure 4, the choice of Annular illumination (RET_1) which allows for more general design rules (no forbidden pitches) might be quantitatively compared and replaced by the choice of Quasar illumination (RET_2), which requires more stringent design rules (2 intervals of forbidden pitches), but has an overall superior manufacturability ranking (reduced process CD error).

4.3 Layout Process Signature

As a further generalization of the methodologies presented in the previous section it is possible to define a *Layout Process Signature* (LPS) metric, as a full chip X,Y distribution of process errors.
Any software implementation of the DFM engine described in Section 3 (Figure 2) can extract not only a statistical count for any given process metric, but also its magnitude at any given point over the full layout.
Figure 5.1 shows an example of LPS. A statistical count of line-end pullback errors has been extracted for a given lithographic process at two values of focus (best focus and best focus -5%). CD error values are indicated on the X-axis, from 0 nm (right of the graph) to negative 60 nm line-end pullback (left of the graph). The Y-axis reports the total count of line-end errors in a given range. It can be observed that when the process is simulated out of focus, the distribution of line-end errors shifts towards an overall reduction in layout manufacturability.
An analogous histogram of line-end pullback errors is shown in Figure 5.2. In this case magnitudes of the process metric are plotted in the 2-dimensional space of lithographic focus and exposure dose. Finally Figure 5.3 correlates the statistical count with the X,Y map over the full layout.

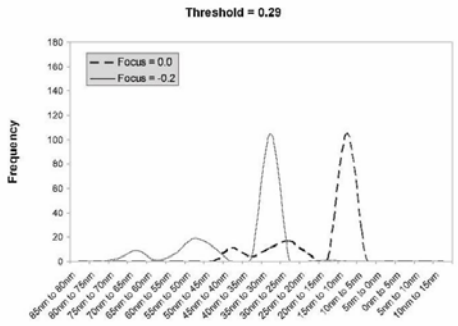


Figure 5.1: Layout Process Signature (Focus)

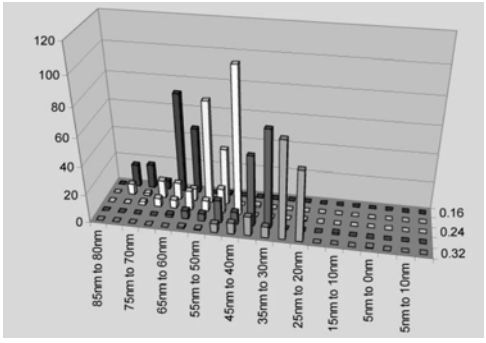


Figure 5.2: Layout Process Signature (Focus,Dose)

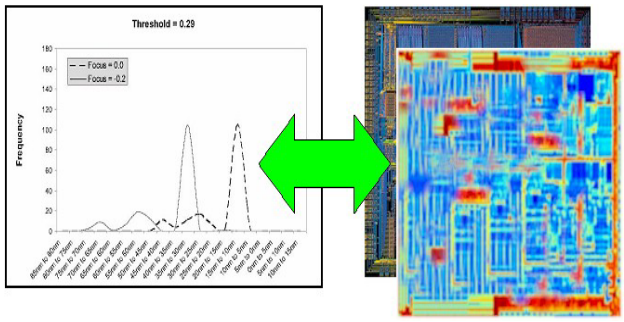


Figure 5.3: Layout Process Signature (Full-Chip Map)

5 Conclusions

Physical integration goals as defined by the International Technology Roadmap for Semiconductors, for 65 nm, 45 nm and below require manufacturability verification of Physical Layouts to supplement (at design time) traditional Design Rule Checks.
This paper has described the fundamental structure of a software building block to be used in such manufacturability applications. Three novel algorithms have also been presented for the extraction of quantifiable manufacturability metrics, either from DR's sets or from full chip layouts.
Challenges in the deployment of these tools in the design flow, lie in the intrinsic interdisciplinary nature of DFM tasks. Collaboration between chip designers and fabrication engineers will be an essential element of successfully manufactured designs. The potential reward for overcoming "cultural" engineering barriers will be provided by the viability of DFM, not only as a technical tool, but above all, as an economic alternative to costly (and continuously delayed) Next Generation Lithography tools.

References

[1] Luigi Capodieci, Juan Andres Torres, Robert Socha, et. al. **Effects of Advanced Illumination Schemes on Design Manufacturability and Interactions with Optical Proximity Corrections** [SPIE, Challenges in Process Integration and Device Technology, September 2001, Santa Clara]

[2] Lars Liemann, Jennifer Lund, Fook-Luen Heng, Ioana Graur, **Enabling Alternating Phase Shifted Mask designs for a Full Logic Gate Level** [SPIE, JM3 April 2002]

[3] Frank M. Schellenberg and Luigi Capodieci. **Impact of RET on Physical Layouts** [Proc. ISPD, 2001]

[4] Robert Socha, Mircea Dusa, Luigi Capodieci, Jo Finders, Fung Chen, Donis Flagello, Kevin Cummings. **Forbidden Pitches for 130 nm Lithography and Below** [SPIE, Microlithography 2000]

[5] L. W. Liebmann, G. A. Northrop, J. Culp, M. A. Lavin, IBM Corp. **Layout optimization at the pinnacle of optical lithography** [SPIE, Design and Process Integration for Electronic Manufacturing 2003, Proc.5042-01]

Design Rules for Real Patterns

Alexander Starikov*
Intel Corp., CA Technology & Manufacturing, Santa Clara, CA 95052

Abstract
The purpose of this paper is to name and discuss some of the most basic issues in today's optical microlithography, promoting industry consensus and, specifically, definition of the industry standard Patterning Design Rules (PDR), that is, Design Rules for "patterns as built". Dimensional metrology for seamless PDR compliance is discussed, highlighting the issues at design hand-off from CAD to pre-tapeout validation and from validation in software to validation on wafers. Some of the gaps in validating the accuracy of patterning models used at design and pre-TO validation are identified, along with a proposal of how to bridge them. It is expected that, as our industry migrates from CAD based DR definitions to PDRs that reflect realities of microlithography at low k_1, a complete self-consistent dimensional metrology infrastructure [1] will eventually be built. This is what will hold together compliance at design and on wafers, with model based WYSIWYG (what you see is what you get) design and process integration environment of the future.

1 Introduction

The need to decrease critical dimensions (CD) of the most advanced microelectronics devices increasingly challenges our views of the practical limits in optical microlithography. Proliferation of "sub-wavelength" optical lithography has obsoleted WYSIWYG paradigm in layout-to-silicon feature replication. Reticle-based image/pattern enhancement techniques (RET) enabled reduction of the largest systematic intra-field errors. Such changes to "as designed" pattern (on reticles) improved device performance and yield, extending processing margins achievable with optical lithography. Improved size tolerance, in turn, made it possible to print smaller devices. Yet, our industry's slow acceptance of the new design, patterning, and processing methodology is a factor in delaying introduction of the significant potential technology gains. Slow acceptance of aggressive RET is, to be sure, largely a reaction of the design, integration and test/yield communities' apprehension of the risks inherent in rapid proliferation of any new technology.

RETs, especially computer based aggressive modification of design can take such risks to an extreme. Numerous accounts of patterning failures and marginalities are found in the recent literature. Even a cursory review of the recent publications at SPIE Microlithography and Photomask meetings highlights many RET applications that should be classified as something that is between sophisticated frills and demonstrably wasteful changes of the design DB. ISQED 2002 Panelists reflected on the gap between those who strongly advocate an EDA-lead changeover of design and manufacturing practices, and business models, vs. the practitioners of microelectronics manufacturing who take proposals to extend the envelope of our POR (process of record) with a grain of salt. When asked, they may comment on the rate of change being too fast, insisting on elaborate risk assessments before they are convinced. Recent industry survey of the systematic non-particle functional bin losses [2] identified a trend for reduced technology-limited yield in CMOS, as it evolved from CD ~ 500nm to 180nm. This suggests that multiple accounts of the isolated cases of failure due to weak links in design and integration of patterning may be a part of a much broader trend, implicating a growing deficiency of our design and integration practices, and PDR validation.

We may well be violating every DR there is, but we do not know that – not on 100% of such instances. The reason is simple: we have long run out of DRs that are applicable realistic patterning and of the capability to sufficiently validate them. Today's validation – the first time a patterning issue is noticed – often is device yield.

This paper reviews basic definitions in dimensional metrology and how we use dimensional metrology (and functional test based data) to design OPC or other RETs. It names a small set of generic Patterning Design Rules, with dimensional metrology capabilities required support them, and goes through in-depth review of one of them.

Dimensional metrology infrastructure of PDR validation (missing today), design/function-based metrology and appropriate metrics of compliance can support the model-based WYSIWYG design and integration environment. This is a solid foundation for "deep sub-wavelength" patterning and efficient microelectronics manufacture.

2 Digital ICs, Lithography and Design Rules

Digital ICs may be the most commonly manufactured microelectronic products of today. Microlithographic patterning of ICs, on the other hand, retained many similarities with analog (objects and their recorded images are of continuous tone, with infinitesimal variations of tone, size and shape) microphotography of 1950s. The built-in expectations of product requirements,

* alexander.starikov@intel.com

materials properties, processing and manufacturing environment are hundreds years old.

Required to be either fully ON or OFF binary devices are relatively forgiving of manufacturing tolerances. This property enables an efficient mass-production of high yield/performance products that are both reliable and inexpensive. The tightest tolerance required is in matched transistor pairs, preventing the circuit lock-up and/or inappropriate change of state. This requirement is served by symmetry-based (translation, mirror, point) cell replication in IC design, instancing. Efficient IC manufacturing is only possible when every its aspect maintains symmetries required by devices.

If an IC were just a massive array of near-perfect transistors, it would be easier to make. But, to provide complex functions, IC must generate its own pulses to many registers' triggers, amplify and invert signal etc. Transistors that serve such functions are much less numerous, but, because they are designed to match multiples of other devices, their being built to size is also critical. The further away one device is from another, the less matched they may be without any losses. All devices in a die are sized and toleranced according to their design/function. Sizing requirements, reflected in Pattern Design Rules, repeat on the next die – from matched pairs to circuit neighbors, small vs. large etc.

Imaging with optics, unlike pattering with many other means, is a nearly perfect match for the need: imaging properties change slowly as the function of distance, that is to say, locally translation invariant. Unfortunately, all other symmetries of optical imaging systems are broken by aberrations, illumination and polarization effects. In-plane image vibration during exposure on scanners is just one detractor in the loss of isotropic (X same as Y) imaging. In order to make devices that rely on more than local translation invariance, all optical lithography systems are designed, built and maintained as free of errors as possible. Optical microlithography enables manufacture of devices that are near- perfectly matched over small distances and very well matched over the whole field. Any device features on the reticle, either isolated or in uniform arrays with period >λ/2NA are replicated with near-perfect uniformity (low variance).

Simultaneously correct sizing of both small and large features, on the other hand, is somewhat difficult due to optical diffraction and interference effects. Significant variation occurs for features in varying proximity of each other and at the end of a uniform array >λ/NA distance is required (wasted) to either prevent or compensate such changes in imaging. To sum up: optical lithography may be the perfect match for the manufacture of uniformly sized and spaced devices. At its incoherent cut-off limit, it allows printing close to one device per (λ/2NA)*2 area on the wafer. Optical microlithography at 248nm (in air/no frequency doubling) can pattern devices at half-pitch of 64nm, with device CD being much smaller.

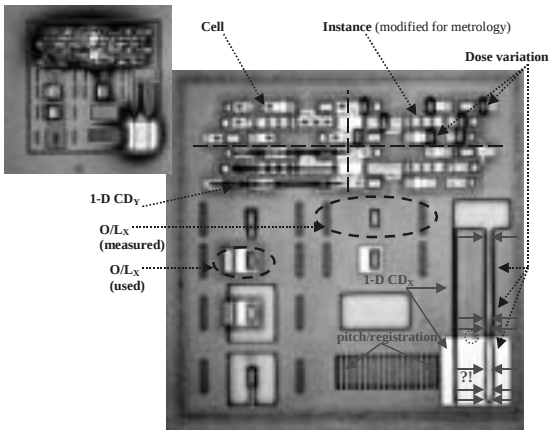


Figure 1. Metrology Test Site in bright field with (insert) NA=0.2, λ=546+/-5nm and NA=0.9, white light.

Printing on the previously patterned substrates, illustrated in Fig. 1 [1], may degrade device replication by optical lithography or it may retain it, if at a price. It may also be quite forgiving of error (self-compensating), precluding accrual of sizing errors in device patterns. The outcome is a strong function of layouts and patterning processes.

Prevailing design and manufacturing practices of today severely limit the ultimately achievable device tolerances by requiring that all features – any shape, size, polarity, orientation, regardless of proximity – be printed "to size". Although much better patterning and device performance can be achieved through design and process integration, limitations of this kind are not obvious at design, let alone what it will cost to manufacture "as designed" or what gains we lost by failing to explore alternatives.

To better account for actual device needs, reducing waste inherent in indiscriminate requirements for tight sizing, as well as losses of performance and yield wasting resources of the superfluous rather than driving tighter tolerance in something critical, we need ***design- and function-aware Patterning Design Rules***. These Rules must be based on the designers' knowledge of the design/device function. They must also account for patterning processes, with our ability to control process and assure compliance. It is of essence to all of us to understand them, and all aspects of IC manufacturing embodied in them, from basic physics to trade-offs in performance and yield, manufacturability, costs etc. Improving device tolerances assigned to such PDRs, as well as improving PDRs themselves, is the path that leads to the emergence of the superior IC design and manufacturing methodologies.

3 Design Rules and Dimensional Metrology

Design Rules is a normative document that specifies the key dimensions of a product that may be manufactured, while staying within known capabilities of a generation of lithography and patterning equipment, supported by everything required to produce an IC – from design and design validation methods and tools, to materials and processing, metrology and process control, failure analysis etc. ***Design Rules*** is our collective acceptance of what are (rather, used to be) the ***most essential properties*** **of a design. DRs may look less than glamorous, but when each DR requirement is obeyed at design and compliance is validated, gross patterning deficiencies are unlikely.** A product may be designed, design can be validated and chips be manufactured - with product properties predictable and close to the expectations. As in any well-run ***business process***, value ranges expected for each single Design Rule are reviewed in the light of all recent learning. New value or tighter range of a Design Rule values may be needed to preclude marginal designs that lead to yield loss. Design Rule is relaxed when it is established – with certainty – that the losses of anything tangible do not occur when it is exceeded, especially when relaxing this Rule leads to quantifiable tangible gains or to the recognizable non-tangible gains.

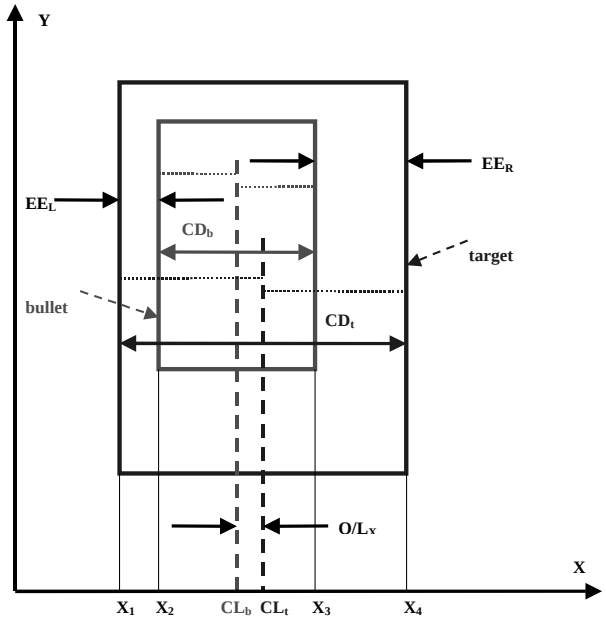


Figure 2. Linewidth (CD), centerline (CL), overlay (O/L) and edge-to-edge overlay (EE O/L).

To define Design Rules, Patterning Design Rules and their counterparts in the dimensional metrology, we use broadly applicable convention [1, 3-5] illustrated in Figure 2. We use the name feature to denote the simplest element in one layer of the design of a thin film device. Feature linewidth (space width, width, critical dimension or CD) in layer t (target, reference) and level b (bullet, resist, current) is denoted CD_t and CD_b, respectively. If the edge coordinates are denoted as X_1, X_2, X_3 and X_4, then

$$\mathbf{CD_t = |X_2 - X_3|}$$

and

$$\mathbf{CD_b = |X_1 - X_4|.}$$

Likewise, the centerlines of the target and bullet featuresare denoted CL_t and CL_b,

$$\mathbf{CL_t = (X_2 + X_3)/2}$$

and

$$\mathbf{L_b = (X_1 + X_4)/2.}$$

Since microlithography involves patterning very many features in each layer, a set of all centerline coordinates, registration, is of interest when it pertains to in-plane distance between a centerline of a feature in one layer and a feature in another layer, used as reference. This distance is centerline overlay (overlay, O/L). Referring to Fig. 2, overlay of two features, whose centerlines are CL_b and CL_t, is defined using the following convention:

$$\mathbf{O/L = CL_b - CL_t.}$$

PDRs require that certain widths in certain features be within stated allowed ranges, manufactured to size within stated tolerances. Accurate account for EE O/L is required for effective design and integration of IC patterning. Failures to comply with EE O/L PDRs in back end of the line (BEOL, metallization) are known to lead to poor yield and reliability. Metrology of device O/L and EE O/L illustrated Fig. 1 was developed [6]. It is on SIA and SEMATECH Roadmaps since 1994. Yet, in today's practice, device level O/L, especially EE O/L, is seldom measured [7]. Designers' assumption that dimensional metrology is readily available and used to validate PDRs, as used at design and pre-TO validation, is violated.

Today's design environment still has, or appears to have, a complete account of all dimensions of all features in all layers. Design and integration universally recognize that "over-design" leads to lower profits, that "under-design" is a risk that may result in loss of product yield and/or reliability. We know that the best-designed product does not fail at "a single weak link" – it does not have one. When stressed to fail, the best-designed product will fail everywhere at once, reaching the highest stress limit. To achieve that, our design practices shrewdly prescribe where the tightest DRs may be used. Designs are thoroughly evaluated to make sure that if the tightest DR is used in just a few layouts of a product, these be re-designed and relaxed or the rest of the product also take full advantage that stems from using that tightest DR. Yet, our design and design validation environment is largely oblivious of the diverse environments in imaging, image recording and image transfer for one layer on a virgin substrate, let alone on a previously patterned and processed wafer. Assumption that the design is uniformly

replicated on the wafer and that the product is compliant with the design, within tolerance, is no longer true.

Low k_1 optical microlithography brought in strong new interactions of imaging and patterning processes with design. This patterning environment is deluged with unintended variations of both CD and image placement: image placement of dissimilar features is not the same, asymmetric aberrations and illuminator errors move image laterally through focus, errors of OPC or assist features, of phase, transmission, size or placement of phase shifters result in variations of both CD and image placement. What may be the least recognized is that potential improvements of pattern tolerances and process windows, the goal of RET, are only possible to the extent that their integration may reduce the total of all patterning errors. The success or failure of microlithography is not dependent on improved image "resolution" or "fidelity". That is predicated by sustained reduction of CD and EE O/L tolerances in device patterning, as required by PDR, on the condition that it is the most cost-effective means to improve product yield and performance.

It is of essence to all of us in microelectronics industry to review and define a set of generic Patterning Design Rules, to make certain that they are supported by a complete set of dimensional metrology tools uniformly applicable at every step of model-based evaluation of design and validation of compliance – from database to wafers:

- models of patterning whose accuracy is known for all PDR cases and easily validated in layouts;
- function-aware metrology for design assessment and pre-TO compliance validation;
- function-aware metrology for post-TO and in-production PDR compliance validation;

Referring to critical layers in CMOS and bipolar ICs, here is a short list of generic PDRs:

One-layer Design Rules

- **linewidth** or **space width** (in 1-D and 2-D features), with MAX and MIN**;** between opposing edges in one layer;
- **area** of 2-D islands or trenches, with **perimeter** and **aspect ratio**; possibly, radius of corner rounding.

Two-layer Design Rules

- **Edge-to-Edge overlay**, distance between two edges (same or opposite, with MAX or MIN) in layer pairs;
- **Area of overlap** for 2-D features in layer pairs (area that is common to both layers).

4 PDR case study: 2-D feature (emitter or contact)

This is an illustration, and discussion, about what is and is not important in patterning ICs. It highlights the key aspects of model-assisted design environment in the case of the simplest serif-based OPC [8]. This OPC is commonly used at contact, via and implant layers, as well as emitter of bipolar devices. This old example has no proprietary aspects and is exceedingly simple, making it easy to observe and discuss the issues.

Problem statement:

Rectangular and substantially isolated emitter openings fail to print to size across all emitter sizes allowed by the Groundrules. When exposure is such that large rectangles are "on target", rectangles with smaller side <λ/NA are undersized; percentage area loss is the worst for squares. Since the sidewall narrowing technology is used to shrink emitter area, this poor size tolerance is applied to a much smaller effective emitter area. Bipolar transistor design and PDRs of relevance are illustrated in Figure 3; a great deal of salient detail related to these generic DRs is available in the public domain [9]:

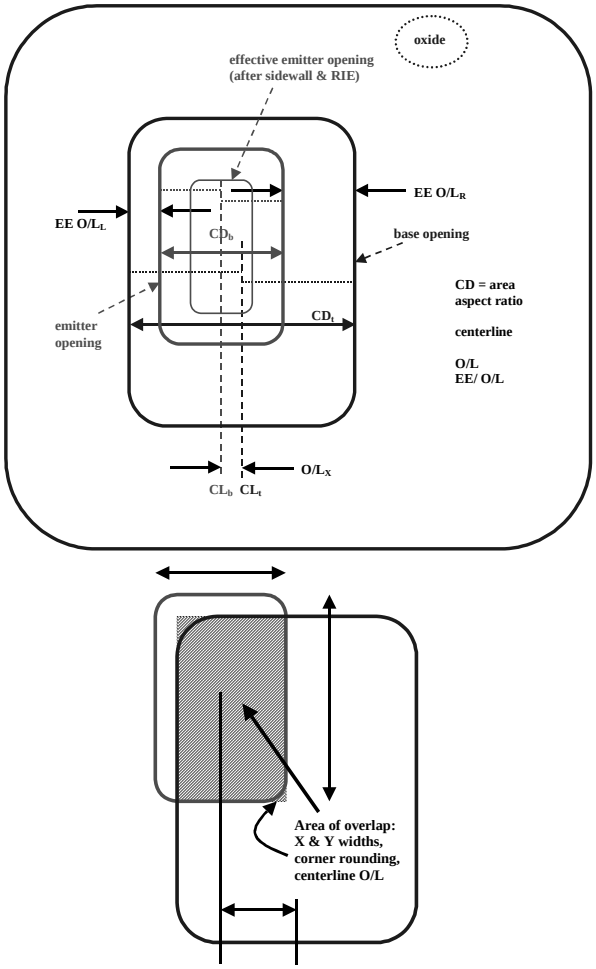


Figure 3. PDRs for emitter opening in bipolar transistors. Design is Fig. 2. Printed resist patterns shown in Fig. 1.

Design #1: non-aggressive serif

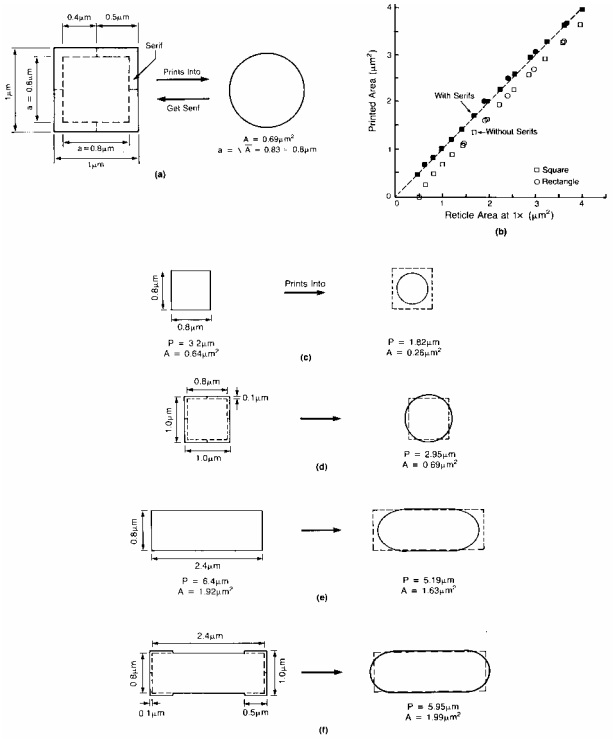


Figure 4. Design and properties of non-aggressive serif.

Although designers **want** emitter printed as a rectangle with ***aspect ratio*** ~1.5 (for performance transistor be fast, driving as high a current as allowed by current density), they **need** to get all emitters built close to ***designed size***, with tight control of ***conductance*** for **all sizes** and **types**.

Quest for best RET (introduction to dialog):
What is "optimal design"? What is "designer intent"? What are design objectives? How should the many conflicting requirements be co-optimized?

A square is the **least complex shape** on the design grid. A model of lithographic printing shows that a single size square serif gives some improvement of areas printed for rectangles across a broad range of sizes and aspect ratios, as needed. This is a two-parameter design (serif size and extension/jog). It involves one or two experimental data points and a "back of the envelope" design procedure and model-based assessment of variable of area, perimeter and aspect ratio correction at "aggressiveness"; Figs. 4-5.

Design must be validated - the means may not exist. Which design is easier to manufacture on the reticle? **Model-based evaluation** of impact, such as influence of mask sizing error on sizing in print and on defect printability: the more aggressive serifs, the stronger impact.

Design #2: aggressive serif

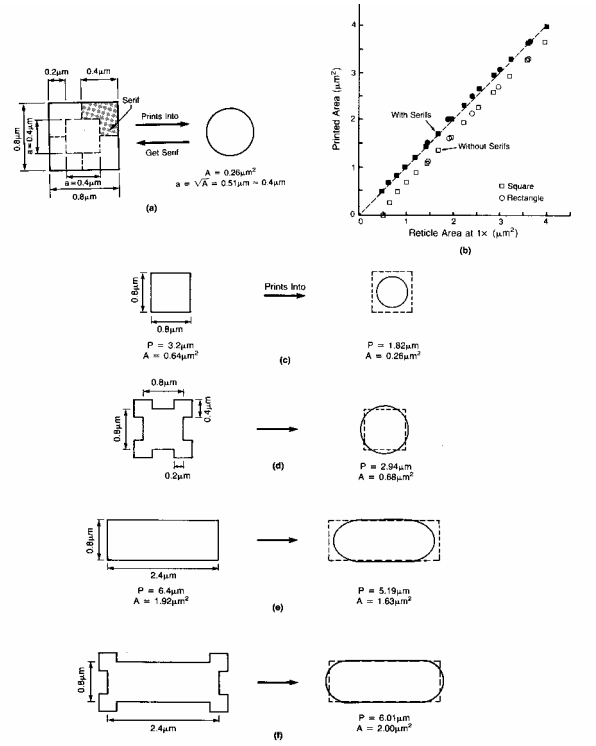


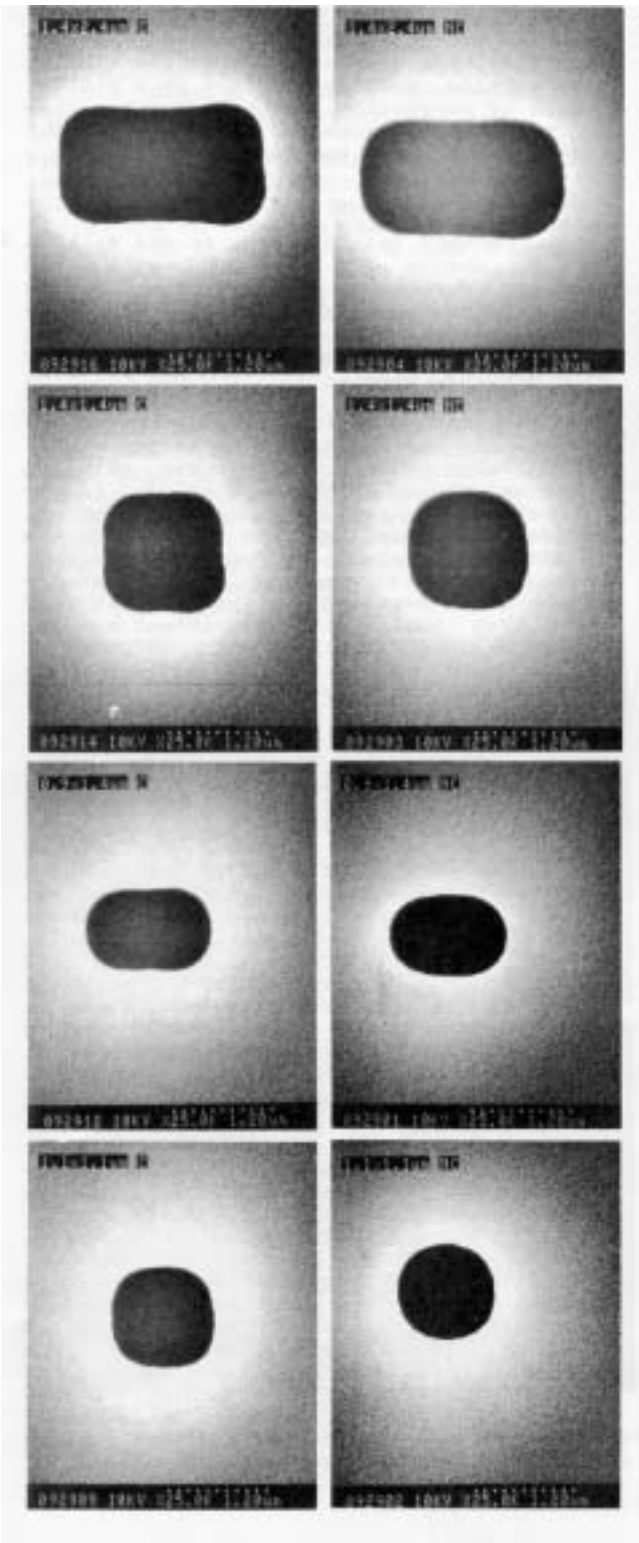
Figure 5. Design and properties of aggressive serif.

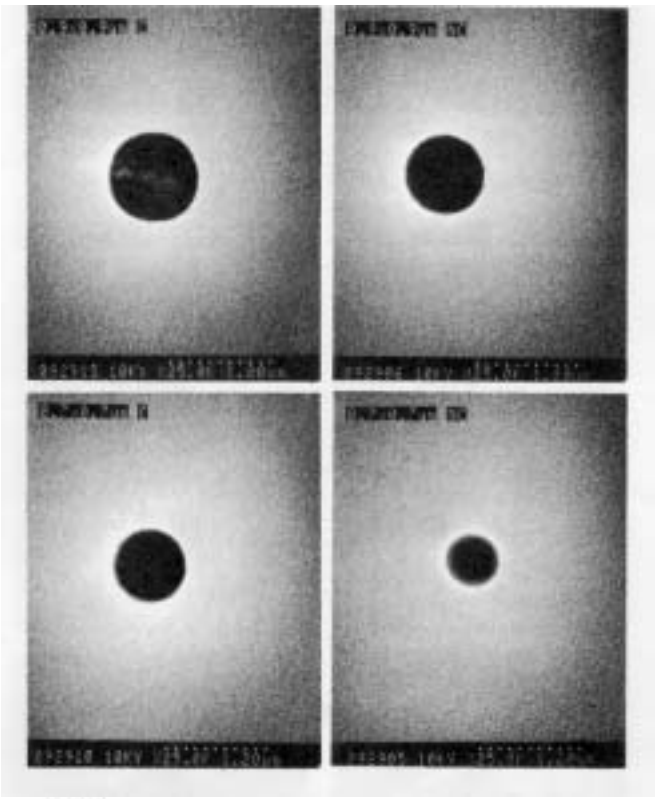
Which design gives the least error of *size to target*?
Which design gives the least *variance for* each size?
Both aggressive and non-aggressive design result in print area, perimeter and aspect ratio brought up to near target for all features. **But…** aggressive design has poor process windows, driving high variance (Fig. 8).

Which is a better design?
What are the metrics?
Is the model accurate?
How accurate is metrology?

<u>Need metrology of area, perimeter and aspect ratio.</u>
This is the DR at stake here – better control of this parameter, **area**, indicates success. Build and print reticles (POR). Measure area in print, Figs. 7, 9-10. Use estimated error to improve the design. Correct inaccuracy during model use in RETs (and model-based pre-tape-out validation). Dimensional metrology (of area in print) and electrical test data (cross-checked and self-consistent) - not the model - are the inputs to model-based design procedure (rule, algorithm).

<u>Pre-T/O design validation requires accurate models.</u>





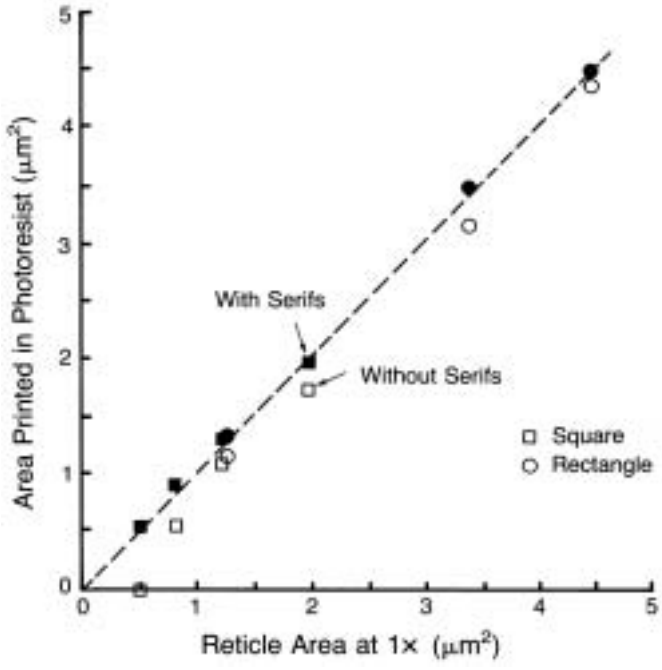


Figure 7. SEM based metrology of area in print.

Post-tape-out design validation requires metrology.
Metrology of area done with a specially modified SEM [6] confirms (Fig. 7) that emitters with serif corrections print closer to target area and aspect ratio across a wide range of sizes and aspect ratios. This was laborious, not all PDRs were checked - poor capability. More is needed: show better process margin, performance and reliability.

Dimensional metrology of emitter size in Fig. 7 and, more to the point, **e-test of effective area** in Figs. 9-10 confirm that emitter openings with serifs print closer to target and (even) with improved distributions.

This is just begins to validate the utility and PDR compliance for *isolated small 2-D features*…

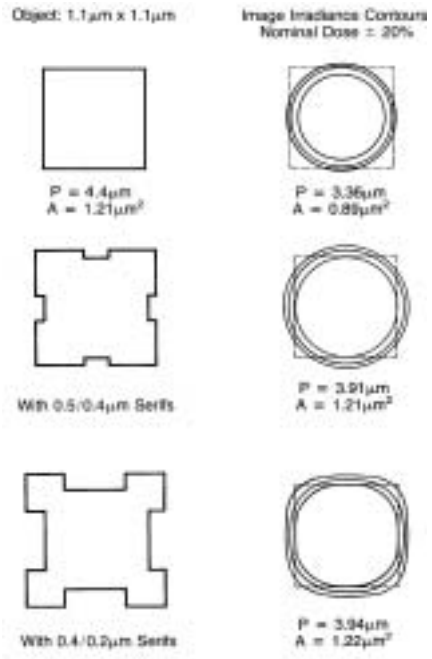


Figure 8. Engineering trade-offs in aggressive serifs.

5 Summary and Recommendations

Standard definitions of PDRs do not exist and, as the result, dimensional metrology capabilities to support them as required in low k_1 patterning do not exist, either.

CD-SEM based metrology of area in 2-D features with aspect ratio near 1:1 has been recently introduced. Their perimeter (PDR for storage node) still is not measured. Metrology of the smallest linewidths and space widths (the smallest distance edge-to-edge in opposing edges) in layouts is still manual and tedious. Validation of device O/L, EE O/L and overlap is handicapped by the lack of dimensional metrology infrastructure [1, 3, 5, 10, 11].

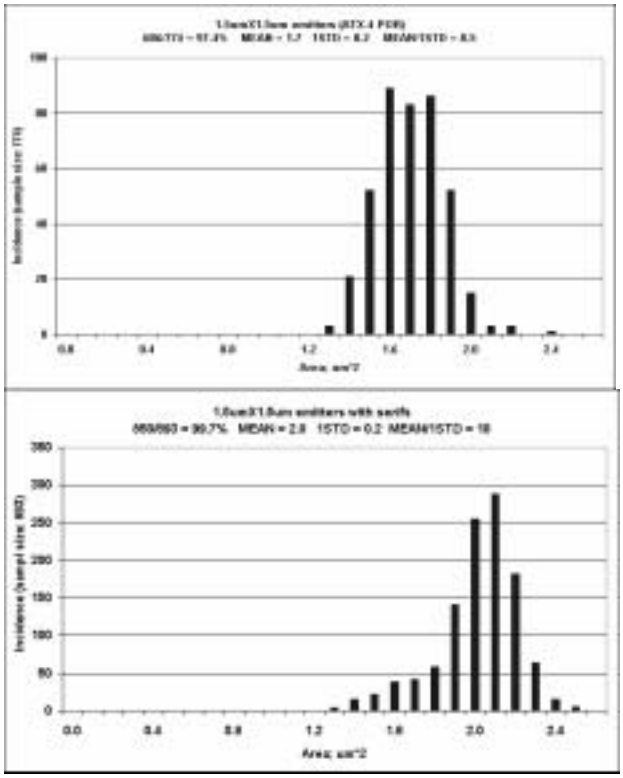


Figure 9. Large emitters with serifs are closer to target and tighter distribution (mean/STD).

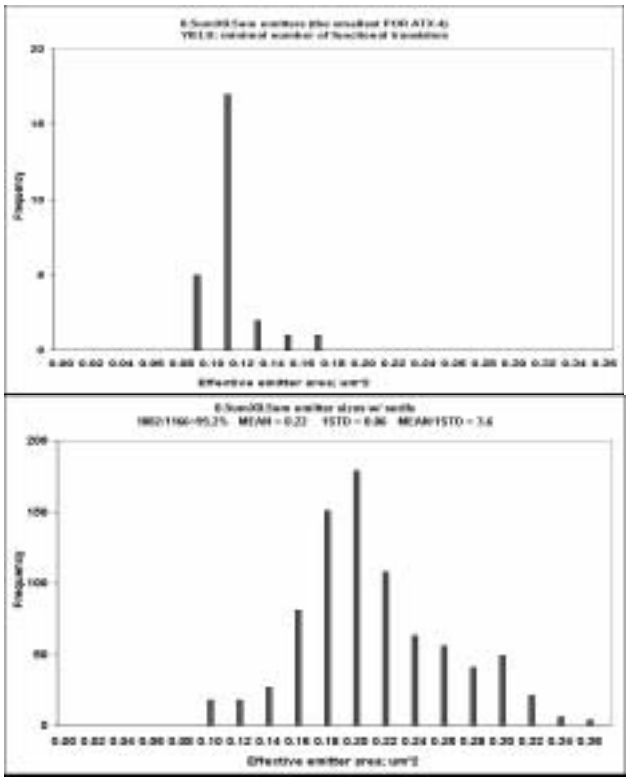


Figure 10. Small emitters without serifs are grossly below size and failing.

Functional test based assessments of patterning [12] point to significant systematic intra-field errors due to design and manufacture. Is yield the best way to validate PDRs?

PDR based dimensional metrology in model images and on wafers is required for sustained improvement of competitiveness and health of our industry.

Required dimensional metrology can be developed.

Effective dialog between designers, lithographers and metrologists is needed to arrive at consensual definitions of Patterning Design Rules. This paper is an invitation for colleagues to get active in PDRs and PDR compliance, to discuss the gaps in the leading technology forums and to publish our emerging consensus in the open domain. This author, with several colleagues joining, plans to take the subject of dimensional metrology for compliance to In-Depth Seminar at Microlithography 2004, like [13].

Our involvement with the suppliers, co-developing the new standard PDR-related dimensional metrology, may then take place at our industry's microlithography and metrology meetings. This will help the extensions of CD-SEM and CD-AFM metrology to emerge, fuel the development of the calibrated patterning/processing models to account for 2-D and 3-D patterning in microlithography and the emergence of the process-aware model-based metrology-assisted WYSIWYG paradigm for design and integration.

Acknowledgments
Discussions on bipolar device emitter Groundrules and what constitutes "optimal design", as well as e-test data in Figures 9-10: courtesy of Fred Barson, IBM Corp. Permission of IBM Corporation to publish the results of this study for the first time, and help of Dr. Michael G. Rosenfield to make it possible, gratefully acknowledged.

References

[1] A. Starikov, "Model-based WYSIWYG: Dimensional Metrology Infrastructure for Design and Integration"*;* Proc. SPIE, **Vol. 4692;** 2002. Fig. 1 is from Short Course "Subhalf-Micrometer Optical Lithography", with co-lecturers Victor Pol (Motorola), James Bruce (IBM) and Burn J. Lin (Linnovation) at SEMICON/WEST '93; July, 1993.

[2] C. N. Berglund, "Trends in Systematic Non-Particle Yield Loss Mechanisms and the Implication for IC Design", Proc. SPIE, **Vol. 5040** and **Vol. 5042**; 2003 (in press).

[3] A. Starikov, "Metrology of Image Placement". In: ***Handbook of Silicon Semiconductor Metrology***, A. C. Diebold, Ed., Marcel Dekker, Inc., New York; 2001.

[4] R. M. Booth, Jr. et al., A statistical approach to quality control of non-normal lithography overlay distributions, IBM Journal of Research and Development, **Vol. 36**, No. 5, 1992.

[5] J. Allgair et al., "Characterization of overlay tolerance requirements for via to metal alignment", Proc SPIE, **Vol. 3677;** 1999.

[6] M. G. Rosenfield, A. Starikov, "Overlay measurement using the low voltage scanning electron microscope", Microelectronic Engineering **17**, p. 439-444; 1992. M. G. Rosenfield, "Overlay measurements using the scanning electron microscope: accuracy and precision", SPIE Proc., **Vol. 1673**; 1992.

[7] Review of state of the art in dimensional metrology: relevant Chapters in Handbook referenced above. Specification for registration marks for photomasks, **SEMI P6-88**. SEMI Specification for Overlay-Metrology Test Patterns for Integrated-Circuit Manufacture, **SEMI P28-96**. Specification for Overlay Capabilities of Wafer Steppers, **SEMI P18-92**. Specification for Metrology Pattern Cells for Integrated Circuit Manufacture, **SEMI P19-92**.

[8] A. Starikov, "Use of single size square serif for variable print bias compensation in microlithography: methods, design, and practice", SPIE Proc., **Vol. 1088;** 1989.

[9] IBM Journal of Research and Development, **Vol. 36**, No. 5, 1992.

[10] D. Schulz et al., Overlay Accuracy in 0.18 micron Copper-Dual-Damascene Process, SPIE Proc., **Vol. 4698**; 2002.

[11] A. Luci, E. G. Ballarin, Optimization of Overlay Markers to Limit the Measurement Error Induced During Exposure by Lens Aberration Effects, SPIE Proc., **Vol. 4698**; 2002.

[12] X. Ouyang et al., "High-Throughput High-Density Mapping and spectrum Analysis of Transistor Gate Length Variations in SRAM Circuits", IEEE transactions on Semiconductor Manufacturing, **Vol. 14**, no. 4; 2001. X. Ouyang et al.; "High-speed mapping of intertransistor overlay variations using active electrical metrology", SPIE Proc., **Vol. 4344**; 2001.

[13] J. A. Allgair et al. "Applications of image diagnostics to metrology quality assurance and process control", Proc. SPIE, **Vol. 5042**; 2003 (in press).

Session 2:

Impact of design-manufacturing interface II

Moderator: Andrew Kahng (*UCSD*)

Layout Methodology Impact of Resolution Enhancement Techniques

Lars W. Liebmann
Semiconductor Research and Development Center
IBM Corporation
2070 Route 52, Hopewell Junction, NY 12533
lliebman@us.ibm.com

Juan-Antonio Carballo
Austin Research Laboratory
IBM Corporation
11501 Burnet Road, Austin, TX 78758
jantonio@us.ibm.com

ABSTRACT
This paper introduces resolution enhancement techniques (RET), discusses the impact that RET have on the physical design and chip layout flow, and proposes two alternative future methodologies.

Keywords
Lithography, resolution enhancement techniques, design for manufacturability, radically restricted designs.

1. INTRODUCTION
While optical lithography has been a key enabler to rapid integration in the microelectronics industry, resolution demands have outpaced the introduction of advanced lithography hardware solutions and made lithographic patterning increasingly difficult. As a result, increasingly complex resolution enhancement techniques (RET) have been required to maintain adequate pattern fidelity. As optical lithography is being pushed closer to its fundamental resolution limit, it is becoming increasingly difficult to implement RET without RET-enabling layout restrictions. While the impact of these restrictions on design rule checking and layout density has been discussed previously [1], this paper focuses on the impact of RET-enabling layout restrictions on established design methodologies.

2. FUNDAMENTALS OF LITHOGRAPHY
2.1 Resolution Limits and Enablers
The resolution limit of a conventional optical lithography system with on-axis illumination can be approximated as

$$R_{min} = 0.5\ \lambda/NA \quad (1)$$

where λ is an illumination wavelength, NA is a numerical aperture, and R_{min} is the minimum feature size, or half the smallest resolvable feature pitch. This equation assumes coherent imaging and a binary system (i.e. using non-phase shifted photomask). How close any given lithography process comes to this theoretical resolution limit is commonly expressed by the Rayleigh factor k_1,

$$R_{min} = k_1\ \lambda/NA \quad (2)$$

Under these assumptions, resolution is proportional to λ and inversely proportional to NA, thereby offering two physical quantities for the reduction of printable half pitch. Unfortunately, it is difficult to increase resolution by increasing NA, because the second fundamental lithography parameter, the depth of focus (DOF, the range of defocus over which adequate feature fidelity can be maintained) has an inverse square dependency on NA:

$$DOF = \lambda/(2NA^2) \quad (3).$$

The Rayleigh factor is also often used as a unitless measure in lieu of feature size, as it expresses how difficult it is to resolve a certain dimension with a given lithography tool:

$$k_1 = \text{Dimension}\ (NA/\lambda) \quad (4)$$

Table 1 shows how lithography solutions have evolved as smaller features sizes are demanded. As expected, wavelength has been decreasing and NA has been increasing. However, k_1 has been continuously declining spite of tooling improvements, i.e., lithography has been loosing ground due to ever harder technology generations. Finally, for each technology generation two distinct lithography solutions can be identified, a very aggressive, low k_1 development phase followed by a somewhat relaxed manufacturing phase.

ITRS Node	Man. Year	Min. Pitch	Dev. λ/NA	Man. λ/NA	Dev. k_1	Man. k_1
180	1999	500	248/.50	248/.75	.50	.76
130	2001	300	248/.75	193/.75	.45	.58
90	2003	214	193/.75	193/.85	.42	.48
65	2005	160	193/.85	157/.85*	.35	(.43)
45	2007	130	157/.85*	?	(.35)	?

Table 1, λ/NA solutions for recent technology nodes [2], illustrating the constant erosion of k_1 for both technology development (Dev.) and manufacturing (Man.) *Potential Solution

2.2 Resolution Enhancement Techniques
As the k_1 factor started to dip below 0.5, resolution enhancement techniques (RET) had to be applied to restore image fidelity. Two examples of RET that enable manufacturing lithography at or slightly below k_1=0.5 are attenuated phase shifted masks (attPSM) and optical proximity correction (OPC).

2.2.1 Attenuated Phase Shifted Mask Lithography
AttPSM lithography improves pattern fidelity by 'darkening' the edges of shapes through destructive interference of light using a mildly translucent photomask. Now commonly called 'embedded attenuated phase masks', mask substrates are used that allow a small amount of light (~7-10%) to penetrate the normally opaque mask regions. Through careful material optimization, the background light penetrates the mask exactly 180° out-of-phase with the light penetrating the clear regions of the mask. As illustrated in Figure 1, this phase shifted background light improves feature contrast at the edges of the printed image. Forcing the electric field vector of the background light to be negative by shifting it 180° relative to the foreground light causes a dark rim in the intensity profile. This 'crisping up' of the printed

images helps to recover some patterning fidelity, but it does not fundamentally improve the resolution or DOF as outlined in Equations 2 and 3. Note that, since attPSM results in no inter-shape phase interference, it can be applied to arbitrary layout configurations with no design restrictions.

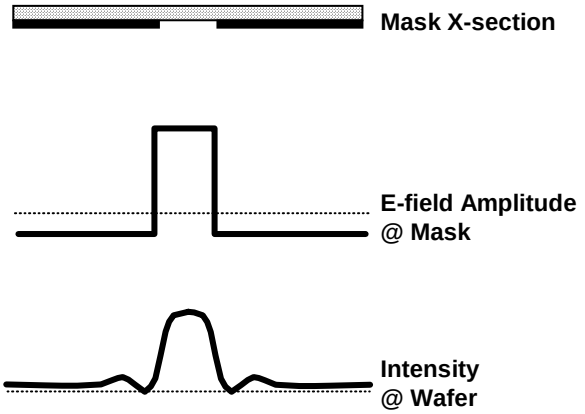


Figure 1, Principle of attenuated phase shifted mask (attPSM)

2.2.2 Optical Proximity Correction

Schematically outlined in Figure 2, OPC begins by characterizing the patterning operation and all its inaccuracies from various sources such as mask build, wafer exposure, etch, etc. In the now commonplace 'model-based OPC' this mathematical description of the process is used in iterative optimization routines to pre-distort the mask shapes to compensate for known, systematic, and modeled patterning inaccuracies.

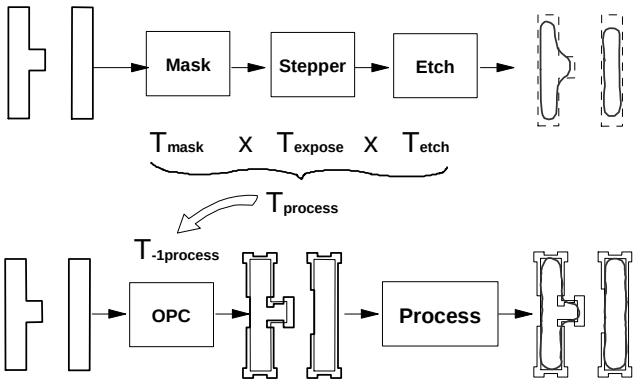


Figure 2, Optical proximity correction

OPC improves the 'effective resolution' of a patterning process by overlapping the conditions with which different feature types can be imaged accurately. Nested features typically image on-size and with the best image quality at a different exposure dose than isolated features. Biasing the mask patterns appropriately will allow both feature types to be imaged adequately in a single exposure. However, OPC does not change the fundamental resolution limits of a lithography system.

2.2.3 Layout Methodology Implications of RET

All the RET implemented up to the 90nm technology node have not altered the fundamental layout methodology as shown in Figure 3. Layouts are generated based on design rules that are enforced through design rule checking (DRC). In this flow, the escalating lithography difficulty is acknowledged through increasingly invasive layout restrictions, but the fundamental layout methodology remains intact.

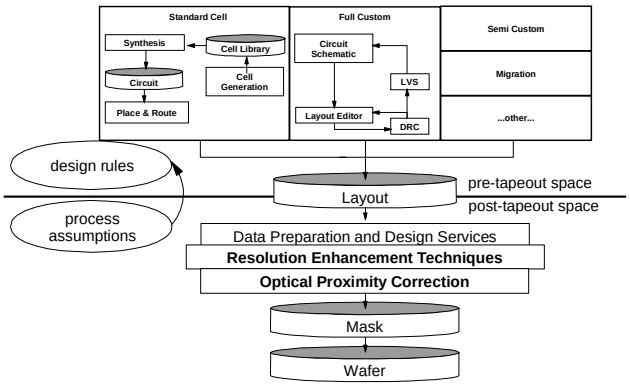


Figure 3, diagram of conventional, DRC-enabled layout flow

3. Future Technology Nodes

3.1 Hardware Options

As shown in Table 1, the 65nm technology node (often referred to as the 1st sub-100nm technology node) is using high-NA 193nm lithography at an extremely small k_1 of about 0.35 in development. Ultra-high NA 157nm lithography was the upfront choice of manufacturing lithography for the 2005 technology node. This would have provided some relief of this extremely difficult resolution challenge but would still leave the k_1 well below 0.5. Unfortunately, technical challenges, predominately related to optical material issues, coupled with significant economic challenges, have caused the 157nm lithography program to slip behind schedule [3]. As a result, 157nm lithography will be late for the entire 65nm technology node.

The prospect of lowering the illumination wavelength to 13.5nm makes Extreme Ultra Violet (EUV) lithography a very attractive proposal. However, significant technical challenges remain in illuminating a reflective reticle, manufactured by stacking 40-50 Mo/Si bilayers at atomic-scale accuracy, with light emitted by a laser produced plasma which is formed by zapping a Xenon medium with a high power laser in vacuum at very low power conversion efficiency. In spite of its technical complexity, EUV is currently moving out of the national laboratories into early commercialization [4]. So it is unrealistic to assume that EUV will have any impact on either the 65nm or 45nm nodes.

Since a shorter illumination wavelength is not available, immersion lithography uses a well-known microscopy trick to improve resolution. Adding a higher index of refraction material in the gap between the pupil and the wafer can improve lithographic resolution. This is due to the fact that in reduction exposure systems (i.e. mask dimension = 4-5x wafer dimension) the resolution of the system is constrained on the wafer side of the projection lens and the resolution and DOF (Equations 2 and 3) should more correctly refer to the wavelength of the exposure light in the medium filling the gap. Immersion lithography is challenging, as it requires introducing wafers in a cleanroom into a watery or oily substance, scanning a precision lens in close proximity at very high speeds, and removing the wafers for further

processing. Despite recent discussions on this technique [5], there are no committed programs and availability dates.

3.2 Strong-RET

Since hardware solutions will be insufficient for 65nm technologies, how can chips be manufactured at $k_1 = 0.35$ if the fundamental limit of conventional lithography is $k_1 = 0.5$? Strong RET provides a solution based on 2-beam imaging. If one of the light sources approximating the mask openings is 'pushed back' by ½ λ, a very different diffraction pattern is obtained. Since the first interference now occurs at an angle that adds ½ λ pathlength difference (rather than 1 λ for conventional lithography) the minimum set of diffracted orders required to form an image for a given pitch are much closer to the center of the imaging lens. For a given NA, the ultimate half-pitch resolution is now given by

$$R_{min} = 0.25\ \lambda/NA \quad (5)$$

or a $k_1 = 0.25$. In addition, no constructive interference occurs at the 0^o angle (the light sources are ½ λ out of phase), so the perpendicular beam is eliminated and with it the DOF limitations of Equation 3. Therefore, 2-beam imaging provides 50% resolution improvement and significantly enhanced DOF.

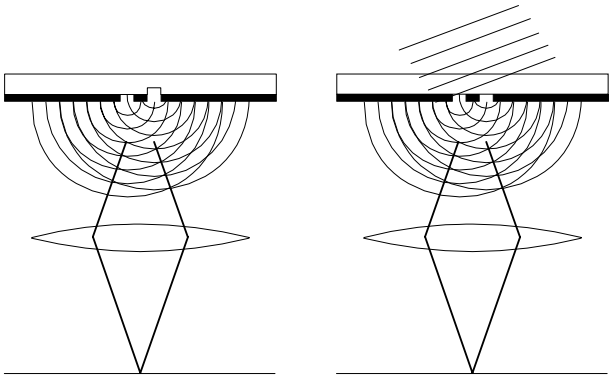


Figure 4, altPSM (left) and OAI (right) produce 2-beam imaging

Two means of achieving 2-beam imaging are shown in Figure 4. To obtain the ½λ phase offset, alternating phase shifted mask lithography (altPSM) manipulates the mask topography to recess juxtaposed mask openings by

$$\text{Etch Depth} = 0.5\ \lambda / (n-1) \quad (6)$$

where n is the refractive index of the mask substrate, typically around 1.4. Off-axis illumination (OAI) achieves the same effect by illuminating the mask at the appropriate angle

$$\sin\theta = 0.5\ \lambda / \text{Pitch} \quad (7)$$

More details on these strong-RET techniques are provided next.

3.3 AltPSM challenges

AltPSM is illustrated in Figure 5 on a pair of transistor-like structures. The resolution-enhancing phase shift is created across the narrow portion of the opaque mask structure, with juxtaposed mask regions exhibiting a step height difference. As shown in the cross-section right of the layout, this is achieved by recessing the appropriate mask region. This causes the electric field amplitude of the imaging light to reverse sign and yields high contrast shadows for the narrow images.

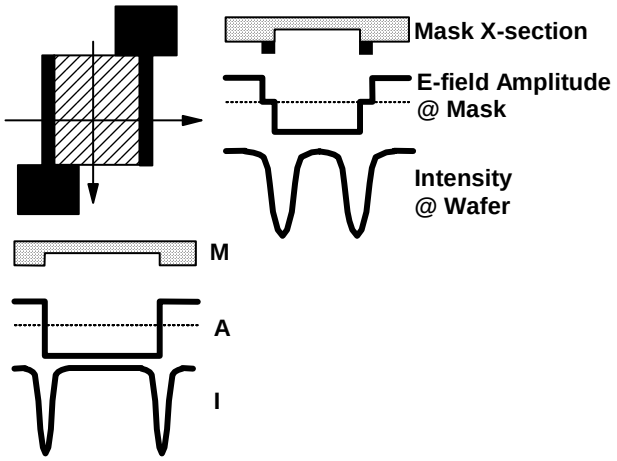


Figure 5 Principle of altPSM

Unfortunately, the recessed region of the mask cannot always terminate on opaque features, causing the printing of unwanted residual images along the phase step as shown in the cross-section below the layout. To address this problem, the lithography community is using double exposure processes, an example of which is shown in Figure 6. In this dark field alternating process the narrow layout segments are imaged by the phase shifted mask (left, layout and image) and a second exposure is used to remove residual images and fill-in the wider portions of the layout (right). The two images add in the photoresist to reconstruct the original pattern (bottom). While this double exposure process adds manufacturing cost, it does not result in additional design impact over single exposure altPSM approaches.

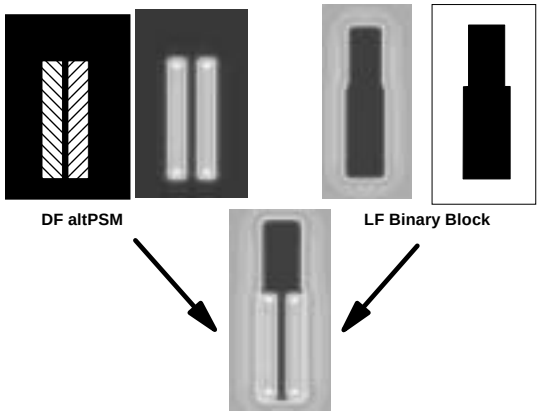


Figure 6 Double exposure altPSM

Identifying mask regions to be recessed requires adding phase shapes to the layout. Lithography and mask manufacturability dictate shape dimensional constraints which in turn prohibit the addition of legal phase shapes to arbitrary layout configurations, driving the need for altPSM-enabling layout restrictions. Layout configurations that are otherwise design rule clean can lead to 'uncolorable' phase errors. A small hypothetical layout (original pattern in solid black, regions of opposite phase in diagonal-hatch) that violates no intra- or inter-shape design rules, yet causes an un-resolvable phase conflict, is shown at the top of Figure 7. Multiple solutions to the phase conflict are also shown.

The optimum layout solution will depend on the specific layout objectives. Key challenges in the implementation of altPSM are (1) lack of reliable design rule checking to guarantee phase-compliant layouts [1] and (2) difficulty in converting abstract colorability feedback into required layout modifications.

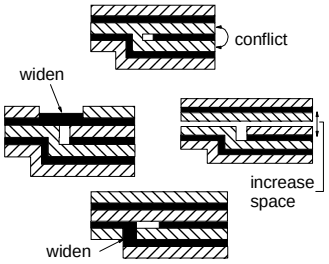


Figure 7 Sample layout conflict (top) and possible solutions

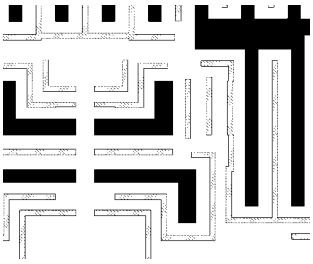


Figure 8 Sraf-enhanced layout showing local sraf conflicts

3.4 OAI Challenges

In OAI, the image is formed by interference between a light beam transmitted perpendicular through the mask (0^{th} –order) and a light beam diffracted by the mask pattern (1^{st} –order). While the illumination angle is chosen to balance the pathlength of these two beams, their light intensities are not balanced and exposure latitude (i.e. the insensitivity to dose variations) is reduced. AttPSM with the correct transmission value can be used to rebalance the intensities of the 0^{th} and 1^{st} diffracted orders and are used in this strong-RET to restore exposure latitude.

The illumination angle in OAI is optimized for a given mask feature pitch (Equation 7). Thus feature pitches significantly different than the pitch for which the illumination was optimized will see much less resolution enhancement. To overcome this problem, sub-resolution assist features (SRAF) are added to the layout [6]. These SRAF are dummy features that are drawn into the layout at a dimension where they optically mimic the diffraction angle of the pitch for which the illumination was chosen, but they are below the dimensional resolution of the lithography system so as to not leave an image in the photoresist.

While OAI and attPSM impose no layout restrictions, the need to add SRAF to the layout increasingly requires SRAF-enabling design rules. The layout shown in Figure 8 presented an acceptable SRAF solution for the 130nm technology node [7], however, the hole in the SRAF coverage just right of center in the layout and the general inability to reconstruct an orderly diffraction grating, will cause significant loss of resolution enhancement for more aggressive applications like the 65nm technology node. The most common 1-D SRAF restriction governs 'forbidden pitches', i.e. pitches that fall in the transition regions between adequate SRAF coverage, highlighting the miss-match of the pitch-centric litho-world and the space-centric DRC-world. In addition, many more complex 2-D constraints will become necessary. Several variations on OAI-attPSM-SRAF have recently been proposed, including double dipole lithography (DDL) [8] and chromeless phase lithography (CPL) [9]. These solutions derive their resolution enhancement from OAI and require the same layout considerations that arise from adding SRAF to a variable-pitch layout.

3.5 Implications

Based on the preceding discussion, the microelectronics industry must adopt strong, highly optimized RET as the lithography solution for all future optical technology nodes. Further, there are no 'miracle RET cures' that avoid layout restrictions. RET-enabling design constraints are the result of tradeoffs between lithographic process window, mask manufacturability, and layout impact. The extremely tight tolerances called for by the 65nm and 45nm nodes leave very little room for lithography tradeoffs.

4. Methodology Impact of Strong-RET

4.1 DRC Shortcomings

Unfortunately, layout constraints required by strong RET cannot be described or enforced through conventional design rules without being unduly conservative [10]. We illustrate this problem here based on altPSM. A design rule to prevent the phase conflict on the simple layout shown in Figure 9, would have to express the equivalent of: "Avoid narrow line-ends surrounded by narrow lines with lateral spacing < (2*Phase-Width + Phase-Space) on both lateral sides and < (Phase-Width + Phase Space) at end."

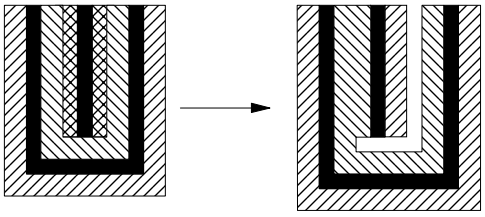


Figure 9 Simple layout forcing complex design rule

This layout restriction is difficult to understand and very phase-parameter specific (and therefore linked to a unique process and altPSM implementation). Simplifications of the layout rules cause inaccuracies that lead to frustration in the layout legalization process. More complicated altPSM errors involve odd-cycles of phases that span many layout features and cannot be predicted even with the most complicated DRC.

Since conventional methodologies, as illustrated in Figure 3, are enabled through design rule checking, and strong-RET require layout restrictions that can not be captured by conventional design rules, new methodologies are needed for future technology nodes.

4.2 RET-embedded Design Flow

One option, referred to herein as the 'RET-embedded design flow' is to enforce layout compliance with strong-RET by moving the RET design upstream in the flow as illustrated in Figure 10. This flow puts the RET design tool directly in the hands of the designers, thereby avoiding the abstraction of RET layout constraints to conventional design rules. Compliance is verified by achieving a layout without RET conflicts rather than via DRC.

This RET-embedded approach presents several challenges. First, committing to a set of RET-parameters long before mask and wafer processes are established (typically finalized 2-2.5yrs after design starts), bears the risk of optimizing to a changing specification. Second, the process-specific nature of RET-specific optimization does not ensure layout compatibility with future lithography solutions thus limiting the layout to one technology node. Third, while significant layout effort is required to make designs RET-compliant (feed-back from the RET tools is much

more abstract than from conventional DRC tools), general manufacturability [11] is not directly addressed. However, this paper focuses on layout methodology challenges, described next.

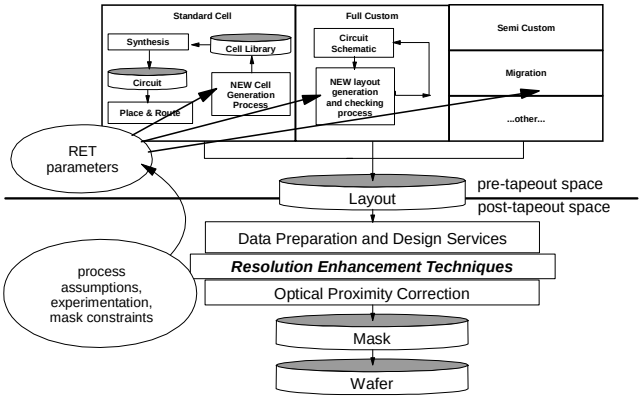


Figure 10. Illustration of RET-embedded layout methodology.

4.2.1 Impact on Custom Cell Design

A major challenge facing designers using this methodology is the fact that layout conflicts, such as the phase-error illustrated in Figure 11, do not suggest a correction as conventional design rules do. The RET tool, in this case the altPSM design tool, reports a layout conflict, but it is up to the designer to manipulate the layout to eliminate the conflict. Some work has been done to improve the cause-and-correction relationship in altPSM design [12], but no fully workable hint-function has yet been developed.

4.2.2 Impact on Placement

The long interaction range of most RET solutions (phase interaction distance is about 5X minimum feature spaces, or approximately .5μm for 65nm designs) requires actively preserving RET-compliance during chip assembly. Failure to do so will lead to conflicts as shown in Figure 12. Short of a RET-aware placement tool, RET-compliance can be ensured either by enforcing large exclusion zones around each cell or by enforcing RET boundary conditions. The former has devastating impact on layout density and the latter adds complexity to the already complex problem of layout legalization in the cells.

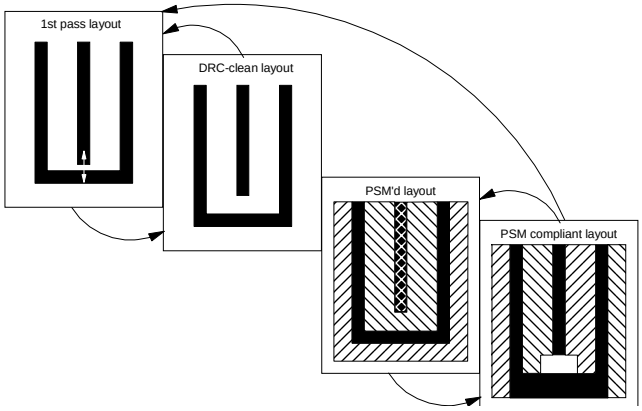


Figure 11. Example of cell level RET-embedded layout flow

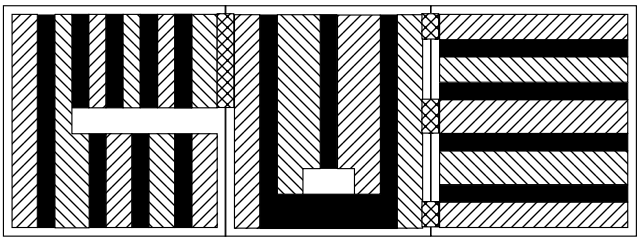


Figure 12. 3 phase-shifted cells with placement-induced conflicts.

4.2.3 Impact on Routing

Typically, the first mask level that needs strong-RET is the poly-conductor level. Since most methodologies prevent cell-to-cell wiring on minimum dimension poly-silicon, RET-aware routing has not been a big issue. However, increasing integration will require strong RET to the metal wiring levels, thereby producing challenges for RET-aware routers as illustrated in Figure 13.

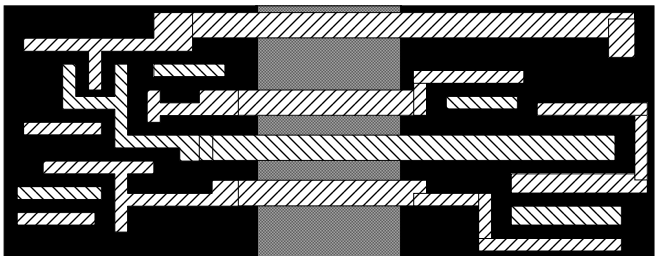


Figure 13, Illustration of challenges for RET-aware routers.

One example of a strong-RET applicable to critical dimension metal levels is dark field altPSM, a variant of altPSM in which phases are assigned to drawn shapes which, for commonly used processes using positive resist, end up as the light transmitting mask openings. After resolving phase conflicts in design blocks and enforcing placement boundary conditions, a PSM-aware router has to connect a wire of appropriate phase to phase shifted metal shapes on both ends of the wire and maintain legal phase coloring (two closely spaced wires have to have opposite phase) over the length of the wire. To address this issue, a RET-embedded layout flow must include phase-design capability in the router, thereby adding algorithmic complexity.

4.3 Radically Restricted Rules

4.3.1 The Design for Manufacturability Mantra

The above-mentioned issues can be addressed by requiring layout design-rules, tools, and methodologies to:

- **Generically enable lithographic RET.** A layout that is optimized for many strong-RET avoids problems associated with early commitment to a specific lithography process.
- **Improve manufacturability at aggressive patterning resolution.** A layout that does not rely on tight control of 2-D detail will overcome two-beam lithography limitations.
- **Ensure designs can be migrated to future technologies.** The investment in a new layout requires reusing designs for multiple technology generations with minimal redesign effort.
- **Allow for density- and performance-competitive chip designs.** Lithography-optimizing constraints that erase the benefits of upgrading technologies node do not make sense.

- **Address a broad spectrum of customer objectives with a single design and process.** A common process solution is needed to leverage mask and wafer cost.

These objectives can be met by moving from a 'minimum perturbation' approach to a 'radical design restrictions' (RDR) approach [13]. By clearly communicating fundamental aspects of the patterning process (e.g. resolution is driven by feature pitch) and fundamental goals of the design (contacted device pitch is the main chip density driver in layout front end), compromised rules can be derived that fundamentally improve manufacturability.

4.3.2 Methodology Impact of RDR

A major benefit of 'radical design restrictions' is the dramatic simplification of the layout methodology. By capturing the 'Design for Manufacturability Mantra' in a set of very restrictive yet easily communicated rules, the established, DRC-enabled layout methodology can be preserved as illustrated in Figure 14.

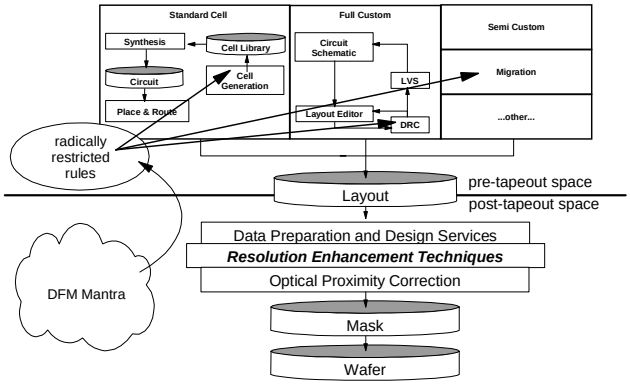


Figure 14. Flow based on 'radically restricted design rules'.

Rather than specifying forbidden pitches and ruling-out complex 2-D constructs, designs are restricted to allow critical dimension features only in one orientation at integer multiples of the contacted device pitch. As the example results in Figure 15 show, a fundamental redesign, involving re-routing the power supplies, can achieve equivalent or better layout density (right) compared to the much less manufacturable, unconstrained layout. The original layout (left) poses many challenges (tight corners, 2-D environments, etc). Addressing these issues by manipulating the CAD polygons causes significant density impact (center). A rigorous redesign achieves all DFM objectives at high density.

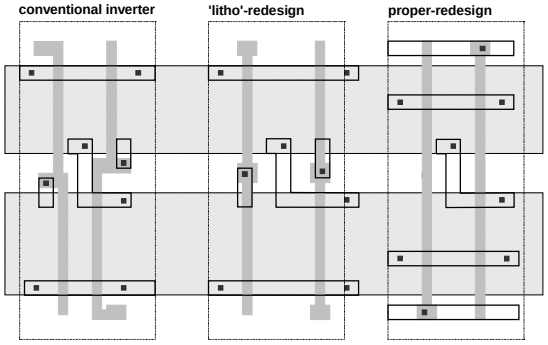


Figure 15. Optimization at design (right) and layout level (center)

Generic RET-compliance is also important for migrateability as is illustrated in Figure 16 on a high-performance, high-density latch design. The radically restricted layout is inherently optimized for all strong-RET, greatly simplifying RET legalization. Note that similar principles apply to dense memory arrays.

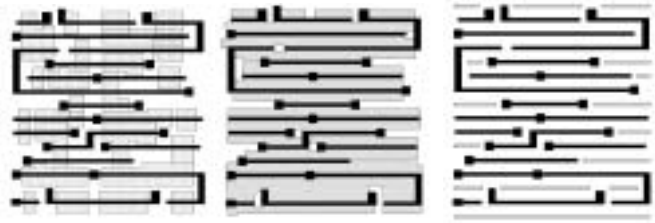


Figure 16. Latch cell designed with RDR (left) is simultaneously altPSM optimized (cntr) and OAI-attPSM-SRAF optimized (right).

4.3.3 Placement

As shown in Figure 17, a fully gridded layout methodology greatly reduces the complexity of RET-compliant cell placement. In a layout where all critical features are placed on a coarse grid, the RET solution, in this case the altPSM layout, is independent of device design. As a result, the RET-layout is identical for all cells and placement rules can be easily established without embedding RET knowledge in the placement tool.

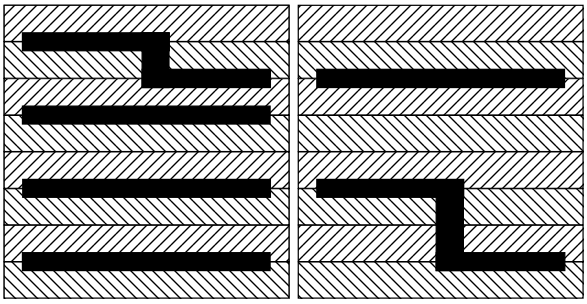


Figure 17. Gridded-layout placement.

4.3.4 Challenges

While the DFM concept is intuitive and provides key benefits to future technology nodes, the following challenges lie ahead:

- **Multi-level optimization tradeoffs**. E.g. avoiding wrong way poly-conductor shapes moves more local interconnect to the first metal level and may increase its complexity; and
- **Multi-parameter layout optimization.** Balancing the needs of RET-enabled lithography, random defect yield, layout density, and chip reliability is not trivial. Even if each need is addressed with first-principle 'rules of thumb', many trade-offs exist, e.g. lithography would favor the diffusion level to be largely rectangular with no small jogs near critical gates, while electro-migration would dictate multiple redundant contacts with extensions on diffusion shapes.
- **Non-technical challenges.** 'Selling' the DFM mantra to fabless companies that often judge foundries by the aggressiveness of their design rules will be a challenge.

5. Conclusion

Optical lithography is approaching a serious wall. While mild-RET solutions have been sufficient and transparent to designers so far, these solutions are insufficient for 65-nm technologies and beyond, and no adequate lithography tools will exist. Strong RET solutions will thus be necessary, but they require layout restrictions that cannot be effectively covered by conventional DRC. This paper has described two design methodology options

to address this issue: RET-embedded design flows, and flows based on radically-restricted rules. RET-embedded flows are significantly more complicated than conventional flows, and affect cell design, placement and routing. Radically-restricted rules impact designers' actions, but result in lower dependence on strong RET and preserves DRC-based methodologies.

6. REFERENCES

[1] L. Liebmann, et al, "Enabling alternating phase shifted mask designs for a full logic gate level: Design rules and design rule",38th Design Automation Conference. Proceedings - Design Automation Conference 2001 p.79-84 (IEEE cat n.01CH37232)

[2] ITRS, http://public.itrs.net/Files/2001ITRS/Home.htm

[3] http://www.imec.be/wwwinter/business/157nm.pdf

[4] http://www.llnl.gov/str/Sweeney.html

[5] http://www.sematech.org/public/news/releases/immersion.htm

[6] S. Mansfield, et al., " Lithographic comparison of assist feature design strategies", Proc. SPIE - Int. Soc. Opt. Eng. (USA) Vol.4000, pt.1-2 2000 P63-76

[7] L. Liebmann, et al., " Optimizing style options for subresolution assist features", Proc. SPIE Vol. 4346, p. 141-152, Optical Microlithography XIV, Christopher J. Progler; Ed. 9/2001

[8] J. Torres, "Model assisted Double Dipole Decomposition," Optical Microlithography XIV, SPIE 4346, 515, 2001

[9] F. Chen et al., "System and Method of providing optical proximity correction for features using phase shifted halftone transparent/semitransparent features", US Patent 6335130 B1

[10] L. Liebmann, et al, "Enabling the 70nm Technology Node with 193nm altPSM Lithography", ", Design, Process, Integration, and Characterization for Microelectronics, Kenneth W. Tobin, Jr., Alexander Starikov, Editors, Proceedings of SPIE Vol. 4692 (2002)

[11] J. Lin, "Semiconductor Foundry, Lithography, and Partners", Optical Microlithography XV, Anthony Yen, Editor, Proceedings of SPIE Vol. 4691 (2002)

[12] Pradiptya Ghosh, Chung-shin Kang, Michael Sanie, Judy Huckabay, "PsmLint: Bringing AltPSM Benefits to the IC Design Phase", to be published Proceedings of SPIE Vol. 5042 (2003)

[13] L. Liebmann, et al.," Layout optimization at the pinnacle of optical lithography", to be published Proceedings of SPIE Vol. 5042 (2003)

Impact of DFM and RET on Standard-Cell Design Methodology

Paul de Dood

Prolific, Inc., Newark, CA
www.prolificinc.com

Abstract

In this paper, we discuss the impact of Design For Manufacturing (DFM) and Resolution Enhancement Technologies (RET) on the creation and use of Standard-Cell libraries. We demonstrate through examples the various types of design-rules and recommended design styles that DFM imposes and how those requirements can be accommodated on a cell by cell basis to maximize yield while still maintaining minimum area.

1 Introduction

Design rule requirements are imposed to ensure a variety of quality metrics, including area, timing, power and yield. Of course, these requirements generally conflict. For example, the minimum width and spacing design-rules are set to be as small as possible to improve area. Any smaller and the design will suffer from unacceptable yield problems. Any larger, and the area of the design will grow with it – which also decreases the yield and increases the cost. The final values for the rules are generally not a "cliff" where making a rule smaller than a certain number results in zero yield, while exceeding a certain number results in zero defects. Instead, the rules are chosen such that the likelihood of a defect is low (but not zero)[1].

Since the 1970's, any novice with a compactor could quickly determine the impact of a global design-rule change to the average area of a standard cell. However, it is far more interesting to see on a cell by cell basis how close each geometry can get to its preferred, highest yield configuration. If the vast majority of geometries can be created using the high yield rules with absolutely no penalty, it would be wrong to ignore the DFM requirements. Likewise, unless the yield effects of a single rule would be catastrophic, it would be foolish to increase the area (and cost) globally to unnecessarily increase a design-rule.

RET requirements introduce additional complexity. The use of RET can be grouped into 4 categories: Optical Proximity Correction (OPC), Phase-shifting, Sub-Resolution Assist Features (SRAF) (e.g. scattering bars) and Off-Axis Illumination. The mixture of RET strategies and manufacturing recipes will dictate what the design should look like. These requirements are represented as a set of design-rules and recommended design styles.

For example, the use of OAI and scattering bars requires either lines to be drawn at an acceptable pitch, or spaced sufficiently far apart such that there is room to place scattering bars – resulting in certain forbidden pitches which cannot be resolved. However, the use of OPC results in the movement of edges such that the final pattern on the wafer is as close as possible to the design intent. Used together, the OPC edge movement can locally affect what the forbidden pitches should be for OAI.

In some cases, the penalty for not using DFM and RET enhancing methods results in large yield penalties. In those cases, the DFM and RET requirements become mandatory design-rules. However, the values are chosen for those design-rules such that yield degradation is an acceptable value. Generally, making the rules more conservative will further improve the yield.

The ProGenesis® software from Prolific enables users to determine the impact of design-rules changes by rapidly and automatically rebuilding an entire library. Also, every design-rule can have both a required and a recommended (or preferred) rule associated with it. By liberally using the recommended design-rules, users can determine how often a particular recommended rule can be applied without increasing area. Finally, all design requirements, including every DFM and RET rule can be individually weighted in importance to create a layout that is truly optimized for area, timing, power and yield.

2 Examples

There are many examples of how designing for manufacturing impacts the standard-cells. We will highlight some of those here. Each example was built using Prolific's ProGenesis® software, which implements all DFM and RET requirements desired as either required or recommended rules.

One potential yield problem occurs when the transistor gate extension is near diffusion, as shown in Figure 1. Note that in the figures, green is diffusion, red is poly-silicon, contacts are white and blue is metal. In this example, the yield is significantly improved when the

gate extension is increased when it is near a diffusion geometry, as shown in Figure 2. This is a sufficiently serious problem that in some processes, the gate extension near diffusion is a required rule, not just a yield-enhancing rule.

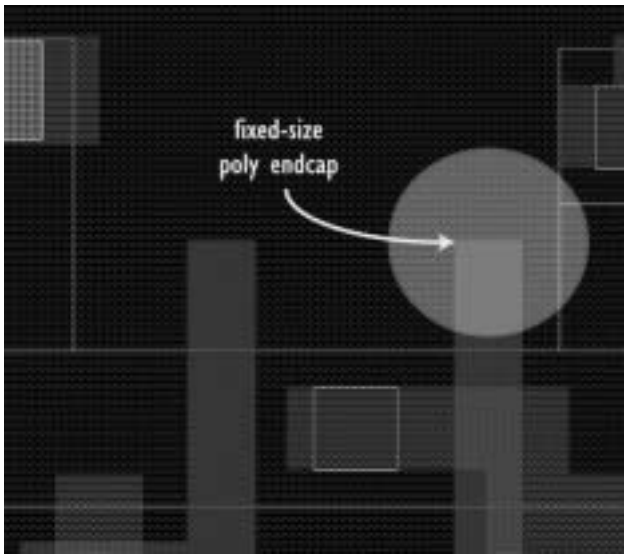


Figure 1. Diffusion near gate extension

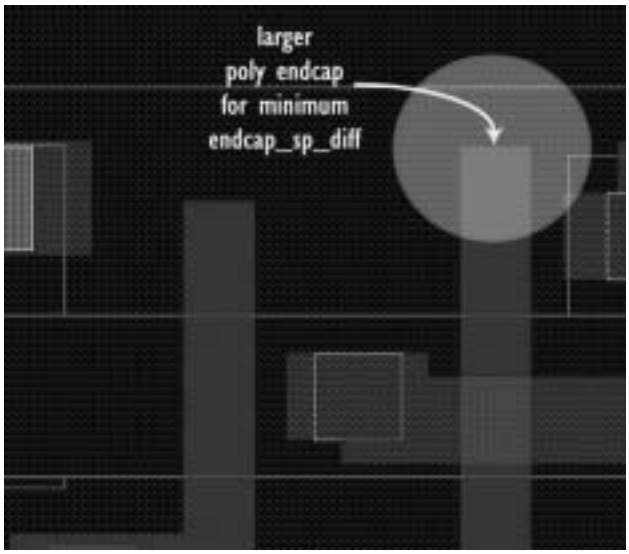


Figure 2. Increasing gate extension near diffusion

Misalignment of contacts and edge placement errors (EPE) due to lithographic effects can cause metal overlap to not fully enclose a contact. This can result in a catastrophic open-circuit, but smaller amounts of misalignment can also cause timing or failures in the field defects due to the increased resistance and electromigration through the contacts. One way of resolving this potential problem is to increase the metal overlap of contact wherever possible. Generally, when layouts are hand-drawn, the overlap is large in only one dimension and minimal in the orthogonal direction, according to the minimum design-rules, as shown in Figure 3. However, yield can be improved by increasing the metal overlap of contact wherever possible. One supported method of improving yield is pick the optimal contact orientation, as shown in Figure 4. Another method is to increase the metal overlap of contact wherever possible without increasing the area of the cell, as shown in Figure 5. In most cases, the overlap can be increased in all four sides, while in some cases, only 3 out of 4 sides can use the larger overlap rule. In this example, forcing the overlap to be large in all four directions would result in an increase in area while improving only a small percentage of the edges.

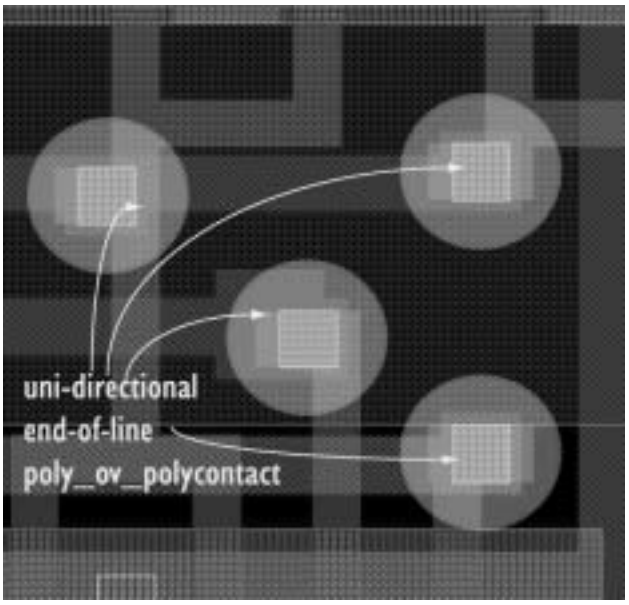


Figure 3 Minimum metal over contact overlap

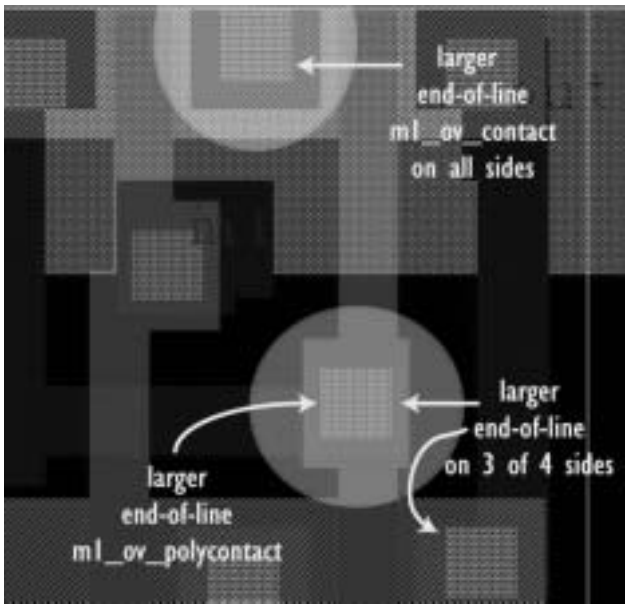


Figure 5 Increased metal over contact overlap

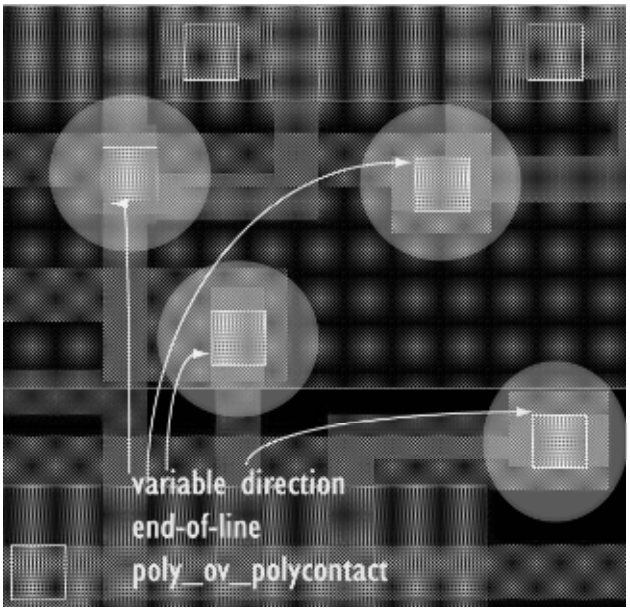


Figure 4 Choosing optimum end-of-line depending on context

In general, the ability to set a geometry to its preferred width or spacing depends only on neighboring geometries. However, often the various preferred rules can conflict with each other. For example, suppose that you have a set of series gates as in the nand4 in Figure 6. Because the width of the cell is dictated by the contacted p-transistors, there is a small amount of slack available in the n-transistor region. One method is to simply give all of this slack to a single gate-to-gate spacing, which would make the spacing between those pairs of gates equal to: *min + slack*, where *min* is the minimum spacing and *slack* is the additional slack available. The remaining gate-to-gate spacings would therefore remain at *min*. However, the critical yield issues often happen when geometries are exactly at the minimum spacing rule, and therefore only one of the three gate-to-gate spacings has improved. A better solution is to distribute the extra slack to eliminate as many minimally spaced geometries as possible, as shown in Figure 6. In this case, the gate-to-gate spacing for each of the three spacings is set to *min + slack/3*, and therefore none of the three gate-to-gate spacings is at the critical minimum spacing.

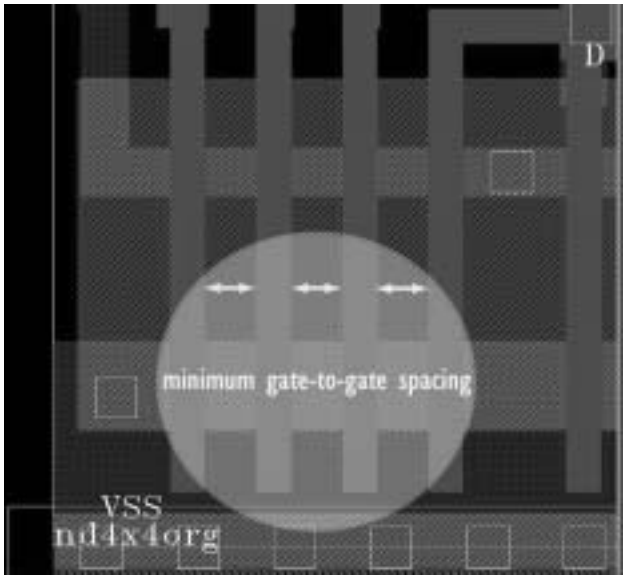


Figure 6 Minimum gate to gate spacing

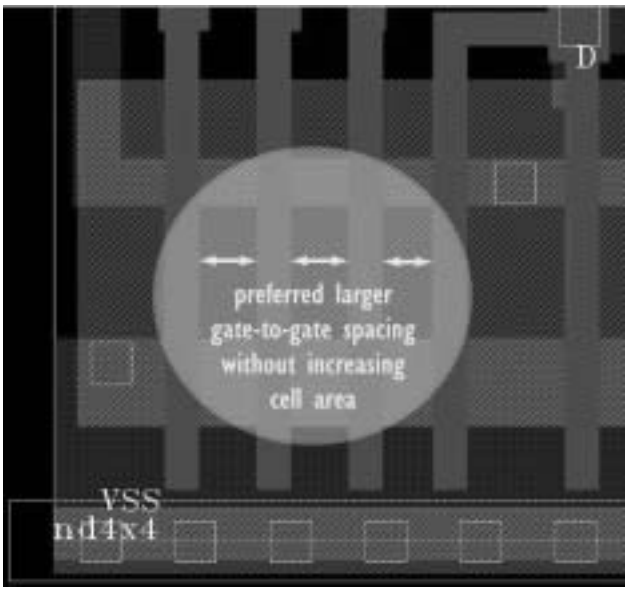


Figure 7 Distributed gate to gate spacing

The use of Off-Axis Illumination results in the resolution being enhanced for certain pitches and degraded for other pitches. For the degraded pitches, scattering bars are inserted at the optimum pitch. However, because of the spacing between scattering bars and design geometries, there are pitches that are degraded but are too small to allow scattering bars to be inserted. These pitches are therefore disallowed and become forbidden pitches. To resolve these situations, the spacing between the geometries must either be decreased to an acceptable pitch, or increased such that scattering bars can be inserted. For example, Figure 8 shows two gates drawn at a forbidden pitch. By applying the forbidden pitch rule, the compactor was able to resolve this spacing by bringing the gates closer together into one of the acceptable ranges, as shown by Figure 9.

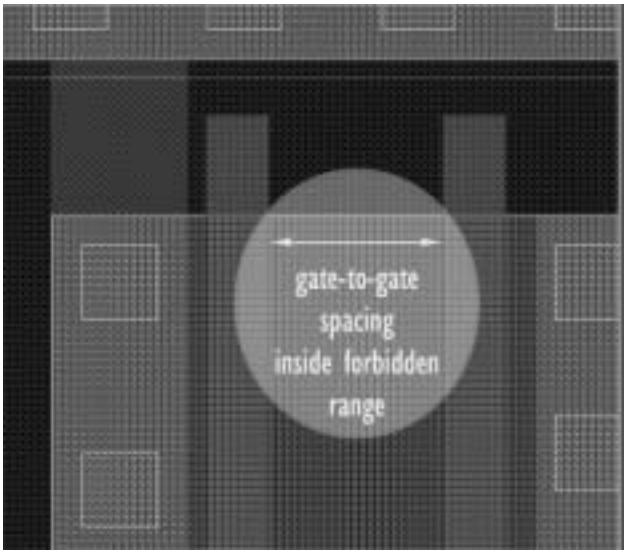


Figure 8 Gate to gate spacing at a forbidden pitch

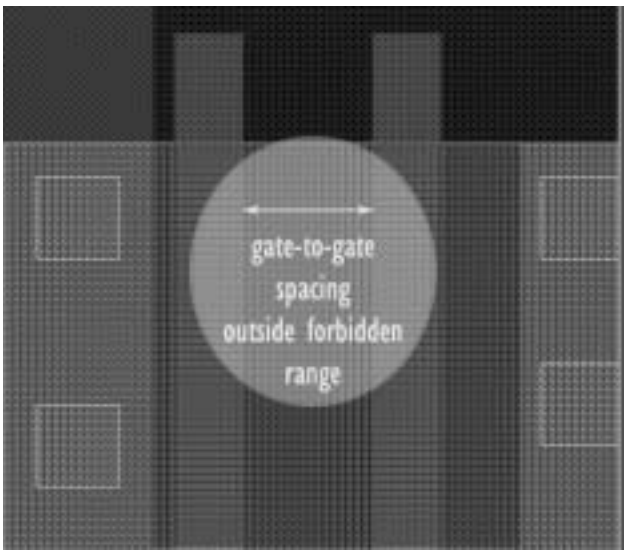


Figure 9 Gate to gate spacing at a legal pitch

The use of strong phase-shifting requires regions to be colored as either 0 or 180 degrees, where neighboring geometries must always have opposite phase. Depending on the layout, it may not be possible to assign a phase to each of the phase regions without assigning the same phase to two neighboring phase regions. This phase conflict must be resolved by either moving the phase-shifted geometries further apart, or by increasing their width such that they no longer need to be phase-shifted to reach the desired resolution. In the example of Figure 10, the entire poly layer is being phase-shifted – not just the gate layer. In order to solve a phase conflict, the width of the center poly wire is increased so that it no longer requires phase-shifting, which breaks the phase conflict, as shown in Figure 11.

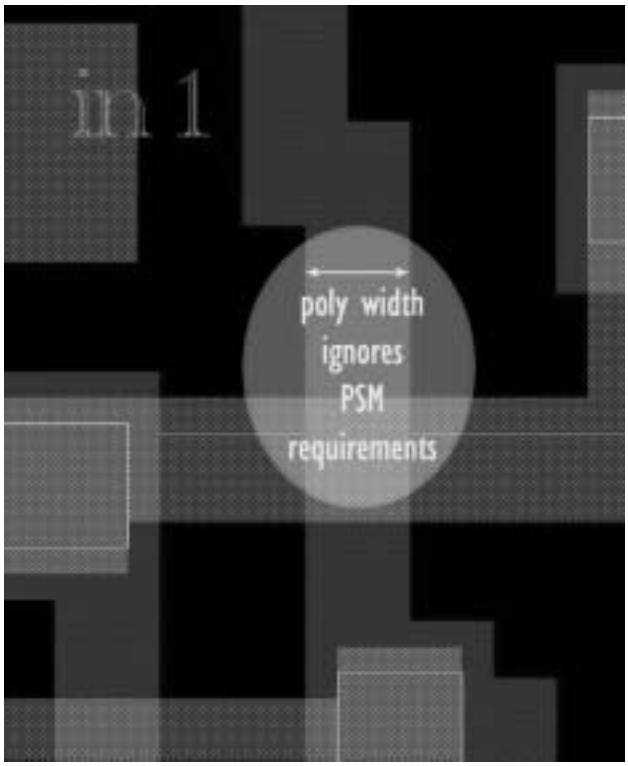


Figure 10. Poly PSM phase conflict

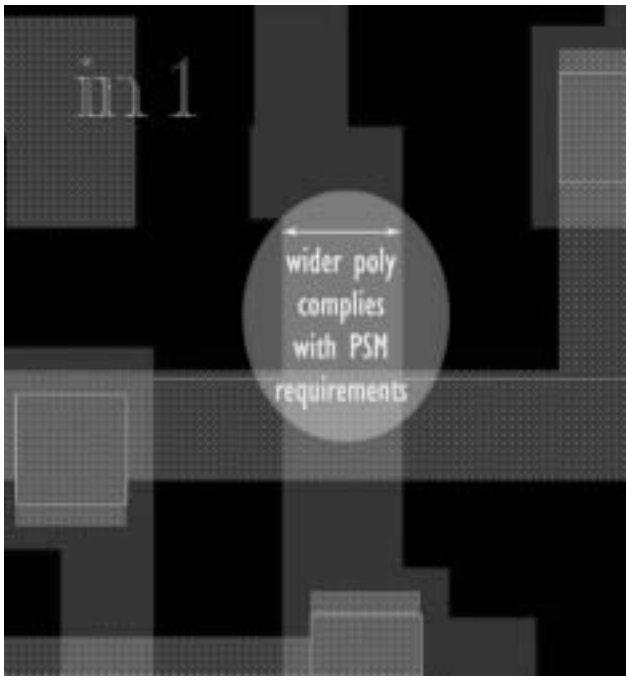


Figure 11. Increasing poly width to fix phase conflict

It is common for the spacing rules for a given layer to be a function of width. However, at the 90nm node and below, the number of different width-dependent spacing rules is increasing and the widths at which they are triggered are getting smaller. At the 130nm node, typically only power-rails were wide enough to trigger a width-dependent spacing rule. However, at 90nm, even the metal overlap of contact or an equivalent width wire can have a different spacing requirement than a minimum width wire. The more common case is shown in Figure 12, where the spacing from the diffusion contact to the power rail is too close because of the width of the power rail. This is resolved in Figure 13, increasing the space between the contact and power rail.

Note that in Figure 13, the ProGenesis® compactor was able to maintain the 3 diffusion contacts by moving the entire structure up, rather than simply etching back the metal, which would have resulted in the loss of the bottom-most diffusion contact. This demonstrates the advantage of using a compactor to resolve these complex rules to find the global solution, rather than using a simple script or manually cleaning up the errors which may result in a valid, but sub-optimal solution.

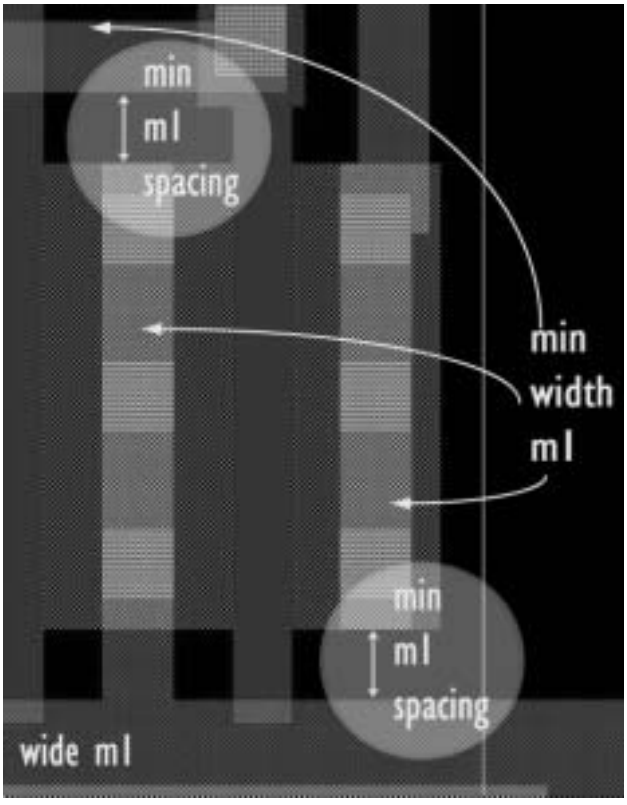


Figure 12. Metal to wide metal spacing

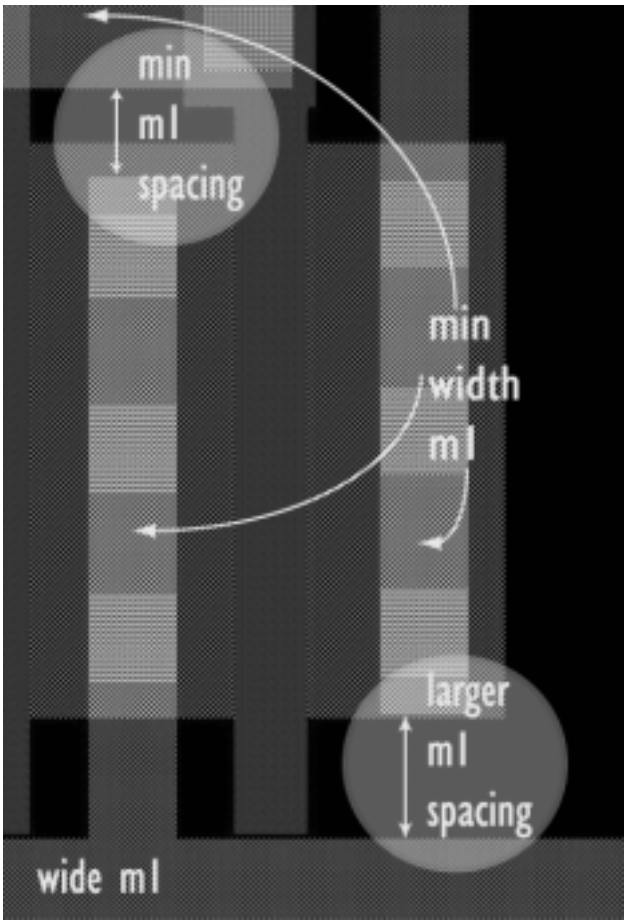


Figure 13. Increased spacing to wide metal

Another method of improving the RET friendliness of a layout is to reduce the number of vertices. The complexity of the OPC depends on the number of vertices, as do the resulting data files. While often jogs are necessary to minimize the area of the layout, some vertices can be removed without an area penalty. For example, the drain region in the leftmost transistor of Figure 14 has a small notch due to the mismatch between the diffusion contact size and the transistor diffusion extension. This can be easily filled without area penalty, as shown in Figure 15.

This example also demonstrates the addition of redundant contacts, which is another DFM enhancement. Because contacts are often a source of yield problems, having redundant contacts can improve the yield and certainly reduce the resistance. In Figure 15, the source contact has been doubled to reduce resistance and increase yield, without an area penalty.

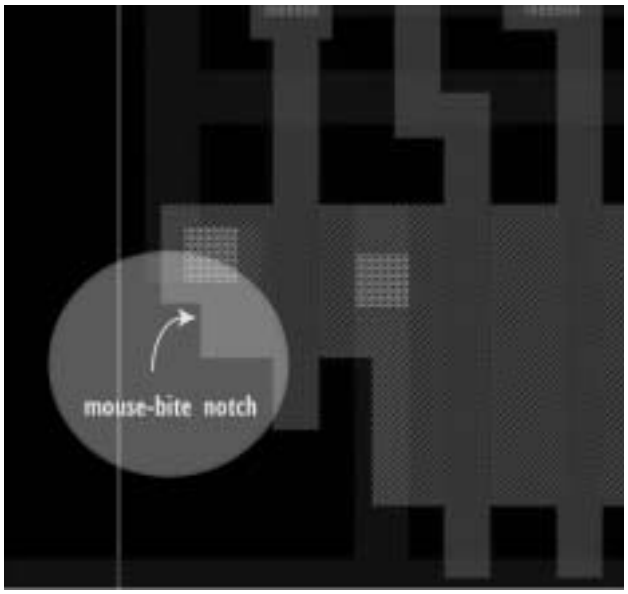


Figure 13 Minimum capacitance transistor drain

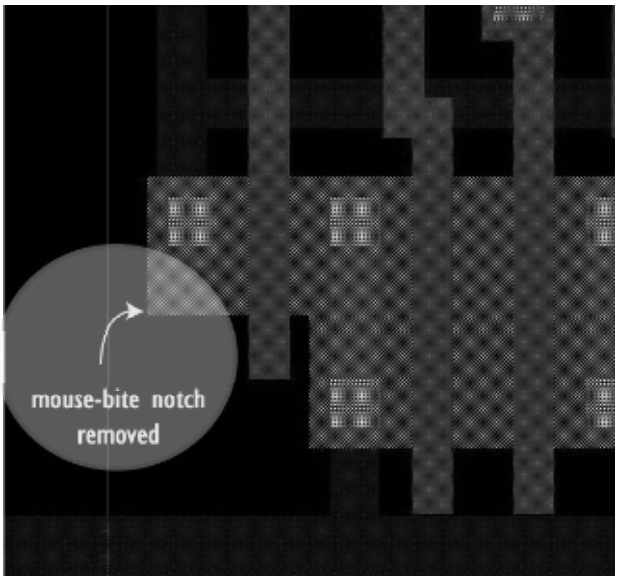


Figure 14 Reducing number of vertices (and resistance) on transistor drain

3 Conclusion

This paper has demonstrated through examples the impact of design for manufacturing and resolution enhancement technologies on standard cell design. Even though there are many additional requirements on the layout design, we have shown that the additional requirements can easily be handled by a DFM aware automated layout creation system. Furthermore, by experimenting with altering not only the required design-rules, but also the recommended rules, the smallest and best possible yielding layout can be created and the standard cell library can truly claim to be Designed For Manufacturing.

References

[1] J. A. Torres, D. Chow, P. de Dood and D. J. Albers, "RET Compliant Cell Generation for sub-130nm Processes", *SPIE*, 2002.

The importance of layout density control in semiconductor manufacturing

Vivek Singh

TCAD Department, Intel Corporation, USA

Abstract

This paper shows that specification of simple constraints on layout density can result in substantial reduction of within-die variation in a number of different processes. The concept is illustrated with examples of characterization of process-layout interaction based on layout density, and subsequent control of process variation through control of this density.

1 Introduction

As the demand for high performance IC products continues to grow, the industry has sought to reduce both device size and within-die variation of these devices. To achieve this, process development engineers have needed to address both these aspects in delivering processes to the factory, often having to balance opposing directions. For example, in lithography development, a higher numerical aperture of the stepper lens is necessary for printing smaller poly lines, but this also reduces the depth of focus needed for process robustness. While creative process engineers and materials scientists have managed to juggle such varied requirements to eventually deliver a process that meets them all, this has ended up pushing processes to a point where the slightest variation in operating conditions can cause instability and, often, fatal loss of yield.

Designers and process engineers have tried to address this present and deepening crisis with a number of collaborative approaches (Reference 1). One approach has been to accept intrinsic limitations in individual process modules, and compensating for them by tuning other modules, as well as the design itself. This has suggested the coupling of TCAD and ECAD tools – an approach that is promising but not without pitfalls. The two main problems with the concept of using coupled tools and methodologies are 1) a significant increase in design time, and 2) greater probability of small errors multiplying without early detection. These reasons have so far limited widespread proliferation of coupled tools, although this will change when the related economics become favorable. The other approach designers and process engineers have used is to agree on additional design rules, intended to make the processing easier, or even feasible in some cases. These design rules vary in terms of complexity of specification and the extent of restrictions they require. The most frequent and restrictive proposals (as evident, say, from the body of recent literature on design-process integration) arise from a lithographic perspective, which is not surprising if one considers the relative cost of patterning, particularly going from one generation to the next. An example of one of the more radical design rule proposals is designing on a regular, coarser grid, which would allow the lithographer to focus the process to print primarily lines and spaces, which in turn can be improved by several techniques such as the use of Alternating Phase Shift Masks. Needless to say, such dramatic proposals pose greater challenges to the designers and as such are slower to materialize. Other design rules are less radical but still require substantial change in the design process; an example is the orientation of gates in a single direction to diminish the impact of lens aberrations that cause differences between horizontal and vertical features.

This paper addresses a simpler design rule – concerning layout density – and provides rationale and some examples from a process development perspective to highlight the importance of this aspect of design-process integration.

2 Shrinking Process Margins

It is useful to briefly review the shrinking process margins encountered by process engineers today, as a motivation for instituting any design methodology changes. In order to limit the scope of that review, this section will focus on process margins in lithography, partly because the lithography process latitude budgets often indirectly take into account the variations in other processes such as Chemical-Mechanical Polishing. In order to do this, let us first introduce here the term NILS, or Normalized Image Log Slope. This metric has been used to describe the process latitude in lithography, and can be described as the percentage change in exposure dose needed to cause a unit percentage change in patterning CD (e.g. the width of a line). By calculating NILS at various defocus conditions, lithographers obtain a measure of process latitude in both dose and defocus. Defocus, in turn, is caused by a number of effects in the process, one of them is the topography variation in the thin film over which the photoresist is spun. Figure 1

shows that NILS values have been decreasing for subsequent process technology nodes. In other words, the process margin in lithography, to accommodate process variation in other process modules, has been decreasing. Similar trends can be documented for other process modules, and it is accurate to conclude that most process modules are becoming sensitive to various process parameters.

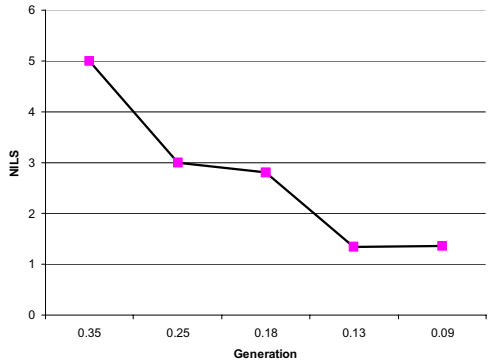


Figure 1: A plot of Normalized Image Log Slope at maximum allowed defocus versus technology node

One such process parameter for many wafer process modules is the layout pattern that has been "inscribed" in the wafer before that processing step, or, in the case of photolithography, the layout pattern in the mask for that process step. Depending on the details of the given process, different layout patterns can cause different processing end results. The central point of this paper is that, for many processes, a simple layout property, namely feature density, is increasingly useful in characterizing the nature of the process-layout interaction. Understanding this interaction is particularly useful in the face of the aforementioned shrinking process margins.

Next we will consider two examples of interaction of layout with process, and illustrate the use of layout density to characterize these interactions.

3 Addition of dummy features for CMP uniformity

The first example of process and layout interaction concerns the Chemical-Mechanical Polish (CMP) process. It has been observed that the rate of polishing is proportional to the pressure that the polishing pad exerts at the point of contact between the pad and the topographically uneven film being polished. Given a constant force on the pad, this translates to a greater pressure when a fewer number of topographic "hills" are in contact with the pad. In other words, the rate of polish is greater when the density of layout features that are patterned as hills is smaller; this is illustrated in Figure 2.

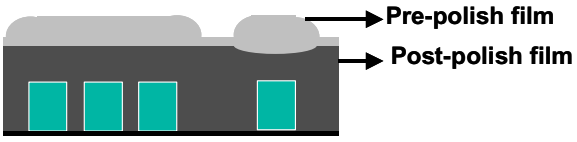


Figure 2: A schematic of the dependence of polish rate on layout dependent wafer topography.

This allows one to model CMP as a function of layout density, and subsequently calculate within-die variation of thickness of the polished film. Needless to say, microprocessor chips are not characterized by repetitive features, i.e. the layout density of these chips is not uniform. As a result, the post-polish film thickness is very non-uniform. As was noted earlier, this non-uniformity can cause local defocusing in subsequent lithography steps, which in turn results in flawed patterning due to the reduced process margins.

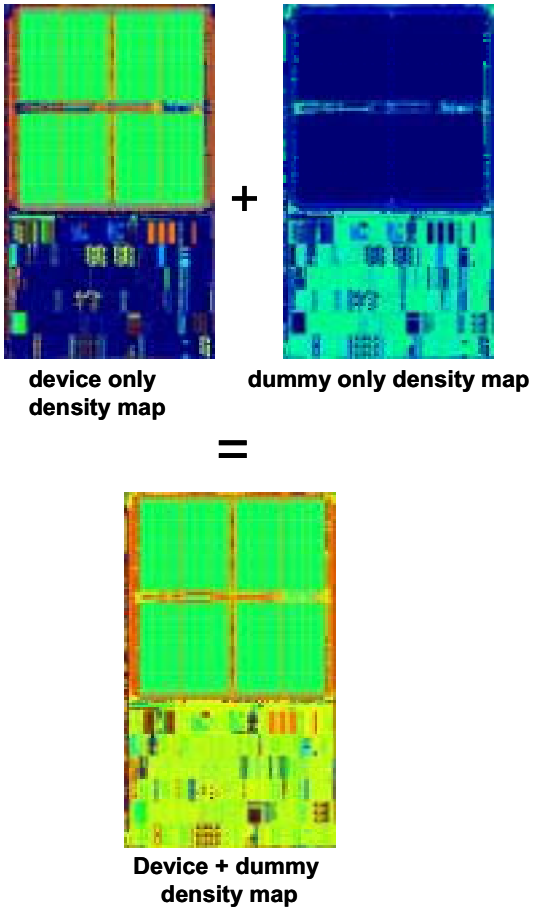


Figure 3: Thin film thickness variation before and after dummification.

One way to overcome this cascading chain of variations and failures is to minimize the variation in layout density by adding dummy features in relatively "open" areas. One illustration of the usefulness of this dummification for CMP uniformity is shown in Figure 3, which is a map of the layout density of a chip, defined as the percentage of the area in a given area (typically about
10μm X 10μm) covered by chrome. Note that the map before dummification shows a wide variation in this layout density. In order to suppress the resulting non-uniformity of the polished film, dummy features are added; a map of this dummy density is also shown. The two maps are added together to get the layout density of the final chip. If this process is iterated while varying the size of the dummies, one arrives at a final layout density map that has minimum variation.

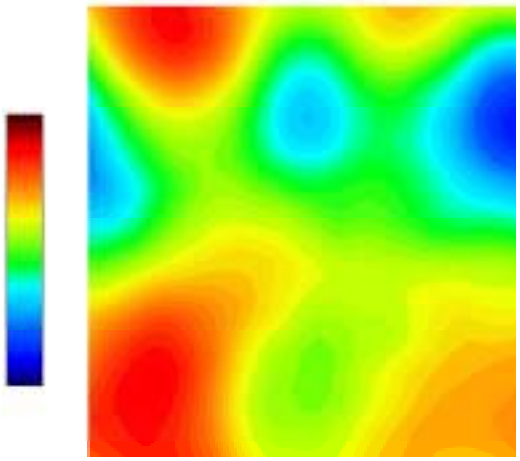


Figure 4: Modeled device performance before dummification.

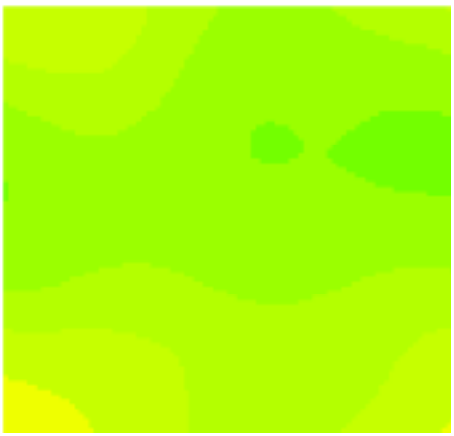


Figure 5: Modeled device performance after dummification.

Thus, the specification of a simple design rule constraining the layout density within a range is necessary to enable a relatively robust CMP process, and subsequently good lithography. The key here is to specify the "window" over which this density needs to be calculated. This window arises from the length scale characterizing the physical process – a parameter that can be obtained both from empirical characterization of this variation as well as analytical or numerical investigation of the underlying phenomena.

It is interesting to note that making the layout density more uniform can affect more than one process modules. For example, besides the variation arising during the CMP process as described before, layout density can also include the etch process. This is commonly referred to as "microloading", alluding to a mechanism wherein etch precursors are consumed at a greater rate in areas which have a greater density of surfaces that are subject to etching. Figures 4 and 5 show a variation in modeled device performance before and after the addition of dummy features, and it is suspected that the post-dummification uniformity results from both polish and etch improvements.

While the process of dummification for improved CMP has existed for a while, new advances in this field will include the process of coupling this simulation with other modules. For example, one might use the final film thickness uniformity map as an input to a lithography simulator. Depending upon the resist planarization conditions, this integrated simulator can be used to study the CD variation across the chip as a function of film uniformity.

4 Layout dependent flare in EUV

The second example of process design interactions is taken from a process technology for the future: EUV lithography. It has been found that residual roughness on the mirrors that act as optical elements in the EUV system causes undesirable light scattering, referred to as flare. This flare has the effect of altering the critical dimension of features, by a mechanism roughly similar to introducing extra exposure dose. If the flare were constant across the entire die, the problem would be less significant; unfortunately this is not the case.

One would like to calculate the effect of mask layout on the flare at any given point on the wafer. In order to do that, it is useful to derive a point spread function, PSFsc representing the scattering by the roughness in the mirrors (References 3, 4). This PSFsc can then be convolved with the aerial image of the mask to give the resulting flare variation across the die. The following equation gives an example of such a PSFsc, which is

representative of the early EUV systems that have become available.

$$\mathrm{PSF_{SC}} = \frac{0.166}{r^{2.39}} \frac{1}{\mathrm{nm}^2} \quad \text{for } r > 600\mathrm{nm}, \text{ zero otherwise}$$

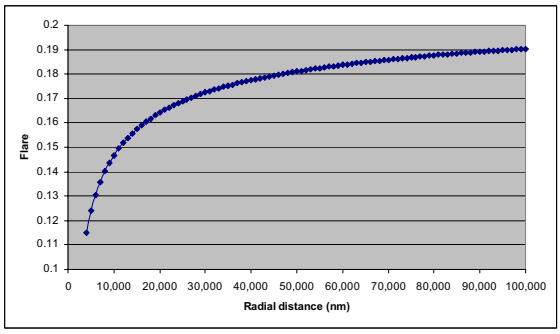


Figure 6: The cumulative contribution of flare as a function of distance from the point of calculation.

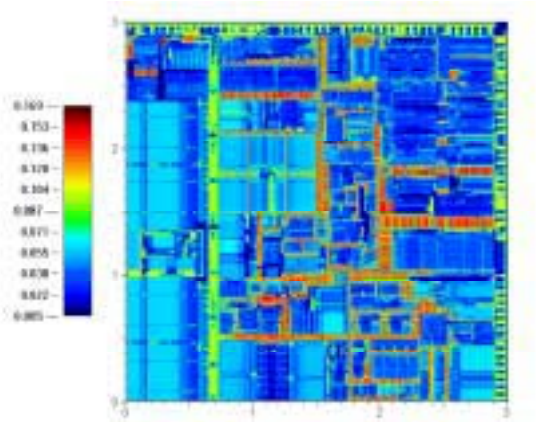


Figure 7: The variation of flare within the die.

Note that the value of this PSFsc first falls relatively quickly away from the point of interest, but then decays much more slowly. This means that the layout has a strong non-local effect on the flare at any point: even features that are millimeters away from this point affect the flare. This slow fall-off is plotted for an open frame exposure in Figure 6. To study this effect, a piece of layout from a processor of the 0.18 μm technology generation was extracted, and shrunk by a suitable factor to give an estimated piece of layout to be manufactured by EUV technology. The feature density of this layout was extracted, and convolved with the PSFsc function to give an approximation of the flare variation within this piece of the layout, as shown in Figure 7. It is clear that this flare variation will be minimized if feature density variations in the layout are minimized. This leads to either a direct constraint on the feature density variations of the layout to be imaged by EUV lithography, or the concept of placement of dummy features in the layout to indirectly constrain feature density on the appropriate grid. The dummification schemes will probably need some modification for application to EUV lithography. It is to be noted that one special consideration would be the fact that the edge of the field would normally receive less flare than the center of the field. Two possible methods of addressing this concern are listed here. The first would involve an investigation of whether the optical system can be modified, without other deleterious effects, in a manner such that the lower flare at the edge of one die is compensated by additional flare during the exposure of the adjacent die. The second possible solution is to vary the size of the dummies as a function of location within the field. Clearly, the exact scheme for EUV dummification requires further study.

5 Discussion

In the preceding two sections, we have seen that controlling layout density has great potential to improve the subsequent manufacturing, particularly for reducing within-die variation. Given that process capability of any module in today's environment has a component relating to within-die variation, it follows that this capability can be improved at least to some extent by specifying design rules on layout density, while not having to resort to more expensive or exotic processing techniques. The key reason for the increasingly urgent discussion on design process integration is that economics prohibits process engineers from delivering process capability for unrestricted layout – clearly layout density control is one of the simpler restrictions.

Acknowledgements

The author would like to thank the following individuals for their help: John Bjorkholm, Yan Borodovsky, Jorge Garcia, Nayanee Gupta, George Sery, John Swanson.

References

[1] V. Singh and J. Garcia-Colevatti, "Relevance of Technology Computer Design to Process Aware Design", JM^3, 1(3) 290-295 (October 2002).

[2] G. Sery, "Approaching the one billion transistor logic Product: Process and Design Challenges", Paper 469236, Design and Process Integration Conference for Microelectronic Manufacturing Conference, SPIE Symposium 2002.

[2] D. G. Ster Stearns, D.P. Gaines, D.W. Sweeney, and E.M. Gullikson, J. Appl. Phys. **84**, 1003 (1998).

[3] "EUV Scattering and Flare of 10X Projection Cameras" by E. M. Gillikson, S. Baker, J.E. Bjorkholm, J. Bokor, K.A. Goldberg, J.E.M. Goldsmith, C. Montcalm, P. Naulleau, E. Spiller,

D.G. Stearns, J.S. Taylor, and J.H. Underwood, Proceedings of the SPIE Conference on Optical Microlithography XII (1999).

Session 3:

Unified Databases and Data Models

Moderator: Gary Smith (*Gartner*)

Design Systems Evolution and the Need for a Standard Data Model

John Darringer
IBM T J Watson Research Center
Yorktown Heights, NY
and
Joseph Morrell
IBM Microelectronics
East Fishkill, NY

Abstract
Design systems have evolved over the years from batch processing systems to highly integrated suits of tools sharing design data in memory. To support today's and future chip designs, a user must have such a tightly-integrated system and is forced to purchase the entire system from a single vendor. Small EDA companies must support many proprietary interfaces to market their new tools. To enable choice in purchasing tools, to promote innovation by lowering the barrier to entering the EDA market and to accelerate the transfer of innovative research into production use, the industry needs a standard data model and a common API for accessing design data in memory. The requirement is more than a "nice to have", it is essential for designers to keep pace with technology and will only be met with strong user support.

1 Introduction

Silicon technology has had a truly remarkable record of progress over the last fifty years from chips with a single gate to today's system-on-a-chip designs with a capacity of nearly 100 million gates. Design systems too have evolved from initially just keeping track of component interconnections to automatically generating gates for chips that satisfy constraints on area, power, performance, noise and yield [1]. IBM began developing design systems in the 1950's to support its designer's use of its most advanced technology. As technology advanced and designs became more complex, design systems needed to consider the growing number of issues confronting designers. Moreover, the historically sequential process of optimizing one factor at a time has been replaced with *tightly-coupled* applications that work together to simultaneously consider multiple factors.

Today, IBM still builds its own design systems, but tries to take advantage of vendor tools when ever possible. In the past the task of evaluating and integrating a vendor tool took time, but was helped by the "standard" file formats. But, we can no longer afford to have files between key design system applications. They must work with shared design date in memory through an application program interface (API). The lack of a standard here means that we can rarely afford the time and effort to evaluate a new tool from an EDA vendor or a prototype from a university.

For design systems to keep pace with technology they must continue to evolve. The need to tightly couple critical analysis and optimization applications through shared memory will continue to grow. Without a standard way of connecting these new applicators there will be no way to combine the best algorithms from different suppliers. The result will be a market with a few very expensive design systems each with a subset of the best applications.

Design Systems in IBM - The Early Years

IBM built its first design system in the late 1950s [2]. It

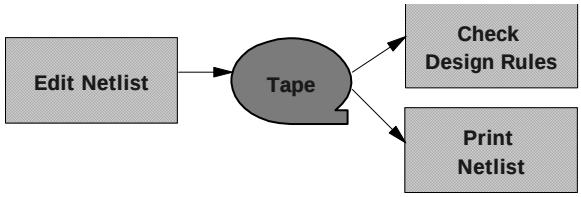


Figure 1 - 1950s Design System

used a central data base, stored on tape, to manage the logic diagrams for its 9000-circuit 7094 computer.

In the 1960s the 3033 machine had a CPU of 90000

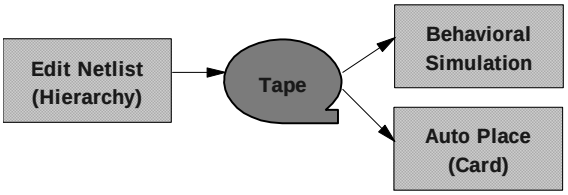


Figure 2 - 1960s Design System

circuits and hierarchy was introduced to handle the growing design information. A behavioral simulator was added to help confirm correct operation and an automatic placement tools was added to position chips containing a

few gates on cards. The data base was still maintained on tape.

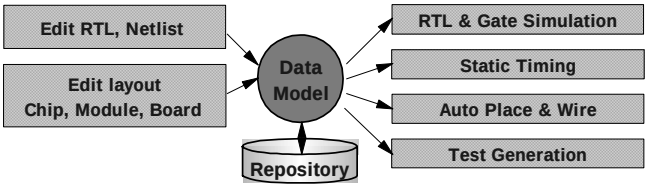


Figure 3 - Engineering Design System

In the late 1960s and early 1970s a more integrated Engineering Design System emerged (EDS) [3]. It provided a random access data base stored on disk along with an early data model that allowed key applications to share data in memory.

EDS maintained both logical and physical data including layouts for chips, cards, boards and cables. Register-transfer level simulation was introduced to handle larger design, such as the 3081 with its 460K circuit CPU. Automatic placement and wiring was provided for the 704 circuit chips as well as the 100-chip water-cooled modules and the 8-module boards. Static timing analysis was used to verify the systems performance, and automatic pattern generation was used to produce vectors from test in manufacturing. EDS was expanded and used throughout the 1970s and 1980s to support all of IBM's advanced hardware products.

Tight-Coupling Begins

In the 1980s, IBM pioneered the production use of logic synthesis by automatically generating most of the chips for its large mainframe systems. Hundreds of chips were processed using a sequential flow with full-chip synthesis followed by static timing analysis. The output of timing was used to revise constraints for another full-chip synthesis. This process usually converged in a few iterations with a satisfactory implementation. But as chips became larger and more complex, runtimes became a concern. At this point *Incremental Timing Analysis* was developed and integrated with synthesis [4]. Synthesis transformations could then consider a small change to the chip logic and calculate the impact to timing and then decide to commit the change or back it out. The new timing analyzer would only recalculate timing for the effected components. This advance not only greatly improved the time required for closing timing, but served as an example for future design system evolution.

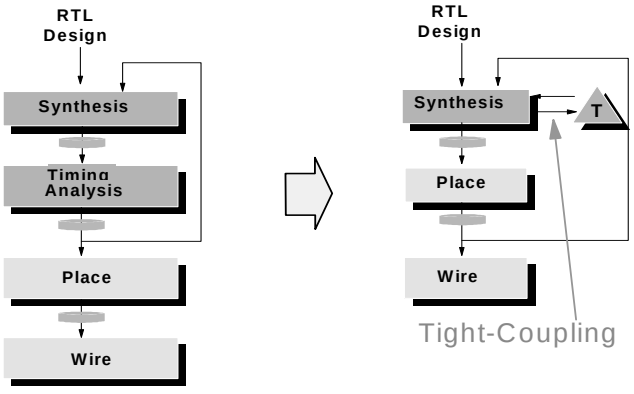


Figure 4 - Initial Tight Coupling 1980s

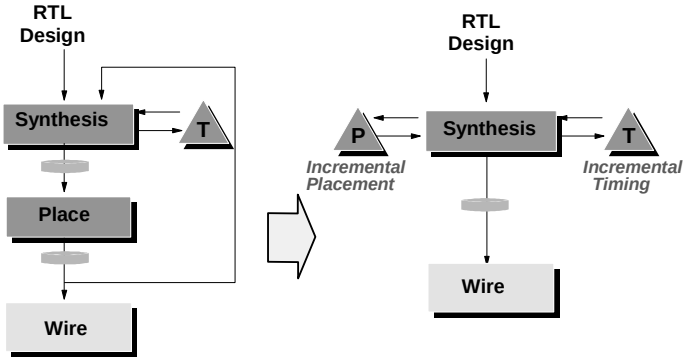


Figure 5 - More Tight Coupling

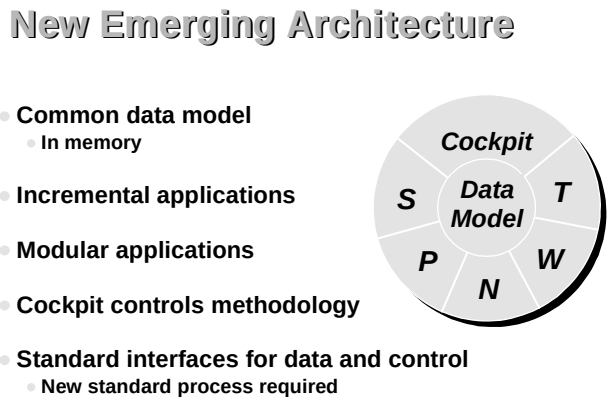
During the 1990s, as chip designs continued to grow in size and interconnect delays became a larger portion of the system cycle times, placement started to have a larger effect of performance. For some million-gate designs the iteration between synthesis and placement became a concern and *Incremental Placement* was integrated [5]. Now a change could be considered to the logic and its placement with excellent prediction of the impact on final chip performance.

These new incremental tools were completely new applications written to operate on a representation of the design stored in memory. Extensions were required to the data model to make each advance possible and the design of the data model was key to the overall improvement in overall processing times.

The movement of more analysis and optimization tools into the tightly-coupled cluster continues as new design issues emerge. For large complex chips placement alone is not sufficient to accurately predict performance. Wire routing information is needed to determine path lengths more precisely and to identify potential noise problems. Thus we now see Incremental Global Routing joining the cluster with Incremental Noise

Analysis close behind and manufacturability in the wings.

New Design System Architecture



It was evident that a new design system architecture was emerging. Advancing technology and more complex designs were forcing designers to deal with a ever increasing list of issues. For design systems to keep up, they too would have to deal with these issues and could not afford to treat them sequentially with files separating analysis and optimization applications. We have had a lot of experience developing central data models with supporting repositories and knew the dangers of too much centralization. Most well know algorithms depend on a specific data representation for optimum performance and no one representation is best for all. In the development of IBM's Integrated Data Model (IDM) its designers strived to share the minimum information between applications, while avoiding unnecessary conversions. There are still many cases where an analysis application will copy data from IDM to its internal data representation, perform its analysis and then put the results back in IDM. But, in these cases the benefits of looking at only the changed data and of avoiding file reads and writes overwhelm the cost of data copying. This careful analysis and justification is required in deciding which applications to tightly couple.

Many other applications, such as simulation and test generation access data from IDM, but are not tightly-coupled. They benefit from direct access to the most current design information but do not need to participate in design optimization - yet.

The Need for a Standard Data Model API

No one company can develop all the ideas and tools needed to keep design automation on track to support the continued advancement of silicon technology. To remain competitive, designers must have ready access to the latest advances, no matter what university or company they come from. The movement to tightly-coupled design systems and today's use of proprietary data models greatly compounds the task of connecting and evaluating a new tool. IBM began advocating a standard data model interface in the early 1990's. We supported Sematech's efforts to define CHDStd and have been actively participating in the OpenAccess Coalition. In particular, we have been contributing to the OpenAccess architecture and planning for an OpenAccess interface on IBM's Integrated Data Model.

Engagement with OpenAccess

When Sematech cancelled it's Design Thrust activities in the late 90's, it thankfully recognized the potential significance of the CHDStd effort that it had been supporting and arranged for the transfer of this effort to Silicon Integration Initiative (Si2) [6] in an attempt at keeping this momentum of establishing an industry standard data model alive. Most of the companies that had been supporting CHDStd also understood its potential and agreed to transfer that support as well. At about the same time, Cadence Design Systems recognized the strong customer demand for a standard and made an "open source" offer of its emerging database technology, known as Genesis. This was the birth of what is now known as OpenAccess (OA) and the OpenAccess Coalition (OAC) [7]. IBM, as a significant player in the CHDStd effort, participated in this transfer and has therefore been involved with the OAC since its inception. In fact, it shares the two "joint chief architect" positions with Cadence on the OAC Change Team, the technical body changed with overseeing the evolution of the standard.

Still, IBM recognizes that simple participation in the OAC will not achieve its objective of being able to "plug and play" its internal tools with commercial and university tools. This level of interoperability is required to provide the "best of breed" support of its semiconductor technologies, which is the fundamental reason that IBM has been supporting these activities for so long. Therefore, we have established a two-phase plan for migrating our internal tools towards the industry standard.

The first phase is actively under way and involves building in-memory translators between OA and IDM. There are three objectives for this phase of development:

1. We must determine how well OA supports IBM technologies. It is clear that IDM supports IBM's technologies today and during the implementation of these translators, we can also determine how well OA supports the IDM constructs and, as a by-product, infer how well OA supports the underlying IBM technology modeling requirements.
2. We must understand how well IDM is aligned with OA. This is a pre-requisite to the next phase of the migration. Our experience has shown that the best

way to address this at a detailed level is by implementing such a set of translators.

3. Although the end goal of our migration is to enable tight integration with commercial and university tools, the OA⇔IDM translators will at least allow the launching of the IBM capabilities as "point tools" within an OA based flow.

Depending on what we learn during the first phase of the migration, we will move on to the second phase of the migration which is to implement an IDM "compatibility layer" on top of the OA interfaces. This is also a technique that has been successfully used within IBM in the past for similar purposes, that is to migrate separate collections of tools to a single data model by providing a thin layer of code which dynamically translates the data presented by the underlying interfaces to match the interfaces used by the tools being migrated, on an object by object basis. Obviously, the outright feasibility, as well as the end performance, of providing such a layer will be dependent upon how closely aligned the two sets of interfaces are. The expectation is that, for the most part, this will not be a significant concern. For those cases where it is a concern, we have three options:

1. We can propose OA extensions through the OAC Change Team process to enable features not currently supported by OA.
2. We can use the robust OA extensibility features to extend OA for our own use. (As an aside, these extensibility features may also be useful in combination with the first option to "prototype" proposed extensions before making them a formal part of the standard.)
3. We can actually change the IDM modeling to better align with OA which, of course, may require an en masse migration of our tools.

Regardless, we may continue to maintain the native IDM implementation for a period until running with native OA tools becomes the normal mode of operation.

Opportunities for EDA Companies and Universities

Given the growing adoption of OpenAccess, it makes sense for EDA companies and Universities to develop their tools and prototypes on the OpenAccess API. First, they can take advantage of a highly efficient production data base to jump start their development. The extensibility features of OpenAccess enable experimentation with new data types where needed. Second, when their tools or prototypes become ready, it will much easier to integrate them into a customers environment for early evaluation and feedback. Neither the EDA company nor the customer need to waste time developing translators that also decrease effectiveness.

Finally, given the availability of source code for the reference implementation of the API, means that the users of Open Access have much more control of their product development or research direction.

Summary

Design systems have evolved over the years from batch processing systems to highly integrated suits of tools sharing design data in memory, all to support high-performance chip designs. Today, the many vendor-specific proprietary data models force a user to purchase such a tightly-integrated system from a single vendor and they make it difficult for small EDA companies to enter the market. The industry needs a standard API for design data to enable choice in purchasing tools, to promote innovation and to accelerate the transfer of innovative research into production use. The requirement is more than a "nice to have", it is essential for designers to keep pace with technology and will only be met with strong user support. The OpenAccess coalition provides a forum for developing such a standard to meet the needs of chip designers and to enable innovative tool development by EDA companies and Universities for the future.

References

[1] J.A. Darringer, John Darringer, Evan Davidson, David Hathaway, Bernd Koenemann, Mark Lavin, Joseph Morrell, Khalid Rahmat, Wolfgang Roesner, Erich Schanzenbach, Gustavo Tellez, Louise Trevillyan, "EDA in IBM: Past, Present and Future", IEEE Trans. On CAD, Vol. 19, No. 12, Dec. 2000, pp. 1476-1497.

[2] P.W. Case, H.H. Graff, M. Kloomak, "The Recording Checking and Printing of Logic Diagrams", Proceedings of the Eastern Joint Computer Conference, Philadelphia, PA, 1958, pp. 108-118.

[3] P.W. Case, M. Correia, W. Gianopulos, W. R. Heller, H. Ofek, T. C. Raymond, R. L. Simek, C. B. Stieglitz, "Design Automation in IBM", IBM Journal of Research and Development, Vol 25, No5, Spt. 1981, pp 631-646.

[4] R.P. Abato, A.D. Drumm, and D.J. Hathaway, L.P.P.P. van Ginneken, "Incremental Timing Analysis", U.S. patent 5,508,937, 16 April, 1996.

[5] Hojat, Villarrubia, "An integrated synthesis and placement approach for timing closure of PowerPC microprocessors", International Conference on Computer Design, 1997.

[6] si2.org

[7] si2.orb/openaccess

Facilitating EDA Flow Interoperability with the OpenAccess Design Database

Mark Bales

Cadence Design Systems, San Jose, CA, USA

Abstract

Building multi-vendor interoperable EDA design flows today is difficult if not impossible. Historically, EDA companies have designed their systems to keep end users captive wherever possible. This has the long-term effect of reducing competition and innovation in the EDA industry, to the detriment of the EDA user community, the EDA developers, and their companies. It has kept the size of the overall EDA market smaller than it should be, and has led to the understanding that for each dollar spent on EDA tools and services, over three dollars is spent by EDA customers in integrating the tools from multiple vendors into a workable design flow [1]. If the EDA industry could do a better job at making this simpler, we could provide better products and services to our end customers that would lower their overall EDA bill, while making more money for our companies in the process.

This paper describes the requirements of interoperable design flows from both the user perspective and the application-developer perspective. It lists the dangers that can prevent a flow from working, even if all of the right foundational components are present. Next, the OpenAccess design database work is introduced, and emphasis is placed on its use in fulfilling the interoperability requirements that have been presented. The future work for the OpenAccess group over the next year is described, and the conclusion that an open shared database like OpenAccess is a necessary piece of technology to realize the dream of a multi-vendor fully-interoperable EDA design flow.

1 User Requirements of Interoperable Flows

The process of IC design turns an idea or concept into silicon. Throughout the design and implementation there are many different representations for the design data. It might start at an architectural or behavioral level and proceed through verification and synthesis. Physical implementation can include custom design, automatic placement and routing, and many different analysis and verification tools, ranging from parasitic extraction and physical verification to timing analysis and circuit simulation. Mask preparation can include OPC checks, PSM creation, and the data modifications done to GDSII-level data to ready it for mask creation. Ties to manufacturing and testing equipment carry along the design intent to allow more intelligent testing of the ICs during their construction. The type of the design may be pure digital (D), analog (A), analog/digital with a small analog content (D/a), or mixed-signal with a digital focus (D/A) or an analog focus (A/D). It is almost certain that multiple vendors' tools are used to create a design flow, and it is possible that the end customer has some significant tools that are a critical part of the flow. As we go from 180nm to 130nm to 90nm process nodes, the need for manufacturing-based knowledge at earlier stages of the design process grows. All of these dimensions combine to make it increasingly difficult to create the wide-ranging fully-integrated design flows needed for 130nm and smaller design. The following paragraphs present the challenges faced by the user in constructing interoperable flows, and details them in their approximate order of importance.

The primary requirement from the user point of view for an interoperable flow is a consistent representation of the design data from the start of the design through the end. Regardless of how many tools, databases, or interchange formats are used in the flow, if data is lost during a step, and then needed at a later step, there will be problems (Figure 1 (A)).

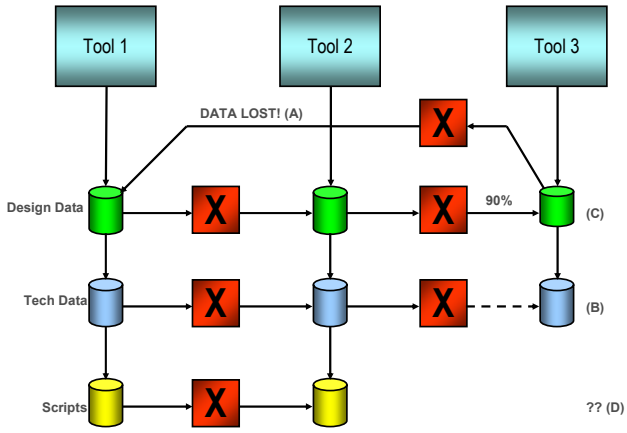


Figure 1

It might be needed to be able to back-annotate derived information to an earlier form (for example, parasitic-derived delays back to the original HDL netlist). It also might be needed to incorporate an ECO from earlier in the design flow. In these cases, the more information we keep about the design as it progresses through implementation, the better job we can do at providing an

incremental ECO that perturbs the physical implementation as little as possible.

In addition to the design data, technology information must be carried along throughout the design process. This can range from fundamental process and geometric rules through complex electrical information used to guide analysis tools (for example, parasitic analysis). There are even more problems in this area than with the design data since many tools have different ways of looking at what is essentially the same technology data. If not accounted for, this can become painfully visible to the user, who must often enter data multiple times, in multiple formats, and is given the responsibility to manage this complexity with little or no assistance (Figure 1 (B)). Add to this the fact that as the data is transformed during the design and implementation process, the required technology information is also transformed in terms of complexity (accuracy) and form. Taken together, these issues highlight the need for consistent interpretation and use of technology information throughout a flow.

Related to the technology information but stored along with the design data are the constraints and assertions that capture the external requirements for the design. As the design progresses these are translated down to lower-level blocks within the design and abstracted to have meaning for specific tools and at specific stages within the flow. For example, an original timing constraint could become a signal-integrity constraint between two adjacent signals that would ultimately become a separation constraint between the routing that implements the two signals. In most systems today the linkage between the levels of abstraction is lost, leaving the management of constraints and assertions up to the end user. Also, inconsistent (or even non-existent) transformation of the constraints can result in them being ignored or misinterpreted by tools within the flow. This can lead to users being required to enter constraint information manually in multiple forms at different stages within the flow (Figure 1 (C)).

Having a set of tools that have a consistent set of capabilities and capacity has a great effect on the ability to create an interoperable flow. For example, if every tool except one in a flow can deal with blocks having a rectilinear boundary, it can make it difficult or impossible to construct a working flow for designs using rectilinear blocks. Also, if one of the tools in the flow is much less efficient in its use of memory than the others, its reduced capacity will make it difficult or impossible to build a flow that will work for practical sizes of design. It can be necessary to resort to 64-bit machines with large amounts of physical memory, and the resulting increase in hardware costs can make the entire flow less feasible economically.

Stability is of paramount importance when building a design flow. When building a multi-vendor flow in today's environment, the use of interchange formats and proprietary databases is necessary. Unfortunately, most vendors' interchange format readers and writers are probably the least stable parts of their systems, both from the standpoint of code reliability and performance as well as in inconsistent mapping between the interchange format and the internal system. Since many of these interchange formats either treat associated technology information in an ad hoc way or through separate side files, the robustness of transforming technology data through multi-vendor flows is even lower than that of design data.

A requirement that is somewhat less tangible involves the increasing use of scripting and extension languages in EDA tools. Tcl/tk seems to be the most popular such language, although Perl, Python, and many proprietary languages are also used. Even when a single language is used throughout the tools found in a given flow, a multi-vendor flow is likely to have different embedded command structures and forms in each tool. This results in scripts that are difficult to maintain. Since scripts often carry part of the design or constraint data during implementation, building a robust flow may require translation of scripts in the flow (Figure 1 (D)). This almost certainly is left as an exercise for the end user or (if they're lucky), their EDA department.

Complicating this already complex situation even further is the need to continually evolve to meet changing technology requirements. Most EDA users set up their flows for a given process node, and 10 years ago, this would suffice for 2 years or even longer. Today, we're seeing new process nodes every 12-18 months, and changes within the context of the same node point may require modification of a flow, and are coming at 6-month intervals.

One way of dealing with the technology-driven change, and a general capability that is needed to help make use of legacy tools, is a consistent extension mechanism. If a tool within the flow doesn't have the ability to contain some aspect of design, technology, or constraint information that is needed in a downstream tool, an extension mechanism can be used to store the data for later use. Similarly, if technology advances impose new requirements that are not yet generally available, the extension mechanisms can be used to store the new data and carry it throughout the flow to the few tools that need and use it. One problem that can occur in either of these cases is that if the stored extensions are related to the design data, they can become inconsistent as the design data is changed. This can be difficult to track or even detect, especially if the tool doing the design data

changes is the one that has not even a conceptual knowledge of the extension.

A last point worth mentioning is a consistent look and feel in the tools within the flow. If simple user-interface items such as selection, key binding, menu picks, dialog interactions, etc. are very different from tool to tool within a flow, this will reduce the overall efficiency and hence productivity within the flow. Fortunately, with the ubiquitous position that enterprise PC software occupies in most companies today, more and more EDA tools operating paradigms are modeled after the PC tools, and this provides an unintended but very welcome consistency.

2 Application-developer Requirements

After looking at requirements from a user perspective, it is useful to look at what the requirements would be from an application-developer's point of view. We can look at data interoperability levels ranging from file or interchange format through a common in-memory model, and discuss which mechanism is appropriate in which circumstances. In addition we can look at the control-interface level for components within a part of an interoperable system. Given an underlying framework that supports the interoperability required at the data level, we'll discuss what is necessary at the algorithmic and application levels to insure interoperability.

File-level interoperability is the simplest kind (Figure 2).

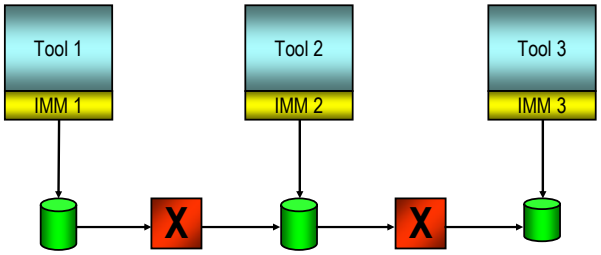


Figure 2

It has the benefit that it decouples the tools involved from each other and probably from the exact version of the interchange format used. It provides a readable format (in most cases) that can be used for purposes of debugging and testing. Using files presents many challenges and issues, however. The time needed to parse a readable format is usually much greater than that required to read a specific database file. The size of the readable files is usually much greater than the design data in a database format. Interchange formats act as "lossy filters" when compared to design databases. If a format doesn't contain a particular type of information, then a "round trip" through that format will lose possibly critical information. File-level interchange is most useful for applications that create a different form of output and are not responsible for round-trip fidelity of the data.

Using a database, but with a different in-memory model, treats it almost as an interchange format, but without several of the problems (Figure 3).

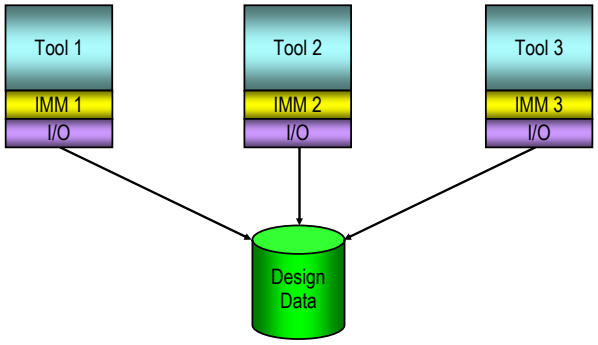


Figure 3

It shares the file-format advantage of allowing the tools to remain uncoupled. It does not directly provide a readable version of the data, but databases often provide a readable archiving or versioning format that can be used if need arises. Databases mitigate almost all of the problems found in file-based formats. They require little or no parsing, and are hence much faster to read. Applications only need be concerned with the relevant data and the database system manages the rest of the data. In this sense they aren't "lossy" like the file-based formats. Round-trip fidelity is achieved by updating the relevant information in the database system. The problems unique to database use arise when significant mapping must be done between the database information model and the in-memory model used by the application. This can make opening and saving a design too slow, and can even cause problems in the fidelity of the data mapping in the worst case. In addition, it is very difficult to share components among applications that use the same database but have different in-memory models. If this is a requirement you're much better off developing a common in-memory model.

Once you have a common in-memory model, you have the best of the worlds (**Error! Reference source not found.**).

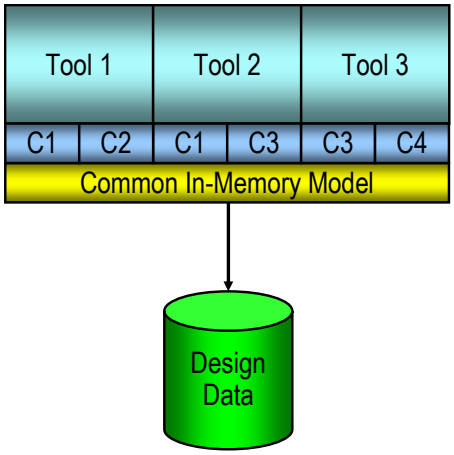


Figure 4

If you take the additional step of making the system components incremental and based upon the core in-memory model wherever possible, then you can collect an entire set of application components; technology and algorithmic engines that can be combined and reused in many different products. This lends consistency to the different tool suites within a design flow; reduces development and maintenance costs over time; makes for a modular system that can remain vital by easily upgrading modules one at a time; and provides a rich base for more rapid development of new products and technologies.

Some would think (with more than a bit of justification) that having a common in-memory model can slow progress. It is true that any sweeping change must be well planned and deployed. It is less true that additions must be well planned; they often benefit from a developmental deployment where an initial design is tested with actual products and refined before being more broadly deployed. The need for rapid progress can be met through use of the extension mechanisms mentioned in the previous section. This allows rapid prototyping, the ability to keep proprietary extensions private, and makes sure that the progress of the standard doesn't limit the ability of the tools to keep up with changes in technology.

Even though the common in-memory model provides the greatest gains, it is important to choose an integration method that is well matched to the task at hand. We continue to dismiss file-based formats for the problems previously mentioned. However, using a database as an exchange format (see Figure 3) is appropriate if:

- The subsystem isn't incremental.
- The I/O time as a fraction of the total run time is small.
- The components within the subsystem are built on a different in-memory model and switching would be expensive, or would cause a degradation in performance.

It is possible to build a highly productive design flow that uses tools plugged into a common database as a backplane that interfaces each tool in the flow in the most appropriate manner (Figure 5). When done in a careful and elegant way, the integration seems completely seamless.

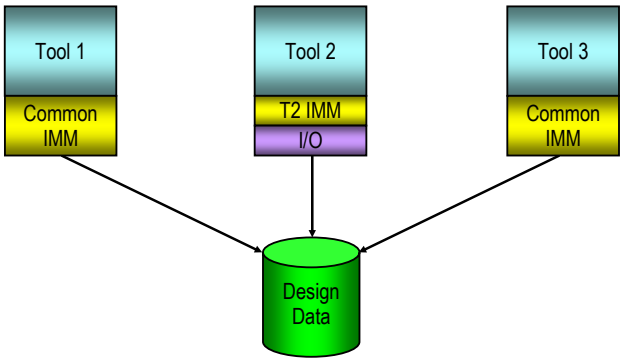


Figure 5

One advantage the common in-memory model gives over the use of the database as an interchange format is the ability to reuse system components. The biggest impact this makes on the overall design flow is the level of consistency and convergence it brings. Timing analysis gives the same answers right after synthesis (modulo a parasitic effect or two) as it does after physical implementation and extraction, *even if the analysis is done in different tools.* An additional capability not yet mentioned that makes it easier to share components is a set of common control interfaces for the components. If you standardize on the way a timing engine works it's easier to reuse it and easier to replace it when necessary.

3 Dangers affecting Interoperability

What factors can affect interoperability? Well, an incomplete database specification can keep different tools from interpreting the data correctly and consistently. It is possible that API implementations might not have enough error checking to detect cases of inconsistent interpretation. Dependence upon idiosyncratic behavior of the database system can cause some applications to fail when they are combined on a single database reference implementation, or the same application code may run

well on one database implementation and fail to run on another. Overuse of localized extensions can lead to data that is no longer interoperable. This is especially true if any of the core data is being replaced. If the database system contains rapidly-changing APIs and/or semantics of the underlying data model, it can be difficult to maintain a functioning flow, especially in a multi-vendor environment. You would need to synchronize the multiple vendors' use of the design database, and this might be impossible if the development cycles of the groups contributing to the flow are widely different.

Lastly, it is important to recognize that an interoperable framework enables the construction of interoperable flows, but doesn't guarantee the end result. It is still easy to build flows that require re-entry of constraint and technology information, have inconsistent interpretation of design data semantics, use incompatible extensions that should be shared, and so forth. Application groups have to work very hard to create interoperable design flows.

4 The OpenAccess Design Database

Given the importance of a common design database in the EDA industry, the OpenAccess Coalition has been formed [2] to develop, deploy, and support an open-sourced EDA design database with shared control. The data model presented in the OA DB provides a unified model that currently extends from structural RTL through GDSII-level mask data. It provides a rich enough capability to support digital, analog, and mixed-signal design data. It provides technology data that can express foundry process design rules through 130nm, contains the definitions of the layers and purposes used in the design, definitions of VIAs and routing rules, definitions of operating points used for analysis, and so on.

The details of the data model and the OA DB implementation have been described previously [3]. This paper will concentrate on considerations of how the OpenAccess design database promotes interoperability and fulfils the requirements already presented.

5 Facilitating Interoperability with OpenAccess

OpenAccess has several fundamental features that make it easy for applications to write interoperable code. The OA extension capabilities allow applications to attach their own data structures to the database data structures, enabling the construction of incremental algorithms that communicate through the database. These same extensions in their persistent form allow applications to rapidly prototype new constructs, and to accelerate the pace of evolution of the system. Using extensions that may even be proprietary (we are each in business, after all), provides a legal way to divert from the standard while maintaining full compatibility and interoperability with other applications for the core of the standard database. Basing applications more directly on the OA in-memory model leads toward consistent tool capacity. Applications must actively drive towards consistent use of the db capabilities to insure workable tool flows. The stability and guaranteed migration of both the OA APIs and the design data created by OA insure that investments in application development are preserved over time. This also helps product developers assemble flows that use application components that might not all be using the same level of OA db APIs (**Error! Reference source not found.**).

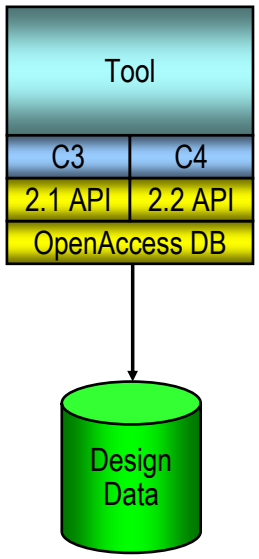


Figure 6

Lastly, extension language hooks for parameterized cells, user-extension management, and extension-language bindings are available and being used to foster interoperability through the availability of a common system language.

Non-persistent OA extensions can be used by applications to tie their application data structures to the corresponding database objects. Doing this gives a very high level of performance (but to be honest can never be as fast as a custom-built implementation created for the specific application). It also reduces the amount of memory required, as it isn't necessary to duplicate storage of data that is already in the database. The biggest advantage comes when the OA callbacks are used to manage application data structure changes in reaction to a database change. In this manner, multiple subsystems within a single process can interact and coexist entirely through interactions with the database. They never need to be directly cognizant of the other subsystem's presence.

OA extensions can be used in either a persistent or non-persistent fashion for prototyping. The OpenAccess

community is comprised of EDA professionals with vast areas of knowledge and expertise. Each of us is interested in different advances that could be made with OA. The extension mechanisms provide a means for prototyping new features. They are almost identical in speed and memory performance to "native" features. This means that meaningful prototype information can be generated to not only demonstrate the functionality of a proposed feature, but also its expected memory usage and speed. This is a very powerful capability that should both accelerate the rate of progress within the OA community as well as providing the "escape valve" needed by most developers to make up for the direct control they trade away when choosing to use OpenAccess.

OA extensions can be used to provide a "legal" way of keeping some information proprietary. In the past, this desire to have advanced features and information in an EDA database has led almost every EDA company to design their own design database. As end users and their integration groups try to assemble working flows from a collection of different data models, databases, and interchange formats, this can become almost impossible. When using OpenAccess extensions, the core database is unchanged and is still fully interoperable with tools from any manufacturer. Only the extensions are not visible. This lets the OA user concentrate on building db models only for their extensions and lets them be as efficient as possible.

Building interoperable design flows using OpenAccess requires that applications do several things during their implementation. They must first agree to base their in-memory model using OA at its core. By avoiding duplication where possible and appropriate, we can keep a higher capacity for the applications, improve the performance by not requiring input and output phases for an application, and make sure that multiple subsystems cooperating in the same process don't have data synchronization troubles. The second thing applications can do is to be designed to work in an incremental way. In this way, small changes made by one subsystem won't require a large change or reanalysis by a cooperating subsystem. A final recommendation for applications is to use the database in a "legal" and consistent way. Sometimes (especially legacy) applications have a different model for some construct than is found in OA. Many application developers aren't sure where to put a piece of information, and decide to put it in whatever field appears "close" to what they need. In this case, "close" won't cut it, since the semantic meaning must be uniform throughout a flow or it won't work. The application should take the time to fully understand the usage of the system, and use the legal extension mechanisms to take care of anything that is necessary but that doesn't fit within the OA model.

OpenAccess guarantees wherever practical that the APIs provided will be upwards compatible from one release to the next. This means that while we are adding *new* functionality on an ongoing and rapid basis, we do not *change existing* APIs without a good reason. Required changes almost always follow changes in technology. As an example, between version 2.0 and version 2.1 of OpenAccess, support for TrueType fonts has been added, requiring a change to the *oaText* APIs to support the new capability. We try to keep these changes to a minimum, however.

In OpenAccess we always guarantee to be able to bring the data forward from one database version to the next. Major versions are released about every six months (at the current time), and are defined by virtue of their direct data incompatibility. Translators are provided to go from an older version to a newer one. When making such a translation in the context of a design flow, it's important to recognize that it is not possible to go back from a newer version to an older one. This is because there is almost always more data added from OA release to release. If we provided the means to go to an older version, it is likely that information would be lost. Due to the "drop-in" capability of OA (and the ability to recompile for new major versions), it should almost always be possible to upgrade a design flow to the newest version used by the tools in the flow and still keep it working, even if there is a mixture of old and new tools.

In OpenAccess, it is possible to tie in a scripting or extension language, such as Tcl/tk, Python, Perl, or Java. These languages (as well as the core OA language of C++) find use as implementation languages for parameterized (programmable) cells; as languages to guide and manage user extensions; and as command-level extension languages for a system built around OA. Parameterized cells (usually abbreviated to *pcell*) are a construct that has existed in the EDA industry for almost 20 years now. They are cells whose contents are not defined by a static netlist or layout but by a *program* that takes a parameter set that can be different for every instance of a given pcell. VLSI Logic, Cadence Design Systems, Mentor Graphics, Avant! (now part of Synopsys) and many other companies have long had these types of cells. The benefit within OpenAccess is that there is no requirement for a specific language so that the end users and application developers are free to add whatever language is best suited for or best liked by a particular company or group. The implementation within OA allows library data to be distributed containing the parameterized cells along with an encapsulation of the language subsystem. This data can be sent to another application that has no direct knowledge of the pcell language, and the whole system just works. A similar capability is available for maintenance of user-extension data, although this is typically handled in C++ rather than

an interpreted extension language, for performance reasons. The same capability can be used as a command-level extension/scripting language for a system built using OA. In this case, a binding of the OA API into the extension language makes possible scripts that modify the database. The language subsystem provides some of the most powerful and system-useful features within OA.

6 Future Work

The work of the OpenAccess ChangeTeam is documented in a 3-5-year Roadmap that is developed on a yearly basis [4]. The major items on the near-term roadmap are Embedded Module Hierarchy, Constraints and Assertions (Timing support), and expanding down into the Universal Data Model level.

Embedded Module Hierarchy is a way to take a snapshot of a design hierarchy at a given point in time and be able to look at the data from three different perspectives; the Module domain, the Occurrence domain, and the Block domain. The Module domain (think Verilog Modules) represents a logical hierarchy. It is folded, and contains all of the levels of an original netlist. The Occurrence domain is an unfolded version of the Module domain. It is fully logical and contains no physical information (Figure 7).

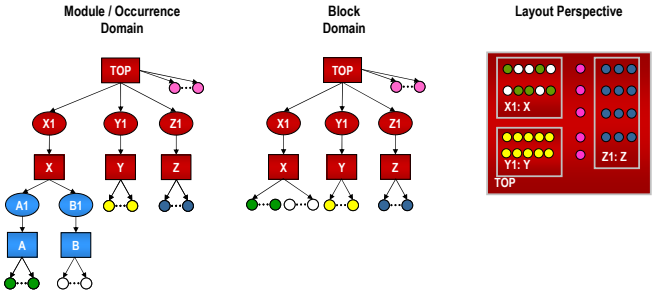


Figure 7

As changes are made in the Module domain, they are reflected forward into the Occurrence domain. When changes are made directly in the Occurrence domain, this causes uniquification of the Module occurrence and the changes are reflected backwards into the Module domain. The Block domain is folded, and most often contains the shape and connectivity information for a physical view. The three domains represent a single data hierarchy with three different ways of looking at the data. The "Embedded" term comes from the fact that some of the Module nodes are also Block nodes. The benefit of EMH is that it allows physically-oriented tools to look at multiple levels of logical hierarchy as though they had been flattened, without losing the information about the logical levels (including intermediate nets and terminals).

Constraints and assertions are a wide-ranging topic, covering items that are timing, electrical, and/or physical in nature. Work is proceeding to expand the capabilities of the OA Technology Database to represent constraints and assertions for timing and signal integrity. We're in the process of adding physical (foundry) rules for 90nm and 65nm process nodes. We expect to ultimately add support for timing models (.lib and/or ALF support, OLA support). The current state can support physical technology rules for 130nm. The goal is to be able to support RTL-to-Silicon technology rules for 65nm. This will be an ongoing effort with continuous improvement in each of the OA DB releases over the next several years.

Another area of great interest to the OpenAccess Coalition is the work taking place in the semiconductor mask-making and equipment areas called the *Universal Data Model*, or *UDM* (Figure 8 [5]).

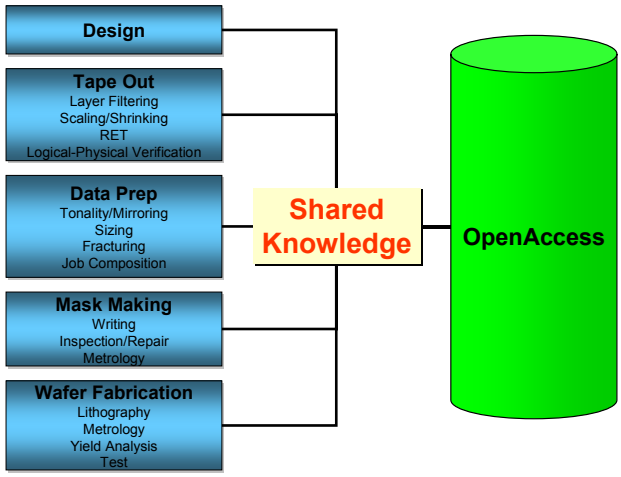


Figure 8 [5]

While OpenAccess currently supports IC design from RTL through GDSII levels, it stops short of supporting tasks from tape-out through wafer fabrication. Support will be added for tape-out tasks such as Layer Filtering, Scaling and Shrinking, and Reticle Enhancement Technology. Support for data preparation tasks such as Tonality and Mirroring, Sizing, Fracturing, and Job Composition will be added. Support for mask-making tasks such as Writing, Inspection and Repair, and Metrology will be added. Finally, support for wafer-fabrication tasks like Lithography, Yield Analysis, and Test will be added [5]. This is an ongoing topic, and the advances in this area will be evolutionary and will take place in each of the OA DB releases over the next several years. We will incorporate available standards wherever possible, such as the OASIS (or NSF) work, the SEMI P10 standard, and of course, the SEMI UDM work itself [6].

7 Conclusions

The requirements for interoperability take two main forms: framework capabilities and technology, and application-level resolve. OpenAccess provides the interoperability at the framework levels, and sets the stage for construction of highly-interoperable flows that use a variety of models. The models range from using the database as an interchange format, to using the database as an in-memory model. A greater level of integration with OpenAccess allows greater modularization and reuse of application level components in products, but is not required or even appropriate for each application within a flow. Lastly, even use of a common in-memory model is not a guarantee of interoperability. Applications must take continual care to make sure they have the correct semantic interpretation of the data and that they use the data in a way that enables use of the greatest number of downstream tools. If we make good use of infrastructure technology and keep clear application discipline, we should finally be able to build fully-interoperable multi-vendor flows.

References

[1] S. Schulz, "Open Access 2003 Conference Keynote", OpenAccess 2003 Conference, February 2003

[2] D. Cottrell and A. Graham, "What is OpenAccess?", http://www.si2.org/openaccess, January 2003

[3] J. Santos, "OpenAccess Architecture and Design Philosophy", OpenAccess 2002 Conference, April 2002

[4] OpenAccess Change Team, "OACT 2003 Roadmap", http://www.openeda.org/openaccess, April 2003

[5] D. Cottrell, "UDM Business Case", February 2003

[6] T. Grebinski, "SEMI IC Design/Photomask Data Path Task Force", International Sematech MASC Strategy Meeting, February 2003

Session 4:

New Automated Methodologies and Flows I

Moderator: Naresh Sehgal (*Intel*)

Hard IP Group Design

Howard Sachs, Michael Barry, John Campbell
Telairity Semiconductor, Inc.
3375 Scott Blvd. Suite 300
Santa Clara, CA 95054
Email: hsachs@telairity.com

Abstract

Today there is significant evidence in the marketplace that there is a surprising 2x loss in performance for most new ASIC designs using normal static CMOS logic. Many have attributed this surprise to non-scaling of interconnects in the DSM range, causing signal integrity issues. But the technology has been scaling which indicates the problem lies somewhere else. The reason for this loss is that the average wire length is getting longer as the number of devices on the chip increases. A new approach will be described that uses generic hard IP blocks to design any type of digital system that overcomes these losses to achieve a 2x improvement in performance over the current methods.

1. Introduction

Today most ASIC designs are very large, in the range of 500K to 20M gates. This level of complexity has caused individuals on the design teams to specialize in specific aspects of the design process. As time has progressed the notion of "The Tall Thin Engineer" has almost disappeared. Instead we now have specialists in Logic Design, Simulation/Verification, Pre Synthesis Floor Planning, and Physical Design. As a result the RTL Designer or Logic Designer really does "Logical Design" since he/she is not usually involved with the physical part of the design process. The expectation is that the RTL is independent of any physical parameters and that synthesis and backend tools will make the appropriate transformations to the logical design to create a good physical implementation. In fact there is a virtual wall between the logical design and physical design.

How important is the understanding of the technology to the actual logical design of a system? Looking at history may provide some insight.

From 1960 to 1980 the understanding of the physical implementation was considered very critical to the design process. Physical partitioning of the design was a key step in the design process. This was an exercise requiring an understanding of the critical paths and the physical distances between the partitions because the interconnect wiring was a significant contribution to the overall delay. This physical partitioning usually became the same as the logical partitioning and specifications were written that described the timing requirements for each partition. Today this is equivalent to what is now called the floor plan.

From 1980 to 1990 CMOS was introduced and became the choice for most designs. Density, adequate speed, high yields, and the resulting low cost made this the technology of choice. ASSP designs were usually implemented with custom gates or standard cells. ASIC designs were implemented with Gate Arrays. Typically the Gate Array designs were much slower than the Cell Based designs because of the non-optimal nature of the building blocks, long wiring channels and the conservatism built into the models for yield purposes. The Standard Cell based designs were much smaller because of floor planning was used. And as a result the designs were significantly faster because the wiring was significantly shorter and the drivers were matched to the interconnect wiring and loads. During this same period Verilog became the de facto standard for net list design and was used for both Gate Arrays and Standard Cell based designs. Logic synthesis was introduced during this period with the ability to create up to 50K gates; later on this number was increased to 200K gates. This made a significant improvement in productivity for both types of designs, typically a factor of two or three, and with very little to no downside.

During the period from 1990 to 2000 CMOS speed continued to improve by 50% and density continued to improve according to Moore's law at 2x every generation. By the end of the decade Gate Arrays were almost totally replaced with Standard Cell based designs in Europe and the United States because of the poor performance and high cost of the Gate Arrays. Wiring length was beginning to be more important but gate delays were still dominant because of the over designed Gate Arrays. The synthesis and backend tools produced adequate results for Standard Cells compared with Gate Arrays. Crosstalk was present in the standard cell designs but not noticed because the performance was so much better than Gate Arrays.

Starting in the year 2000 with 0.18um technologies new problems emerged. When we moved from 0.25um Standard Cells to 0.18um we expected a 50% increase in performance, which we did not get because of signal integrity issues. The actual wire lengths increased over the expected scaling because designers were synthesizing much larger blocks than before. When larger blocks are synthesized the amount of floorplaning done is reduced, causing average wire lengths to increase. These larger blocks also make the work of the placement tool more difficult, resulting in longer average interconnect wires as well. Longer wires also allow the crosstalk effect to negatively influence the speed of the circuit, causing surprises. Also, proper matching of transistor drive strengths to the loads (wire and gate loads) have not been very accurate. Both the longer wires, a result of less floorplaning, and inadequate transistor sizing have caused most designs to lose at least a factor of 2 in performance.

Floorplaning and synthesizing the design into smaller pieces, less than 10,000 gates in each partition, will keep the wires reasonably close to the optimal length, otherwise the wiring could increase by as much as a factor of four longer than optimal.

2. Wires in More Detail

Scaling

Figure 1 shows the RC delay for a fixed 500um length wire in different geometries. This chart shows the familiar continual increase in RC delay shown in many presentations. Note how this chart shows a significant problem when going from 0.13um to 0.09um. Fortunately for ASIC designers, this chart does not tell the true story because the interconnect wires should be scaling at 70% along with the technology.

Figure 1 RC Delay with 500-um Fixed-Length Wires

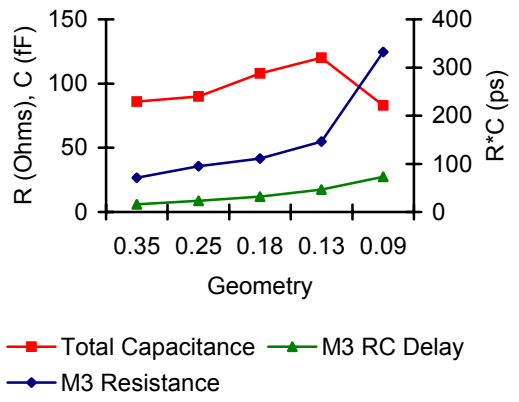


Figure 2 shows the simple RC delay as a function of geometry for a wire that is 500um in 0.35um and scaled at each generation by the metal scaling rules. It is clear that the RC is not getting worse; in fact it has been getting better. All of the figures in this paper assume copper and low k dielectric for geometries at 0.13um and lower. The RC delay may go up slightly in the future but we do not have enough data at this time to accurately show the trend.

Figure 2 RC Delay with 500-um Wires Scaled

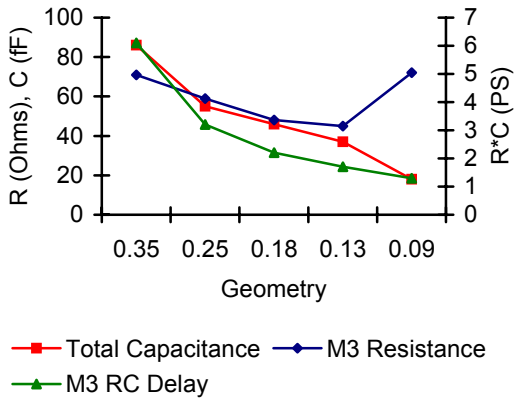
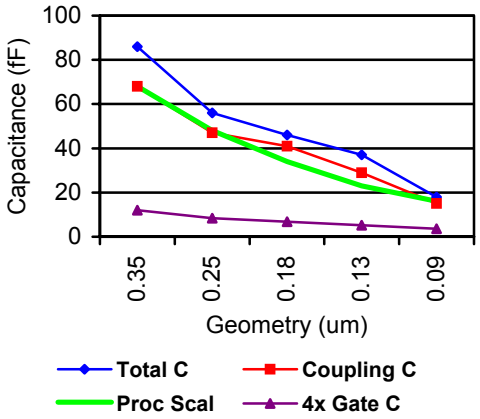


Figure 3 shows that the total wiring capacitance and crosstalk coupling capacitance is scaling as expected. The same design done in a 0.35um technology and a 0.09um technology will scale accordingly. If the wire lengths and capacitances are scaling, then the delays and any effects of crosstalk should be in the same ratio and therefore present no new surprises. Ho [2] also suggests that wire lengths, crosstalk, and wire delays are scaling with the transistors. It is obvious from Figure 3 that if the technology is scaling and yet there are excessive signal delays, then the wires must be longer.

Figure 3 Process Scaling



Wire Lengths

Chang, Cong, & Xie [6] suggest that wire lengths are from 1.6 to 2.5 times longer than optimal because of non optimal placement algorithms. This data was taken from synthetic benchmarks that have known wire lengths. This data also indicates that there is a significant increase (up to 25%) in wire length when the number of modules or gates is increased by ten times.

Horowitz [3] argues that the major concern now, is that we have more long wires to deal with because the designs are larger. The following are summary quotes from his presentation.

"Communication on chip is no longer free"

"Back to the future – it looks like board/box design"

The back to the future comment is emphasizing the need for engineering the wires the way it was done years ago and not to let these wires get out of control. A typical wireload model also shows this expectation quite clearly. A set of typical standard cell wireload models for a standard 0.18u technology is shown in Table 1.

Table 1 Wireload Model for a Fanout of Four

Gates (K)	Capacitance (fF)	Wire Length (um)
10	5.2	26
20	13.1	66
40	17.0	85
80	19.0	95
160	20.0	100

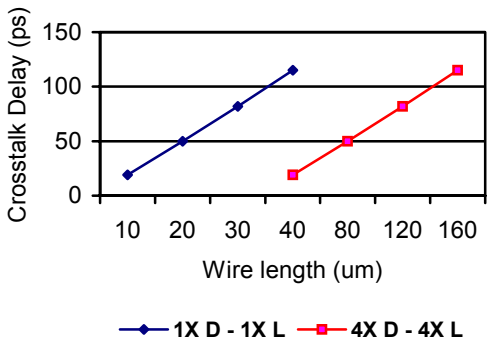
This model suggests that the average wire length increases from 26um to 100um when the number of gates in the same partition to be synthesized grows from 10K gates to 160K gates. Our experience shows that the wire lengths can vary from 5um to over 200um for small designs in the range of 3000 gates. This can result in significant RTL rework of the design after the physical placement and routing indicates this variation in the loading.

Dally & Chang [1] have shown that wire lengths for automated standard cells for a data path design can be much greater than the custom wire lengths. They show metal-2 wire lengths 34.9X longer and metal-3 longer by 7.92X than in a custom design. In this example much of the additional wiring is due to the replacement of the synthesized register files. The seven-port register file in the design was built using ordinary flip-flops and multiplexers, which is quite common, since custom register files are expensive to design or purchase. This additional wire length can cause additional rise time delays and crosstalk delays. Of course, if the cell library has sufficient control of the drive strengths, delays of the longer wires can be overcome by increasing the drive strengths of the gates.

Gate Sizing and Drive Strengths

One solution to longer wires is to match the drive strength of the driver to the wire load and gate load in the design. Figure 4 shows that a 4X gate will drive a 4X load with 160um of wire (equivalent to a 4X load) at the same speed as a 1X driver with an equivalent wire and gate load with the same total delay. (Note that in this paper a 1x gate is made with minimum size transistors, contrary to most cell libraries, where a 1x is really 4 times the minimum size allowed.) Therefore the delays for a net can be managed to the optimal delay with proper sizing of the driving gate to the loads.

Figure 4 Wire Crosstalk Delay



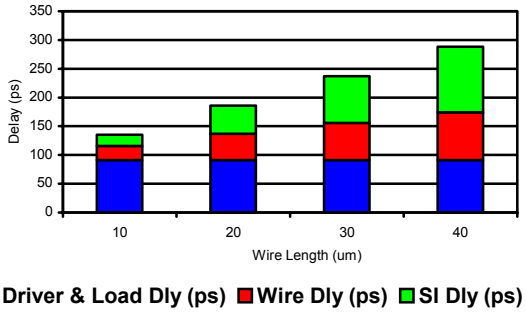
Many standard cell libraries have only a few different gate sizes and corresponding drive strengths, making the speed of the net (the rise and fall time) either too fast or too slow. In addition some standard cell libraries have multiple stages inside the cell to maintain a 1X fan in. This makes life very simple when wire loads change; all that is required is to replace the cell with one that has larger buffers inside. The problem is that the delay is no longer constant because of the intrinsic delay of the two additional inverters. Most designers use small complex gates and inverters with more power to drive large loads. This can optimize the net for speed as well as area. The approach we recommend and have used in our designs have 16 different power levels for inverters to optimally drive the nets. In addition we have proprietary software for automatic sizing of the gates to the loads in our designs.

Crosstalk

Crosstalk has recently become an issue because the wires appear to no longer scale with the technology, but in fact this is a result of poor placement and/or no floorplaning and not the technology. As a result the amount of crosstalk and delay in the net can be increased significantly.

Figure 5 shows for a given driver and receiver the delay is constant. The wire delay, actually the rise time delay, increases, as the wire gets longer because of the added capacitance of the wire. This chart shows the added signal integrity delay due to aggressors switching in the opposite direction from the victim. For each case we simulated using the same 1X driver and receiver for the victim, with a 200-ps rise or fall time. The aggressors are driven with an ideal voltage source with a 200-ps rise and fall time. With longer wires the crosstalk coupling capacitance ratio to the total capacitance increases thereby causing the signal rise time to be further delayed. Increasing the size of the victim's driver can significantly reduce the signal integrity delay, since the equivalent driving resistance is reduced allowing the victim to recover faster. This is illustrated in Figure 4.

Figure 5 Components of Delay



3. A Solution to Improve Performance

The solution is a large number of small reusable IP blocks, called groups that can be used over and over and are kept in hardened form to guarantee the performance, area and power of the IP. The groups are designed with custom slices or standard cells to achieve optimal performance, area and power. In order for this solution to be economically viable these groups must be reusable to a very high degree and port between processes automatically; otherwise the added engineering costs outweigh the advantages. The focus for this solution is as follows:

- Custom-like performance for groups
- Wiring is engineered and minimized early in the design
- Transistor drive strength is optimally engineered
- Global wires are isolated from local group wires

- Automated group portability from Fab-to-Fab and process-to-process

Design re-use is a very alluring methodology because of its inherent simplicity, high productivity, and repeatability. If one looks at cell libraries, the re-use factor is always 100% and is the standard method for assembling logic. Each cell typically has between 1 to 4 gates. The other end of the spectrum is block level re-use with 50,000 to 200,000 gates, but this has a very poor re-use factor, less than 33%. Choosing the proper building block size is critical to this approach.

We have examined many designs and determined there is a flat 95% re-use curve from 4 gates to about 3,000 gates. After that the re-use factor starts to drop off rapidly. This reinforced our notion of a group with an optimal group size from 300 gates up to a maximum of 3,000 gates. The average "Group" will have ~1,000 gates.

We have identified a natural partition of any design into four fundamental types of logic: Data path, Control, I/O, and Memory and have demonstrated that any digital circuit can be built using these building block groups. Characteristics of the groups are described below;

- Control

Control structures are typically PLA-like structures and state machines that contain RAM's or ROM's.

- Data Path

The data path is the implementation of algorithms and is only concerned with how data flows. Typical groups are adders, shifters, multiplier parts, etc.

- I/O

General I/O interfaces such as UARTS, USB, SDRAM interface typically have low implementation or licensing cost and hence are usually soft cores that would not be built out of groups.

- Memory

Memories such as register files, FIFO's, & CAM's are key components of this technology. Moderate size SRAM's for caches are used. DRAM's are not addressed since they are readily available from a number of sources.

Table 2 shows a few examples of different types of groups. It is interesting to note that the range in average wire length for groups vary in length by over a factor of four. The group with the smallest wire lengths, 25um, is a 2 input 32-bit adder (Ad2_32). The group with the longest wires, 121um, is the Perm_80, which is an eighty-bit permuter that is primarily composed of multiplexers. The average wire length does not include the wiring within slices or standard cells. Therefore the average wire length would be somewhat shorter if we included all of the very small interconnect wires. The Ad32_32 agrees quite closely with most small size wire load model tables with a fan out of 4 at 25um. The wide variation in actual average wire length, as shown in table 2, assuming a fan out of 4 demonstrates why assuming an average wire length that is dependent only on the number of gates being driven can lead to significantly over loaded or under loaded nets.

Table 2

Group Name	Ave Length of Wire (um)	Total number of Equivalent gates
Buf_32x20_r2w1	111	2265
Iseq_8	42	2473
Ad2_32	25	873
Adsb3_80	29	3029
Perm_80	121	654
Bth_mux_32	39	2719

We have done an analysis of the critical path on a hand crafted 32-bit adder group with 15 stage delays (flip flop not included). The average wire length for the complete adder is 25um. The average wire length in the critical path however is 43um with a range from 5um to 227um. This variation is so great that a normal wireload model would be dramatically in error on most of the stages causing most of the nets to be either over driven or under driven. The driving stage can vary from a 1x in the 5um case all the way to an 8x in the case of 227um of wire length. Of course, if the gates were sized up or down in each case to optimally drive the wires the delays would be optimal. As described earlier some cell libraries use multiple stages to power up the load driving capability. This makes for easy insertion, no downstream effect, but usually has three stages and the resulting delay is not the minimum. Sutherland [4] describes this problem in detail on page 20 of the reference.

Additional delays can be attributed to crosstalk in the wiring. This added delay due to crosstalk could be as large as 0.7 times the normal delay, using equal rise and fall times of 200ps. The total delay for the 32-bit adder without crosstalk is 1635ps. So the impact of crosstalk could be zero or as small as 10 to 20% to as much as 70% of the stage delay. Figure 5 shows that the heavily loaded gate could be delayed as much as 114 ps because of crosstalk. In the 32-bit adder case if this was the only stage affected there would be a 6.9% overall delay in the path. Note that most clock cycles have 15 stages of logic, and the crosstalk typically only affects one or two stages out of the fifteen. For example, the 32-bit adder referenced above averaged 81ps delay per gate and 28ps average wire length delay. Even doubling the wire delay by crosstalk for four stages of the fifteen would result in a delay, which would be quite small:

$$(4 \times 28ps)/((81ps + 28ps) \times 15) \text{ stages} = 6.8\%$$

So, typically we would expect that large delays due to crosstalk would be primarily concentrated in long global wires where the global wire is a large percentage of the cycle time.

4. Demonstration chip

We have designed a demonstration chip, which includes a SIMD FFT [5] to illustrate the use of our group methodology in a typical design. This circuit was designed to run at 400MHz in a UMC 0.18um standard logic process with worst-case temperature and voltage and nominal

process. The chip is expected to be back from the factory in early March with actual speed results available in mid-March. HSIM [7] was used to determine the speed of the design dynamically using functional vectors. Using the adder reference above with an average delay of 109ps per stage and a total of 15 stages between flip-flops as a typical group, the total path delay would be 1635ps. Adding a flip-flop stage at the output adds another 400 ps for a total of 2035 ps, which is 491 MHz.

Table 3 shows the mean and average wire length for the entire FFT design. The mean length of the M2/M3 wires in the adder group was 15um. The mean global wire length was 125um. The mean wire length is more meaningful since there are always long wires in a design but many of the long wires are not in the critical path. The limiting path in the FFT design was the booth encoder and carry save adder and that path was 2439ps limiting the design to 410MHz – which is 2 to 3x faster than a typical synthesized design.

Table 3 Wire Lengths for the FFT

	Ave wire length (um)	**Mean wire length (um)**
M2/M3	**51**	**15**
M4/M5	**284**	**125**

Table 4 shows the metal utilization for each metal layer in the design. Metal layers 1, 3, and 5 are the most utilized since these wires run in the direction of the data path. The cell library was designed using 14 tracks, which is much wider than most generic standard cell libraries. Most generic libraries are in the 8 to 10 track range to increase the reported cell density. The lack of porosity however causes more pressure on the routing resources. Our intention is to make sure that all of our groups can each be exclusively routed on the first three metal layers, insuring that no global wires would ever be routed through the groups. Metal-4 and metal-5 are global wires and are very under utilized. This means that as the designs get larger there will be adequate routing resources without increasing the wire lengths unnecessarily.

Table 4 Metal utilization for the FFT

	Percent Routing Utilization
Metal 1	**55.0**
Metal 2	**23.6**
Metal 3	**43.4**
Metal 4	**6.4**
Metal 5	**20.1**

5. Conclusion

We have shown that the technology has been scaling quite nicely, contrary to popular belief. Recent ASIC designs have not run at the expected frequencies because the wiring on the chip has not scaled accordingly. These longer-than expected wires are the result of two factors. The first is that designs are getting more complex at each generation, which causes placement tool problems and results in longer wires, and the second is that the methodologies used do not focus on engineering the wires. We have shown a new approach based on pre-designed hard IP building blocks that are optimized for speed that can improve the performance of a typical ASIC chip by 2 to 3 times.

6. Acknowledgements

The authors would like to thank the following engineers for their efforts in providing data for this paper. Richard Dickson, Luigi Di Gregorio, Joe Varghese, Sarita Thakar.

References

[1] W. Dally and A. Chang, "The Role of Custom Design in ASIC Chips", *DAC* 2000

[2] R. Ho, K. Mai, M. Horowitz, "Wires: A users Guide" *SRC/Marco Workshop at CIS, Stanford*, May 1999

[3]M. Horowitz, R. Ho, K. Mai "Wires: A users guide" *PDF Internet,* no date

[4] I. Sutherland, B. Sproul, D. Harris "Logical Effort: Designing fast CMOS Circuits"

[5] S. Arya, "Designing High Speed DSPs" *ISPC* 2003

[6] C. Chang, J. Cong, M. Xie, "Optimality and Scalability Study of Existing Placement Algorithms*", ASP-DAC Conference Japan*

[7] HSIM is a product of Nassda Corporation.

The Arrow of Time:
Following Timing Constraints in an RTL to GDSII Flow

Dwight Hill
Synopsys

The process of transforming a technology independent design representation in VHDL or Verilog into a process specific mask set is commonly referred to as an "*RTL to GDSII*" flow. In a classic flow this involves logic synthesis, followed by placement, then routing, and then buffering as distinct steps, with part or all of the process repeated in a loop until design goals are met. Current practice is to tie these functions together within a single flow. Most of the presentations and articles on this area concentrate on the algorithmic issues in logic synthesis, floorplanning, or placement. But in everyday practice the management of timing constraints forms a parallel flow, which can be almost as demanding and consume as much designer effort as the physical synthesis process.

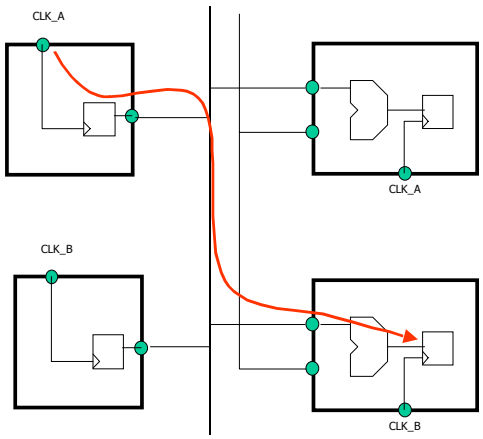
This paper serves as an introduction to the issues in supporting timing constraints through the process. As background, we first review the nature of timing constraints in current design practices: where they originate, how they evolve, and how they are represented. Secondly we consider how they must be manipulated by the tool to fit into a practical floorplan flow. Each of these steps requires its own view of the design. Finally, we summarize the characteristics of any desirable flow.

What are Timing Constraints?

Timing information can be divided into two categories, clock specifications and timing exceptions. Verilog and other RTL languages represent the logical behavior, or logical structure of a design and are often used for timing simulation, but give little information about timing exceptions. (In fact, many physical synthesis flows begin with structural Verilog, which is completely devoid of timing information.) In an ideal world only clock specifications would be needed. There would be only a few clocks, and the relationship among them would be symmetric and clear. In practice, designs with thirty or more distinct clocks (with different periods) plus potentially hundreds of derived clocks are common. Likewise an ideal circuit would have few, or no timing *exceptions*. In practice, designs tend to have a large fraction of their paths (10% is not uncommon) marked as false paths, and a similar but usually smaller number marked as multicycle paths or other exceptions.

Where do these timing exceptions originate? In the best case the set of exceptions would be obvious from the initial design specification. For example many SOC devices have common busses that act as interchange points among independent subsystems. Unless properly handled, these can appear as timing violations when a signal launched by one clock is caught by another unrelated clock. The architect generally knows that this is not a supported operation, and that the path is not exercised in the normal flow of operation. But unless properly informed, the timing engine will generate violation warnings about them. This is the first, and better source of timing exceptions: those generated proactively by system architects using knowledge of the intended system behavior.

A typical path between clock domains is illustrated below:



This path could be disabled in one of several ways, e.g.

```
 set_false_path -from [get_clocks
clk_a] -to [get_clocks clk_b]
```

or

```
 set_false_path -from
reg_a_left.out -to reg_b_right.in
```

Note that the design's intention is probably the inter-clock relationship. If the EDA tool translates this into timing exceptions directly on the registers, such information will be lost.

Unfortunately another source of exceptions is also common. These originate from the process of running timing analysis and finding violations, with a *post-hoc* analysis used to justify that the path in question is not valid. The problem with the second approach is that you can never be sure that you have a complete and correct set of timing exceptions. Each timing analysis run requires a circuit, an estimated or extracted set of loads, and a set of constraints. Early in the design process, the load estimates may have large errors in them, making the design faster on some paths and slower on others than the final tape out will observe. Pessimistic loads are create lots of "false positive" timing violations. Worse than this, even if only a small percentage of loads are optimistic but these happen to occur on true paths, timing analysis will not catch any errors. The result will be of very limited value. While this may seem to be an unlikely event in a "random" circuit, modern physical synthesis technology can actually exasperate the problem. This is because physical synthesis generally transfers delay from tightly constrained paths to unconstrained paths. Thus if a false path is incorrectly left active, it will pull resources (both CPU during the EDA tool run, and silicon area/power) from real paths. And if a true path is left underconstrained it is likely to be slowed down so much as to cause circuit failure in the final chip.

Formats for Timing Constraints

Essentially all timing constraints start out has human generated ASCI text files, usually in SDC (Synopsys Design Constraint) or some similar format. (SDC is a subset of the Tcl language accepted by PrimeTime and other timing tools). Since exception scripts are human generated they tend to be relatively compact and contain clever procedural or compacted notations for multiple constraints. For example, the command

```
 set_false_path -from [get_pins
blocka/out*] -to [get_pins
blockb/in*]
```

would be a common example. More complex example might be

```
 proc set_tc {fromblockname  toblockname}
{
     foreach inp [get_pins -filter
"direction==in &&
        type==signal" -of
$from_blockname]  {
       foreach outp [get_pins -filter
"direction==out" -of $to_blockname]  {
              set_false_path -from $inp -
to $outp;
              }
       }
      }

# now invoke the proc
set_tc blocka blockb
```

The above Tcl proc has the same effect as the original statement (more or less) but is more convenient, especially when multiple blocks must be processed because the "proc" can be reused

and does not depend on the names of the pins (only their direction).

This type of procedural format entered by the user is generally the most compact representation. However, in many flows it is not used directly. Rather, there are several types of transformations done on the constraints before tape out.

Perhaps the simplest is to eliminate the procedural aspect by executing the script. In the example above, the script is loaded and the Tcl proc "*set_tc*" is run, resulting in dozens (or hundreds) of individual false path statements executed. These may be interpreted directly by the timing engine, or stored for further processing.

The second level of processing is eliminating the wild card ("*") in object names. This is not necessarily done in the Tcl interpreter: even tools that are not based on Tcl may support this. The expansion may be done internally, by associating the constraint with many objects internally. After this expansion, the constraints are generally stored from one design object to/or through anther specific design object. The memory required for this representation is generally proportional to the number of objects involved.

In either of the above cases the intent behind the original text representation may be corrupted or be completely lost. That is, the tool may represent the constraints internally but not be able to reconstruct them for the user, except as a flat representation. Thus a few lines of SDC input can be expanded considerably.

Hierarchy Mapping

In addition to the expansion due to the text to in-memory processing, many flows involve moving constraints across the design hierarchy. These are typically at least two hierarchies the designer is concerned with. The "original" logical hierarchy, often corresponding to a behavioral RTL representation, is typically very deep. At the other end of the process, the "final" physical hierarchy corresponding to mask data (or at least to extracted routing) is usually very shallow. Thus the hierarchy takes several forms and each may have impact on the constraints. For example, if constraints are originally expressed using pins from hierarchical blocks but the design calls for those blocks to be flattened the constraints need to find a new "home" or they will be lost. Typically, this calls for algorithms that will propagate the constraints from a block pin down to a leaf cell pin. This mapping is not necessarily one-to-one. In some cases, a hierarchical block pin can propagate down to hundreds of leaf pins.

Another type of hierarchy manipulation occurs when individually designed blocks are combined into one chip, especially during System on Chip (SOC) design. The constraints associated with each block are typically written for the context of the block, often by a team in a remote location. Once the block arrives at headquarters for consolidation, it must fit onto the die. Less obviously, the timing constraints associated with the block must fit into the overall chip's timing scheme.

This process can be complex and time consuming, especially if the blocks were developed independently. One obvious problem is that each block tends to use the names of objects (pins, cells) locally, like "nand17/in2". But when processed at the chip level, they must be prefixed with the block name, e.g. "blockC/nand17/in2". More subtle problems arise from clocks, which are generally global in nature as far as the timing verification is concerned. So clocks defined within one scope may conflict with others. In both cases the most widely used solution is to modify the names of the objects with a block prefix. This tends to increase file size and decrease readability.

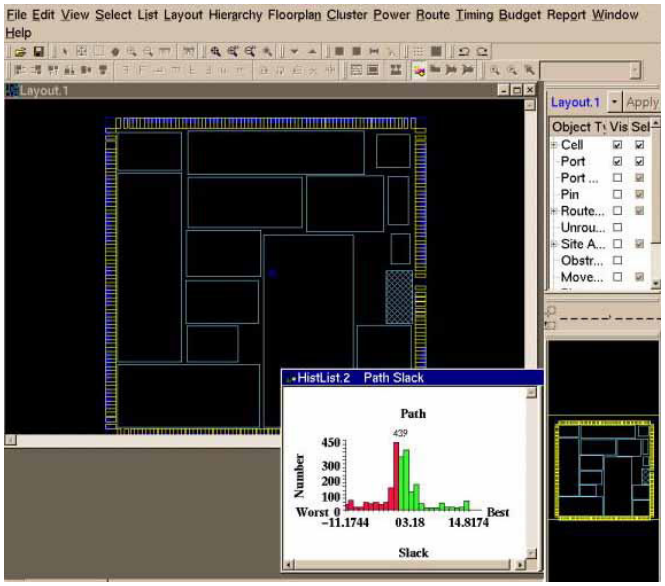
Time Budgeting

After the blocks are assembled and the constraints are made legal there is the issue of making the chip actually meet timing requirements. This can be thought of as a series of operations that identify problems (sometimes potential problems) and assign resources to them. The problems are usually slow paths. They may be due to poorly structured logic within a block, long paths of logic that run through two or more blocks, or paths that involve very long cross chip signaling without adequate buffering. The resources that

can be assigned may be obvious: larger drivers, buffering, clock-tree style synthesis. Or they may be "virtual" in that time may be apportioned away from one block and given to another, in preparation for a complete or partial physical synthesis run. This process is known as *time budgeting*.

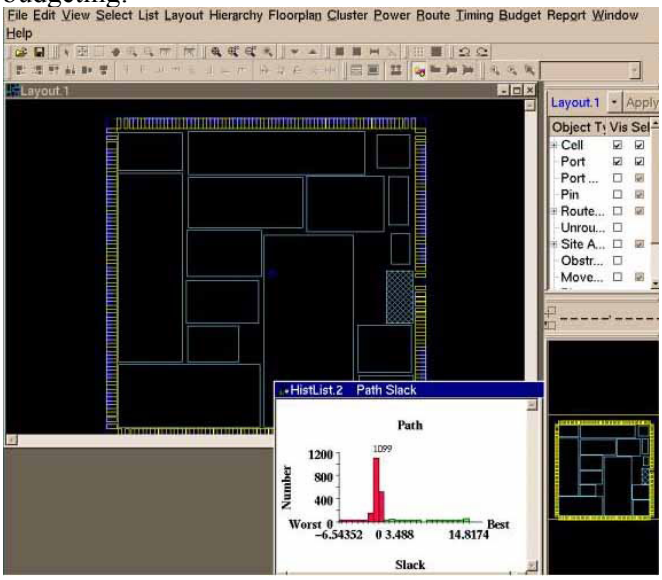
The input to time budgeting is generally a circuit early in the design process with preliminary versions of the logic and estimations of the load and delays among blocks, although time budgeting can be also used later in the process when the structure and delays are more precisely known. Time budgeting starts with the timing it is given and attempts to solve problems by "spreading the pain" out across the blocks. Thus if a path has a negative slack, the delays associated with blocks that it traverses will be reduced in the budgeting model, according to heuristics that attempt to parallel the potential for speed up in the final circuit. Similarly, paths with positive slack have their elements slowed down. Eventually the model reaches a point with zero slack and the complex process of generating timing constraints for each block can begin.

The figure below shows a typical design before time budgeting:



In this case, you can see the distribution of slacks, both positive and negative.

The figure below shows the same design after budgeting:

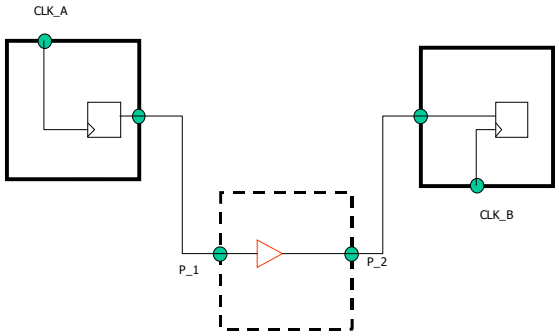


In general, time budgeting will attempt to eliminate all positive slack, but may chose to leave some small negative slack on selected paths. This is because over constraining physical synthesis may result in somewhat excessive run time, but under-constraining physical synthesis will result in the chip failing to meet timing after re-assembly.

Transforming Chip-level Constraints to Block-Level Constraints

Modeling the context of each block involves several steps and can be as complex as the overall timing analysis of the chip. The timing context of each block is a function of the surrounding logic, which can involve the entire chip. The set of clocks for each block includes not just those directly by the block but the set of clocks used by paths that enter, or even just pass through the block. Input and output characteristics of the ports of the block may reflect these other "virtual" clocks and may have independent specifications for rise time, fall time and other conditions. But the biggest source of complexity in these budgeted contexts is usually the timing exceptions. Paths within the block that have timing exceptions need to be processed, as well as paths that enter, leave, and/or pass through the block. Exceptions may be expressed as between

pins, ports, nets, clocks, or any combination thereof.



In order to process this model correctly, the clocks in the adjoining blocks must be represented in the dashed block, even though they have no physical embodiment in the block. "Virtual" clocks are used to represent them. The constraints in the dashed block might look like:

```
create_clock -name clka_virtual01
-period 7.0;
create_clock -name clkb_virtual01
-period 8.0;
set_input_delay -clock
clka_virtual01 3.25 [get_ports
p1]
set_output_delay -clock
clkb_virtual01 4.75 [get_ports
p2]
```

ECO

Many articles have discussed techniques for engineering changes (ECO's) of logic function, but there is usually little consideration to the ECO process for timing constraints. Yet timing constraints are typically as complex, and certainly as prone to error as RTL logic. Systems that flatten and expand timing constraints make it very difficult to make changes in the constraints after the floorplan is well underway. What started out as a 10K line procedural SDC file may be expanded to more than 1 million lines of text. When changes are needed in the intended chip level timing behavior, it is usually awkward (at best) to modify the expanded constraint file.

Other technologies

One distinct aspect of the time constraint processing is that it is dependent almost entirely on the complexity of the design, not on the underlying technology. Thus while other EDA problems, such as test generation might be simplified when designing with FPGA's (which tend to be 100% testable without vectors), timing analysis and constraint processing are not simplified through their use. Timing constraints are likely to continue to grow in complexity with each generation of technology.

Summary: Effective Processing of Time Constraints

The complexity and volume of timing constraints in the flow of a chip can be overwhelming. Current experience tends to show about one line of timing constraints are generated per placeable object. So for a SOC with 2 million objects (and perhaps 200 RAM's, or other blocks), one would expect an overall timing exception file of about 2 million lines by the time of sign off.

How can this situation be improved? There is no single simple answer but in general a key to effective processing is to keep constraints in the original format as long as possible, and avoid expanding the constraints as much as possible. Thus a system that starts by reading the constraints and expanding them internally is not likely to be as effective as one that can continue to accept the constraints in the user format deep into the flow. And similarly, a flow that is required to flatten, expand or enlarge them will have disadvantages over one that can output them in a more compact format.

eL-Architect: Using a Design Flow e-Learning Tool in CAD/EDA and VLSI education

José Augusto D. F. Lima, Ph.D.

Departamento de Informática, Universidade do Minho,
Campus de Gualtar, 4710-057 Braga, Portugal
Email: jal@di.uminho.pt

Abstract – The *eL-Achitect* is a design flow oriented meta-platform originally created to provide a flexible integration, documentation and tutorial design guidance for the UML/UEDK CAD/EDA open platform. Its usage has been extended to accommodate any software platform (e.g. CAD) whose functionality is characterized by bundles of independent operations, receiving inputs and outputting results. Such bundles are represented as a knowledge base (KB) of Modules (operations) and Types (input and output data) with associated documents and executable "programs". Selected sequences of such operations should represent a meaningful design-flow within the software platform described in the KB. These are called workflows (WF), and are constructed as sequences of instantiated Modules and Types. The *eL-Architect* can be used as an e-learning meta-platform, of any subject matter besides those of a design nature. As well as allowing the design and implementation - at a chosen level of granularity – of a structural model of any given software system, the *eL-Architect* allows the addition of images and questions and the automatic generation of a corresponding HTML questionnaire, for any sequence of Modules and Types. Upon submission the *eL-Architect* allows the processing of true/false, text, multiple choices and any software program output - binary or ASCII - answers, as well as the management of the authors and their marks.

Departamento de Informática,
Campus de Gualtar,
Braga, Portugal

eL-Architect

A meta-platform

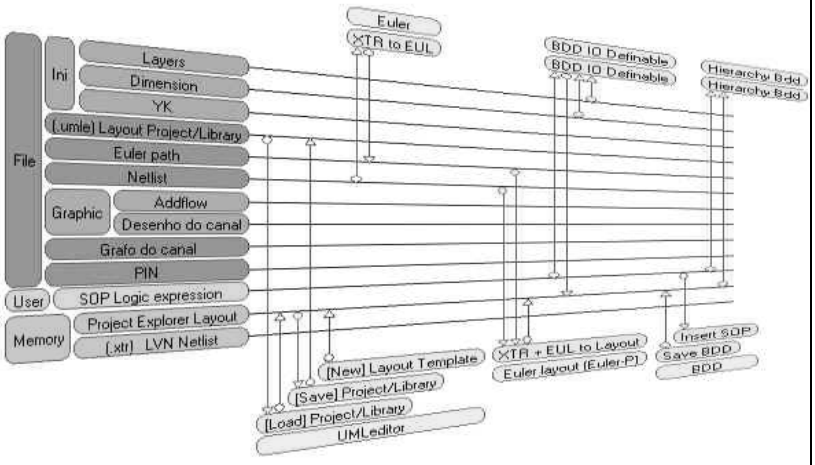
Using a Design Flow e-Learning Tool in CAD/EDA and VLSI education

Over Viewing Key Aspects

- How does one learn about Design-Flow tooling?
- The UML/UEDK CAD/EDA Open platform:
  - Fundamental Features and Design Concepts
  - Resulting Design-Flow and Documentation Challenges
- What is the *eL-Architect e-learning meta-platform*?
- [*eL/D*]-*Architect*
  - Specifications & Design Concepts
  - Key Features and associated Technologies

jal@di.uminho.pt

The eL-Architect



Using eL-Architect *(demonstration)*

- Creating a XML Knowledge-Base for Design:
  - Describing a Structural Model of the System
  - Modules, Types, IO Actions using a *DOF* representation:
    - Adding *e-content* for System's Behavior and Documentation
    - Information Views, KB filtering and Work-Flow Design-Views
- Systems ' Design Consistency and Information Updating
- Creating and Updating Design-Flows
  - Design-Flow Customization and Forking
- Computer Assisted Creation of HTML Design-Flow Tutorials
- Design Evaluation using XML/HTML Quiz Generation
- Solution Submission, Organization and Evaluation

jal@di.uminho.pt

Conclusion & Future Work

- Creating *eL-Architect* Subject Specific Templates
- Include features, for Web centric Design-Flow Integration
- Protocols, Data and Control Integration
- Solution Proliferation
- Document & Tool Versioning Management

jal@di.uminho.pt

R&D Team

- *Prof. José Augusto D. F. Lima, Ph.D.*
- *Marco Aurélio Costa, Ph.D.*
- *Vitor Manuel Campos Rodrigues, Lic.*

jal@di.uminho.pt

Session 5:

New Automated Methodologies and Flows II

Moderator: Takahide Inoue (*STARC*)

Software-Compiled System Design: A Methodology for Field-Programmable System-on-Chip Design

Jeff Jussel and Chris Sullivan

Celoxica Ltd., Abingdon, Oxfordshire, UK

Abstract

With the proliferation of Field-Programmable System-on-Chip (FPSoC) devices such as Altera Excalibur and Xilinx Virtex II Pro, more system integrators are now facing the challenge of merging hardware and software design. This paper describes software-compiled system design, a methodology supporting the design of electronics containing both programmable logic and software-driven microprocessors. To facilitate hardware and software design convergence, this methodology includes C-based design descriptions, hardware/software co-design partitioning and analysis, multi-level and multi-language co-verification, and direct compilation to programmable logic. The paper presents a practical implementation of this methodology using the Celoxica DK Design Suite of tools as well as a design example employing a JPEG2000 algorithm.

1 Introduction

As FPGAs have developed from logic prototyping devices into fundamental system elements, there has been enthusiasm for the concept of using high-performance processors, closely coupled to – or immersed inside – the FPGA fabric to create a Field Programmable System-on-Chip (FPSoC).

In the FPSoC architecture, the microprocessor typically runs system applications while the FPGA manages computationally intensive tasks. The availability of both processor and the equivalent of millions of gates of programmable logic give designers new opportunities to quickly develop, prototype and implement complete systems, delivering both better performance and flexibility.

As FPSoC entails hardware *and* software design, these complete system devices also present designers with all the challenges of system design. In this paper we will explore these design challenges, and present a description of a methodology created to address those challenges, Software-Compiled System Design. The paper also presents a system design example implementing a complex design example on an FPSoC device as validation of this methodology.

The paper necessarily also provides brief descriptions of the tool set used to implement the methodology for this design example. Though Celoxica developed this methodology and provides a set of tools to support it, the methodology itself is independent of any particular design language or tool vendor. The vision is to represent a methodology entailing successful system design practices that may be applied to all system designs that include custom logic and software-driven microprocessors. As the Software-Compiled System Design methodology provides a direct path from higher-level languages, through co-design and co-verification to direct implementation of FPGA logic, the methodology is especially beneficial for FPSoC design.

2 Challenges of System Level Design

Not too long ago the challenges of system design were mainly discussed in the context of System-on-Chip (SoC) devices that combined millions of gates with embedded processors. However, in the unrelenting march of Moore's law, programmable logic designers now have the opportunity to put millions of gates into FPGA devices. Many of them are doing just that as half of FPGA designs now include over 500K gates[1]. As design complexity increases, so does the adoption of higher-level languages – some simulation usage of C and C++ is now included in 74% of designs[1]. The propagation of FPSoC devices combining mega-logic with embedded processors will further spur the demand for system design tools that improve design productivity while maintaining quality of results (QoR).

Traditionally FPGA hardware is designed using techniques, methodologies, and languages borrowed from ASIC design[2]. Software development is undertaken using techniques, methodologies, and languages designed to describe software systems. There has been a divide between the disciplines and it is obvious that hardware and software methodologies do not talk together[3].

As an example, current methods for embedded systems design require that hardware and software be specified and designed separately. System specifications typically use C or C++ and then create a paper document that delegates the functional design to the respective hardware and software teams. The design implementation is then

coded in different languages. The software team maintains the use of C or C++, while the hardware team translates the paper specification into VHDL or Verilog. This process precludes co-design and the partitioning is often decided *a priori* based only on historical divisions. And once the partition decisions are made they can not be revisited, as changes to the partition can necessitate expensive redesign of both the hardware and software,. System verification in this process is similarly hindered by the gap between the hardware design flow and the original specification. The lack of continuity and direct implementation between design phases necessitates exhaustive functional re-verification at every step.

The deficiencies of the current state of system design are clear:

- Lack of a unified hardware-software representation leads to difficulties in verification of the entire system, and to incompatibilities across the hardware-software boundary;
- Defining system partitions in advance leads to sub-optimal designs or requires costly redesign;
- Hardware implementations of the system specification require time-consuming and possibly error-prone rewriting into HDL;
- Lack of a well-defined and flexible co-design methodology makes specification revision difficult and affects time to market.

3 Software-Compiled System Design

To address the challenges of system-level design generally, and FPSoC specifically, we present Software-Compiled System Design (SCSD), a methodology converging hardware and software techniques in the design of electronic systems. This methodology provides a cohesive path from system specification and functional algorithm identification, through partitioning and verification, to system implementation. The resulting output is a functionally verified design implemented in an FPSoC device.

Software-Compiled System Design is defined by the existence of four elements in the design process. First, the algorithm design begins with functional modeling using higher-level language (HLL) design descriptions. Specifically the system descriptions utilize C-based languages such as C, C++, SystemC, SpecC or Handel-C to name a few of the options available to designers. Next using the SCSD methodology, the designer flexibly partitions the design between hardware and software, finding the optimal split before beginning implementation. To meet the third requirement in the SCSD methodology, the system is verified at the highest level to create a functionally correct design before implementation. Finally, the SCSD methodology lives up to the 'software-compiled' portion of its name by providing direct paths to implementation through compilation from C-based descriptions into both software and hardware.

The high-level representation of the Software-Compiled System Design flow is shown in Figure 1 as it is implemented using the Celoxica DK Design Suite.

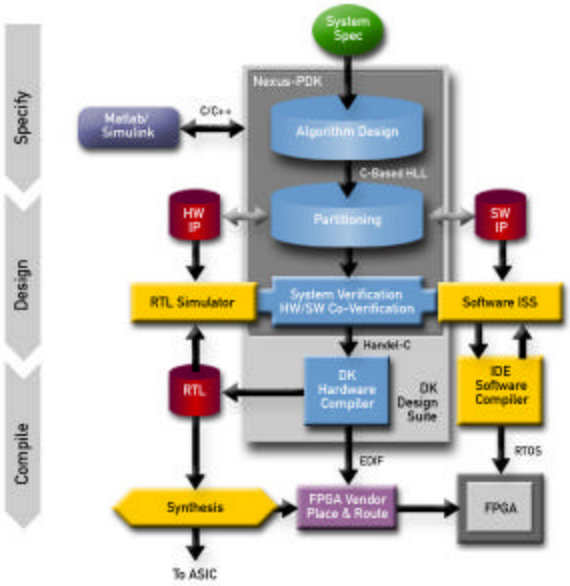


Figure 1: The Software-Compiled System Design flow

3.1 C- based Descriptions

The success of the SCSD methodology is predicated on the use of higher-level language descriptions for system functionality. The exact language used for modeling at this level is less important than the capability to provide compatibility with system architectural analysis and to support easier, verifiable partitioning.

The SCSD methodology supports C-based descriptions that are able to represent the functionality of a design block, independent of the eventual implementation in either hardware or software. While VHDL and Verilog are mature and capable hardware description languages, they are not optimal for compiling software elements of the design to run on processors. C-based description languages enable a common development path between hardware and software elements of the design. This trait of the SCSD methodology allows designers to easily target and retarget functionality between hardware and software implementations.

Often C-based languages are also the functional modeling language of choice for system architects due to their high level of abstraction and interdisciplinary acceptance. Algorithmic models developed in C, or generated from a Matlab/Simulink environment may be used to drive the design. By using these descriptions as the source for functional implementation, the SCSD methodology avoids time-consuming and potentially error prone translations from paper specifications.

There are many C-based languages that could fulfill the requirements for software-compiled system design. So for modeling both hardware and software or maintaining continuity from the algorithm development, the SCSD methodology supports many C-based languages for system functional development. In the Celoxica implementation of the SCSD methodology users have found benefit modeling systems in C, C++, SystemC, SpecC and Handel-C, often employing different languages for different blocks in a mixed-language environment to take advantage of specific modeling strengths in the languages, or to avoid re-writing existing IP. However, as the system design gets closer to implementation, the tools often dictate the specific language for compilation. Embedded software is compiled into the processor from C or C++. Similarly, the Celoxica tool flow employs Handel-C as a simplified route for direct compilation of FPSoC hardware using ANSI-C syntax with simple hardware implementation directives.

3.2 Co-design and Partitioning

Fundamental elements of any co-design methodology, profiling and partitioning are used to help identify optimal design methods early in the design cycle. In the software world, the profiler is used as an analysis tool to examine run time behavior of a program. Applying a similar approach, the profiler in the SCSD methodology can help find the optimal system partition by analyzing hardware and software performance and the interfaces between the two.

Software-Compiled System Design provides a practical means for the co-design of hardware and software. System functionality is developed using technology-neutral C-based descriptions to create a working model of the design. The design is then divided into blocks representing functions that will be implemented in either hardware or software. Using the system specification as a driver, the performance of these blocks and their interaction can be analyzed through profiling tools.

Software block analysis checks for efficient utilization of the processor and performance bottlenecks where parallelization may be added by moving the function into hardware. Hardware blocks are analyzed for execution throughput and the hardware efficiency (gates or FPGA elements). Hardware blocks that are too large or execute a primarily serial function may be moved into software. In addition to the functional blocks, the partitioning analysis must also review the interface between hardware and software to find and remove communication bottlenecks. Due to latency between the system boundaries and interfaces, system profiling can be used to minimize dataflow between the hardware and software. By performing this design analysis and enabling rapid re-partitioning, the SCSD methodology can be used to improve design performance and efficiency.

A key to making this FPSoC design methodology successful is the ability to easily retarget code between software and hardware implementations. In addition to using C-based languages to converge hardware and software modeling practices, the difference in hardware and software interfaces must be addressed. Typically, the FPGA logic is connected to the microprocessor in a memory mapped or programmed I/O fashion. This creates the need to redevelop individual communication protocols and data marshalling routines for each functional block that is moved between hardware and software. This problem is overcome in the SCSD methodology by using an API interface to insert an abstraction layer between the application code and the hardware-software interface.

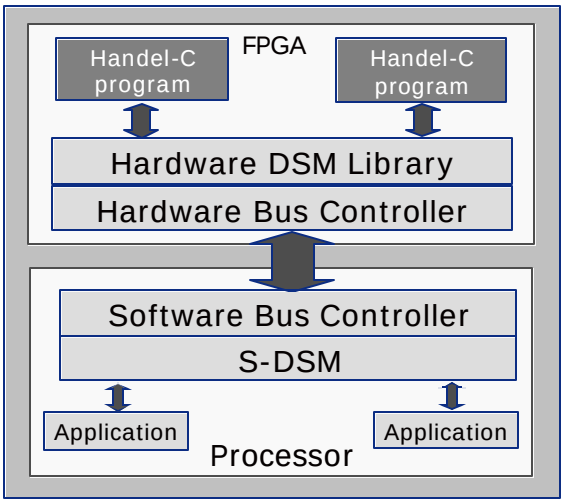


Figure 2: DSM provides an abstraction layer that simplifies the retargeting of code between hardware and software implementations.

The Celoxica Data Streaming Manager (DSM) provides this API for use between Handel-C representations of hardware and ANSI-C software models. DSM offers a simple and transparent interface for transferring multiple independent streams of data between hardware and

software (see Figure 2). DSM simplifies system partitioning and final implementation by abstracting away the interface issues, allowing hardware and software descriptions to share common code and communicate through simple system calls. The DSM API provides a structure that minimizes the overhead to this approach and can be easily retargeted to support different processors. The DSM portability means that multiple partitions can be rapidly evaluated and verified with the original system models used as a test bench.

3.3 Co-verification

Much has been written regarding system level verification strategies and this paper supports the view that functional verification should be done up front in the design process. The Software-Compiled System Design methodology is compatible with specification-driven system verification strategies. The methodology can be described as a functional design process (see Figure 3). The design starts with the specification, and continues with representations derived from the original models. A common test suite may be used to verify the functionality at all levels of the design. The end result of the SCSD methodology is a functionally correct design model that can be compiled directly into FPSoC designs.

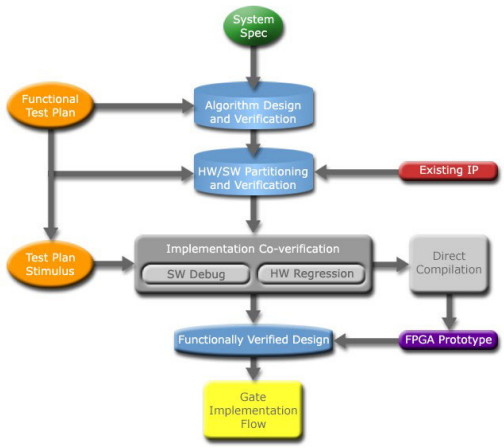


Figure 3: The SCSD methodology supports a specification-driven system verification design flow

To support RTL hand-off to traditional implementation flows, the SCSD methodology also outputs functionally verified HDL models. Using this process, functional correctness is developed at a high-level of abstraction, and maintained via a reference test suite throughout implementation. System performance is also addressed through the SCSD methodology by using profiling and partitioning. This solves major verification and performance issues at the system level, using more efficient levels of abstraction, and significantly reduces the pressure on the RTL implementation flow.

Co-verification technology is critical to the success of the SCSD methodology to support simulation of multiple languages and multiple design levels. Using this approach, C-based representations at the system algorithm level may be simulated simultaneously with RTL hardware representations and software device drivers. The simulation environment must be capable of verifying the models in all the various C-based system representations. The verification environment should also handle verification of all the models generated during system design. This generally requires connected co-simulation between RTL simulators, system-level simulators and software Instruction Set Simulator (ISS) environments. Models of algorithms from a Matlab/Simulink may be run with existing hardware models in VHDL, or with programs running on the processor in an ISS. This allows the designers to modulate the abstraction levels to efficiently simulate the entire system design at each stage of the verification process.

Celoxica implements the SCSD methodology with their Nexus-PDK environment to provide both multi-level and multi-language co-verification. When debugging system performance, it is important that the design verification environment provide a path for both re-partitioning of the system, and for direct implementation to hardware. Nexus-PDK provides a cycle-accurate simulation environment that allows ANSI-C programs (representing system software) and Handel-C applications (representing hardware modules) to interact using DSM. The tool also provides multi-language support for C, C++, SystemC and Handel-C simulation; and third-party co-verification with HDL and ISS simulation, enabling simulation of complete systems. To support partitioning with co-verification, a utility monitors the data passing between applications during simulation. The API functionality used with the SCSD methodology allows system verification to begin immediately and continue seamlessly throughout the functional design process. Once the functional design is completed, it can easily be transferred to a target FPSoC platform for final testing and implementation.

3.4 Direct Compilation

To make co-design and co-verification successful, the methodology must provide a direct, simple, and proven path to implementation. This path should also support the convergence of the hardware and software methodologies so that functionality and performance can be addressed at the system level, instead of at low levels of abstraction. Just as embedded software portions of a

system design are compiled to run on a processor, the hardware portions may be 'compiled' directly for implementation into FPGA logic.

For Software-Compiled System Design, Celoxica has worked with partners to develop a direct flow for FPSoC devices. The DK Design Suite compiles C-based descriptions of hardware directly into FPGA logic. Figure 4 shows the interaction of the combined tool set implementing this flow.

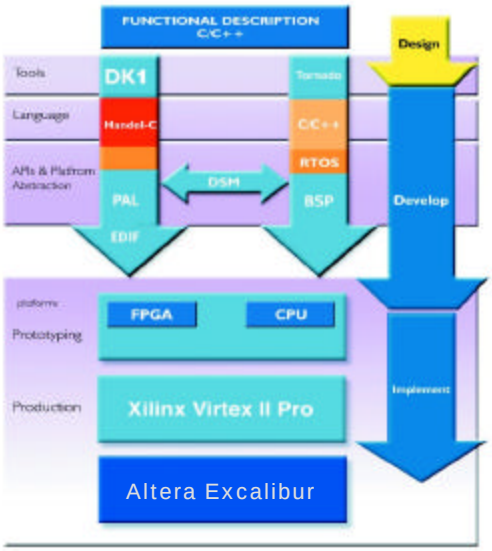


Figure 4: Tool and data interaction in the Software-Compiled System Design flow implementing FPSoC

The SCSD methodology provides an ideal environment for the rapid development of FPSoC applications. Due to the benefits of system partitioning and verification, excellent quality of results can be achieved, substantially beating the results of RTL flows where designers can spend their time optimizing a poor system partition.

3.5 Methodology Benefits

Fusing the techniques of hardware and software methodologies provides significant benefits to designers of FPSoC devices and other systems containing both hardware and embedded processors. The SCSD methodology makes it possible to:

- Prototype the system and easily explore the design space to identify the optimal design solution;
- Partition and re-partition at any time in the design cycle;
- Drive system verification from the specification throughout the functional design process;
- Modulate the design abstraction levels for optimal simulation efficiency;
- Generate human-readable VHDL and Verilog representations of the hardware from C-based descriptions;
- Directly compile FPGA hardware from C-based descriptions for fast design implementation and iteration.

The design case study cited below provides an example of the benefits of Software-Compiled System Design in action.

4 Design Case Study: JPEG2000

To demonstrate the abilities of Software-Compiled System Design, Celoxica and Xilinx partnered on the co-design of a JPEG2000 for implementation in a Virtex II Pro device. In particular, we set out to address the design challenge of system partitioning, co-verification, and the ease of hardware and software integration.

JPEG2000 is a standards-based image-coding system that uses state-of-the-art compression techniques based on wavelet technology. Its architecture lends itself to a range of uses from consumer electronics such as digital cameras, through to medical imaging, remote sensing, surveillance systems and scanners.

The major engineering goals for this project were to maximize the overall system performance while demonstrating an efficient and effective co-design environment for FPSoC design. The JPEG2000 project used an ANSI-C software specification as the starting point for the system design and employed Software-Compiled System Design using Celoxica, Wind River and Xilinx tools to implement the methodology. The project was completed in four phases with verification continuously applied at each stage: Profile, Partition, Design, and Implement. The design was completed with one engineer over the course of about 2 working weeks.

4.1 Phase 1: Profile and Verify

The project started with the JPEG2000 ANSI-C source code, an application that could benefit from acceleration and flexibility of the FPSoC hardware solution. This code was considered the functional specification for this design. This example is not unique, as a multitude of similar applications could also benefit from the SCSD methodology.

To drive the system verification flow, the JPEG2000 specification code was simulated as software through an appropriate processor target, in this case the IBM PowerPC 405GP. From this exercise the functionality of the system was simulated and verified to establish a test bench that remained constant throughout the design.

The design engineer then performed code profiling to establish where the program spent its execution time and which functions called other functions during execution. This profiling quickly identified the compute-intensive functions in the program. Using Wind River's WindView visualization and diagnostic tool, the DWT (Discrete Wavelet Transform) and Tier 1 encoder were determined to be the processor-intensive functions, consuming 87% of processing time (see Figure 5). Those two functions were selected for further partitioning and analysis.

Windview Trace of entire JPEG2000 algorithm showing the total amount of processing time spent.

Analysis Indicated DWT & Tier 1 were processor intensive functions

Profiling results:

Total time = 8.01s (100%)

Tier 1 = 5.08s (62%)

DWT time = 1.90s (25%)

Figure 5: WindView trace of the JPEG2000 algorithm

4.2 Phase 2: Partition and Verify

Software-Compiled System Design is unique in providing the designer with a flexible partitioning methodology linked to system verification. Using the Celoxica tools and DSM technology the designer can confidently explore and innovate in the design space to identify the optimal system partition for the best Quality of Design (QoD).

In the JPEG2000 project, DSM validated the profiling information determined in Phase 1 of the design. Using DSM, the design engineer analyzed the data flow, burst length and frequency between the hardware and software and fine-tuned the partition to optimize the project goals. The DSM API facilitated the process of design partitioning between hardware and software (see Figure 6).

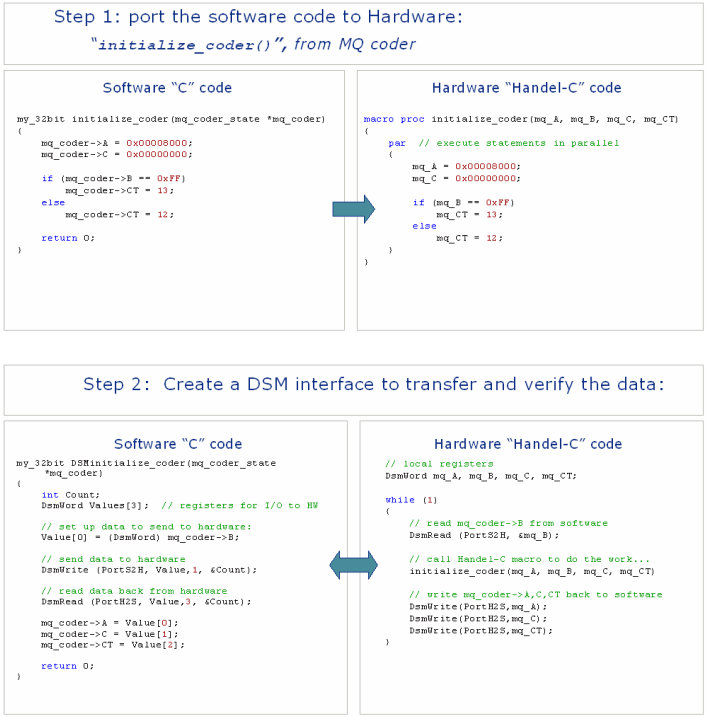


Figure 6: DSM is used to ease the process of moving design functionality between software (ANSI-C) and hardware (Handel-C) representations

4.3 Phase 3: Design and Verify

With the optimal partition determined and verified, the optimization of the design began. With SCSD, the performance and size optimization can be done by combining hardware and software functionality, looking at the system in its entirety. Optimizations of the JPEG2000 software included combining multiple function calls into single calls. Blocks destined for hardware were optimized by employing more parallelism, and editing functions to infer pipelining and efficient hardware constructs.

DSM was used to provide a cycle-accurate simulation environment that allowed the hardware and software to interact, keeping them connected throughout design optimization. The software in the system was run as a native executable on a PC, with the hardware being run in C and Handel-C using the simulation capabilities of Celoxica's Nexus-PDK. Co-simulation between the hardware and software was performed in tandem the Wind River Tornado environment (see Figure 7).

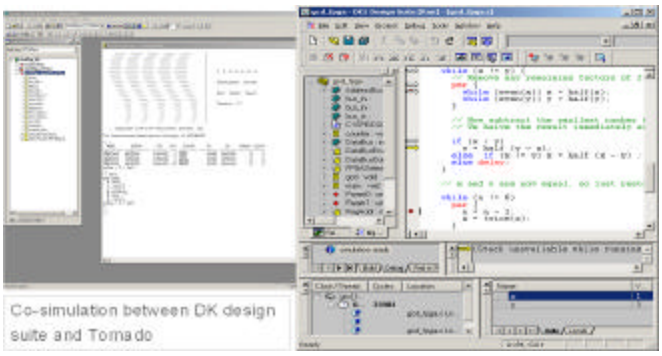


Figure 7: Celoxica's Nexus-PDK and the Wind River Tornado environment running co-verification of the system hardware and software in tandem

Since the system specification was described in ANSI-C, the designer implemented the software in ANSI-C, and added Handel-C hardware extensions to the code to represent the hardware. These extensions provided efficient control over the area, timing, clocks, RAM, ROM and interfaces of the FPGA logic.

4.4 Phase 3b: Specification Change

At this stage in the design a specification change was introduced. A novel lifting algorithm was developed that performs a two-dimensional DWT to improve processing time. The algorithm was readily available as an HDL IP block and the decision was made, in the context of minimizing time and maximizing IP investment, to integrate the IP into the design as a black box. The integration was simplified by using the 'interface' declaration from a Handel-C block in order to connect the third-party IP, complete with RTL co-verification, into the Software-Compiled System Design flow.

4.5 Phase 4: Implement and Verify

The target platform for the JPEG2000 project was the Wind River SBC405GP and Proteus FPGA daughter card, a Virtex II Pro prototyping platform. The design was implemented in this environment for timing simulation, emulation and block optimization. Final implementation was retargeted to the Virtex II Pro ML300 evaluation platform.

Re-targeting from the development platform to the final evaluation platform was simplified using the Celoxica Platform Abstraction Layer (PAL), which provides an API to shield the application code from the low-level hardware interfaces. This built on a library of low-level interfaces specific to each target platform called the Platform Support Library (PSL), which was accessed using the PAL API by the Handel-C application code representing this FPGA logic.

Software object code for the PowerPC processor was compiled directly from C into the PPC405GP under VxWorks. The hardware implementation was compiled directly from Handel-C using the EDIF output generated by the Celoxica DK Design Suite. This EDIF netlist was optimized for Virtex II Pro for maximum efficiency and best Quality of Results (QoR). The DK Design Suite also produced VHDL and Verilog output of this design hardware, though the RTL path was not used for implementation this project.

4.6 Design Results

The DWT results for the JPEG2000 project are shown below in Table 1. These results are compared against the performance of the handcrafted VHDL authored by a JPEG2000 domain expert. The Software-Compiled System Design methodology provided a systematic approach to the problem leading to not only substantial savings in design time, but also to improved design quality.

To achieve the results, the DWT portion of the JPEG2000 design was compiled directly from Handel-C to FPGA hardware. The design was optimized to achieve maximum system speed while maintaining or reducing the hardware utilization area. To validate the methodology, the results were compared against the same DWT functionality as originally hand-coded in VHDL. Using the SCSD methodology, the designer was able to achieve a comparable size design with faster performance results in less time.

JPEG2000 Results	Original (VHDL)	Celoxica SCSD		
		Pass 1	Pass 2	Final Results
Slices	800	646	546	758**
Device Util.	7%	6%	5%	7%
Speed (MHz)	128	110	130	151
Lines of code	435	386	386	395
Design time (days)	20*	6	7 (6+1)	7 (6+1)

*VHDL design time does not include partitioning
**Final SCSD result Includes HDL IP block

Table 1: JPEG2000 case study DWT function implementation metrics

The engineer on this JPEG2000 project, while an expert in Celoxica tools and Handel-C implementation, had no prior experience with the JPEG2000 algorithm and yet was able to translate the algorithm into a working hardware implementation in less than half the time of the

original VHDL implementation. The engineer also easily met the system design constraints for size and performance as seen in the table of results.

The difference is that with the SCSD methodology the designer was able to work at a higher, more efficient level of abstraction, and then concentrate on optimizing the entire system. These results provide clear validation of the SCSD methodology raising the level of abstraction to increase designer productivity without compromising design quality or performance.

5 Conclusion

Software-compiled System Design is a proven methodology for the design of programmable systems. It employs high levels of abstraction through the use of C-based design languages to provide solutions for system partitioning, co-verification and the integration of hardware and software into FPSoC devices.

The Celoxica implementation of the SCSD methodology connecting the DK Design Suite to common embedded software and FPGA design tools provided the tool set for a real-life system design test. The performance results from this test demonstrated significant improvements in overall design productivity, while also improving system performance and quality of design. Overall improvements in the quality of design were realized by informed and accurate partitioning decisions. Better up front system verification and direct compilation to FPGA hardware improved the design productivity. Finally, the system methodology allowed the designer to find speed and area optimizations in the system by analyzing the entire design including interactions between the FPGA logic in the microprocessor.

The bottom line is that the Software-Compiled System Design methodology offers real competitive advantages for designers of programmable systems.

References

[1] G. Smith, "ASIC Design Times Spiral Out of Control", *Gartner Dataquest Research*, April 2002

[2] P. Garrault, Synthesis Tool Enhancements for Virtex Architectures, Xilinx, 2002

[3] G. Smith, "EDA 2002: The Acceleration of RTL Design", *Gartner Dataquest Research*, October 2002

Improving SoC Design Flows with Robust and Precise Embedded Memory Models

Jay Abraham

Silicon Metrics Corp., Austin TX, USA

Abstract

Practically all digital System on Chip (SoC) designs contain embedded ROM, RAM, or register file memories of various sizes. These types of embedded memory on average consume 30 - 50 percent of die area and this percentage is growing at an astounding 25 percent annually [1]. With so many designs containing memory devices, on-chip critical timing paths will either start or end at memory address, data, or control pins. In addition to clock circuitry and I/O ring power consumption, memory power consumption is the most significant factor for on-chip power expenditure [2]. Therefore, in order to generate high-quality SoC designs, one must have the ability to accurately model embedded memory devices [3]. Existing methods for embedded memory characterization and modeling for timing and power take too many shortcuts. These methods result in poor quality models that when used in design flows for nanometer technology processes cause timing closure issues and unpredictable power analysis results. This paper presents a unique methodology for accurate and complete timing and power model generation for embedded memory. Unlike current solutions that advocate simplifications that sacrifice accuracy and correctness of memory models, our solution enables processing and analysis of the entire memory in order to automatically generate accurate and complete embedded memory models. The benefits of using this solution with two SoC design flows will be discussed.

1 Introduction

With the increasing use of embedded memories in SoC designs (see figure 1), it has become more difficult for design construction, optimization, and analysis tools to efficiently perform their function. Standard cells are the building blocks of modern IC designs. However, embedded memories cannot be characterized and treated like standard cells [1, 3]. This means that construction and optimization tools like synthesis, floorplan, and place and route, are unable to accurately account for the correct timing and power of the instantiated memory. Instead, these tools rely on users providing approximated constraints. Users then rely on final analysis to rectify approximations made earlier. This puts a significant burden on the accuracy of final analysis.

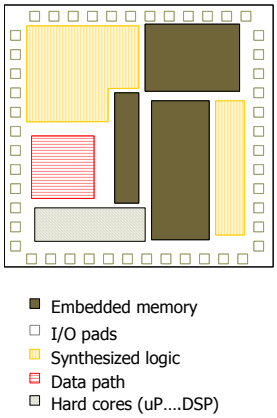


Figure 1: Typical SoC Design Layout.

To mitigate these inaccuracies, SoC teams are forced to gaurdband and over-design. However, over-designing to accommodate timing inaccuracies translates into increased die area and increased manufacturing costs. Over designing to accommodate power inaccuracies directly influences chip pin-out and packaging costs. Lack of information and inaccurate estimates can delay power related decisions, thus negatively impacting design schedules. This is especially troublesome in mixed-signal designs where pin-out can determine the success or failure of analog components. The implications of all these trends are clear: memory characterization and modeling is of significant concern to SoC design teams and accurate memory models are required in all phases of design (see figure 2).

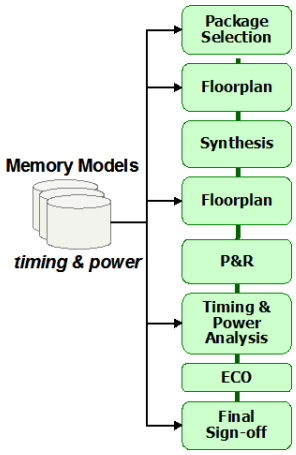


Figure 2: Use of Memory Models in a Design Flow.

2 Traditional Solutions for Generating Memory Models

Traditional solutions for memory model generation are mainly repackaged memory compiler based techniques and lack significant levels of automation. Typically they consist of either a netlist prune/cut based approach to approximate electrical behavior of the memory device or, the use of memory compilers to generate required models. Both of these approaches will be considered in detail.

3 Pruned Netlist or Cut Based Memory Model Generation

Some design teams attempt to simulate a memory with SPICE by means of a pruned netlist or cutting approach. This manual methodology can be a very time consuming and error prone task. Due to capacity issues, it is impossible to simulate an entire memory using SPICE. Therefore, the memory netlist must be cut and pruned. Netlist cutting creates a piecemeal analysis of the memory components. The address decoding logic, wordline, bitcell, bitlines, column multiplexing, and output logic are analyzed individually using SPICE and the final model is composed of calculations based on the individual results. Netlist pruning is the process of isolating one or more addresses in the core while discarding the rest. The result is a smaller netlist on which SPICE can be used effectively. Both of these processes are error prone and time consuming, especially for designs in the post-layout phase. Furthermore, neither of these methods addresses pre-layout model needs and real power consumption. Standby and active power usage are usually estimated based on the size of the memory.

As shown in the figure 3, the process for generating a memory model with a netlist prune/cut based approach is fairly complex. The phases for critical path identification (for pruning purposes), reduction of the extracted netlist, and specification of the measurements and stimulus to drive SPICE are not trivial. These tasks require transistor-level expertise as well as significant preparation time. Once again, there can be significant loss of accuracy due to the cutting process and reduction of the netlist. Cutting leaves lots of dangling nets which if tied off with lumped capacitors will not reflect the actual behavior of active transistor devices [1]. Additionally, it is vital that this approach find the correct critical path to prune because this is the only path in the entire memory that will be simulated with SPICE. Without full transistor-level analysis of the netlist, including all of the detailed parasitics, it is difficult to manually determine the actual critical path.

Typically, the final accuracy of a netlist prune/cut based method is better than relying on a compiler generated memory model; however, significant inaccuracies still exist. When comparing to SPICE, about 25% - 40%. Error is usually observed. This error is still significant for today's nanometer SoC designs. Finally, a netlist prune/cut method does not address the need for accurate power characterization. Instead a maximum power usage value can only be estimated. This estimation can have dire consequences for a today's battery operated devices.

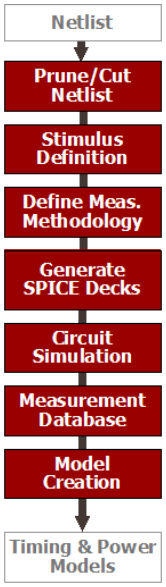


Figure 3: Typical Flow for Netlist Prune/Cut Based Characterization and Model Generation.

4 Memory Compiler Generated Memory Models

Memory compilers generate models of a specific memory instance by interpolating the performance from the data of a few known instances. Therefore, the actual memory instance instantiated in the design may not have been characterized. Additionally, the chosen instances are characterized by pruning and cutting statistically determined critical paths. Pruning and cutting as explained in the previous section adds additional error to the characterization process. Interpolation and extrapolation is accomplished with various mathematical functions, such as linear interpolation, statistical curve fitting, time series analysis, and splines. Given that interpolation and extrapolation isn't an exact match, there can be significant error when comparing the interpolated data to the actual results [7].

In order to verify accuracy, the following experiment was conducted with an 8X2bit single port RAM. This analysis consisted of running SPICE for various memory constraints such as access time and address setup. The data from these measurements were then compared to data from typical memory compiler generation processes.

We created a compiler using standard memory compiler techniques which involve selective characterization of unique memory types, i.e. user selected address and data input/output bus widths, aspect ratio, etc. This method is favored by commercial compiler developers utilizing a netlist cut or prune based approach [8]. The characterized data was then interpolated to generate data for the memory model in question (the 8X2bit single port RAM). Due to runtime limits of running SPICE on larger memories, a small memory was chosen for this example. The graphs for SPICE versus compiler data are shown in figures 4, 5 and 6. Due to runtime overhead for power consumption data (>3 days per measurement) we had to use aggressive netlist cutting techniques. This approach is typical in commercial compiler designs. Data for power accuracy is shown in figure 7.

As shown in the graphs in figures 4 - 7, cutting, pruning, estimating, gaurdbanding, interpolating, and extrapolating can inject a significant amount of error in to the final analysis of the memory. In this example a maximum error of about 110% was observed (see figures 8 and 9). Given that the memory device is often in the critical path of most SoC designs, inaccuracies as shown below will significantly impact the final performance metrics of the design.

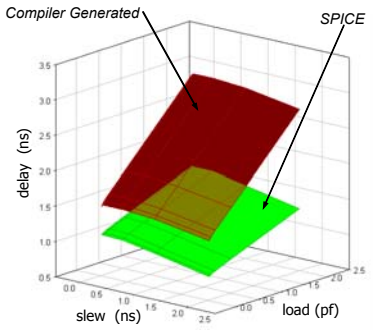


Figure 4: Access Time.

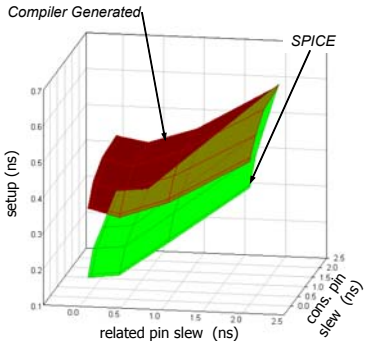


Figure 5: Address Setup.

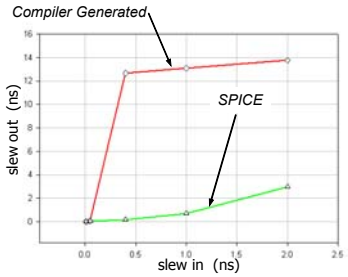


Figure 6: Output Slew.

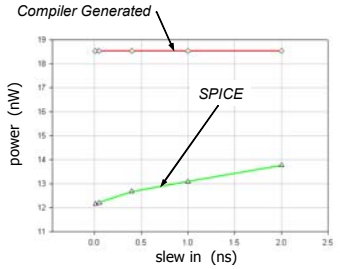


Figure 7: Average Power.

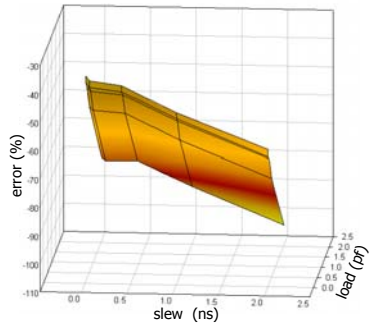


Figure 8: Access Time Error (-40% to -100%) for Compiler vs. SPICE.

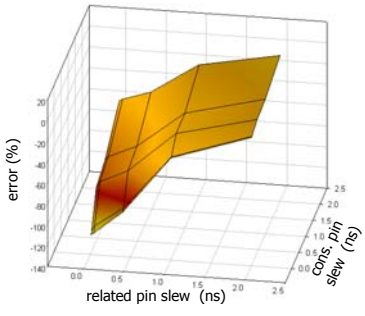


Figure 9: Address Setup Error (-2% to -110%) for Compiler vs. SPICE.

Additionally, memory models generated from compilers for timing analysis are usually created for single process, voltage, and temperature (PVT). If the model is not designed to represent the actual operating point

(environment), then there is a real danger the memory may not operate within the timing constraints [1]. Additionally, compiler generated memories rely on foundry data available during shipment of the compiler. Therefore, the memory models generated may not be built from the most recent foundry SPICE models. These deficiencies in modeling accuracy can result in failed power-on of the chip.

Standard compiler based memory designs only marginally support ultra-low power design efforts. Due to estimation in standard power characterization methods for memories, power usage is often approximated to a fixed value. However, EDA tools utilizing the ALF [4] and Liberty® (Synopsys .lib) [5] library formats require variable average memory power usage. In the absence of adequate characterization solutions, memory compilers will insert gaurdbanded estimations which are closer to maximum power usage than average power usage. As a result, power consumption budgets have artificial constraints and additional guardbanding. In ultra low power applications, this type of guardbanding may overburden other parts of the design where the excesses must be compensated for in order to meet strict power consumption limits.

5 A Contemporary Memory Characterization and Modeling Solution

The latest solutions for advanced memory modeling should strive to improve the accuracy of the models, while also providing an automated, high-throughput flow for the generation of the models. Ideally characterization and model generation should be a push button process. One of the ways to accomplish high throughput runtime is to leverage hierarchical simulators. Array reduction capabilities in these simulators are suited for memory devices. Additionally, the variance to traditional SPICE is often only 2% - 4% , resulting in highly accurate timing and power models [6].

The flexibility provided by this method enables designers to generate memory models that fit the unique needs of a particular design including the actual operating point(s) (PVT). Because the entire post-layout netlist is actually simulated, designers no longer need to compensate for estimation in the measurement process. The models generated are instance specific for the SoC design that they will be embedded in. Because the memory model is applicable to a particular design, guardband is user-controllable. Most importantly memory characterization no longer requires netlist cutting, netlist pruning, or synthesis of the measurements of the individual memory components.

In order to verify memory characterization and modeling with, we built the following automated flow (see figure 10).

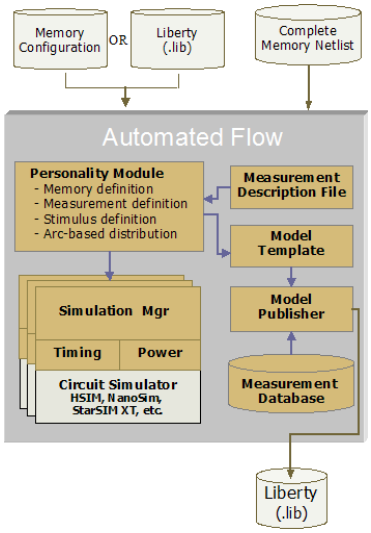


Figure 10: Automated Flow for Memory Characterization and Model Generation.

The inputs to this system consist of a complete memory netlist with interconnect parasitics and a memory configuration file. The memory configuration hierarchically describes the function of the memory, including pins, functional relationship between pins, and in some cases the timing and power arcs within the memory. The memory configuration input is simple to generate and is typically 15-30 lines of Tcl code. Based on the configuration input, the system automatically determines the required measurements. The alternate input is the industry standard Liberty™ (.lib) source file. Once the measurements have been determined, simulation is performed with a high-speed simulator. Data is stored in a measurement database and model writers generate Liberty™ (.lib) models. We also built a model verification capability to verify syntax and semantics of the generated models.

Results from this methodology showed significant promise in terms of the accuracy of the models generated as well as the throughput of the system. In terms of accuracy, the graphs in figures 11 and 12 show the results of comparing measurements from this system compared to exact SPICE measurements.

As depicted in figures 11 and 12, the new models generated with the automated flow utilizing high-speed simulators generates models that are very accurate. Typical variance from SPICE is 2% - 3%, with a maximum variance of about 5%. Throughput is extremely efficient as well. Utilizing a parallel processing technique

for arc by arc parallel distribution as well as SPICE deck optimization, the single port RAM (size: 8X2) was processed in approximately 15 minutes. As a result of the hierarchical processing capability of simulators, larger memories are characterized very quickly. For example, a 32Kb memory requires 2.5 hours and a 256Kb memory requires 9 hours.

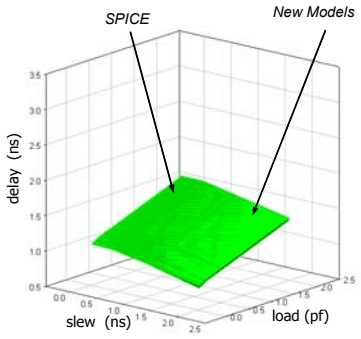


Figure 11: Access Time.

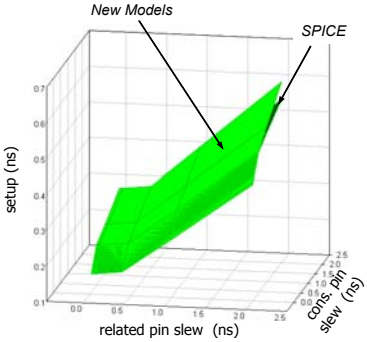


Figure 12: Address Setup.

6 Using Accurate Models in a Contemporary Design Flows

Correct and accurate memory timing models are needed for timing driven place and route, static timing analysis, and final timing sign-off. Accurate memory power models are needed for low cost package determination.

It is important to eliminate complexity from the memory characterization and modeling process to turn characterization and memory model generation into a pushbutton process. Requirements for automation include; automatic stimulus generation, arc-based job distribution, automatic deck creation, archiving, and prepackaged timing and power methodologies that match the target model. High throughput for characterization is obtained with the use of hierarchical simulators, parallelized distribution of simulation jobs that are granularized at the per-arc level, and intelligent spice deck creation to utilize hierarchical processing.

Our characterization and modeling solution addresses these key issues of timing and power closure with embedded memories in advanced SoC designs. With a pushbutton process for generating precise memory models for timing and power, designers can generate memory models for their unique application.

The solution solves power accuracy issues by providing easily configured power acquisition capability. Complex analysis can be performed to determine average power consumption in de-sign-specific configurations. Designers can account for typical utilization patterns such as sequential access versus random access and generate power numbers appropriately. Analysis of various modes, such as low power or quiescent modes is also possible. This gives designers the information needed to make important decisions earlier in the design cycle such as decisions related to packaging, chip pin-out, floor planning, and power routing.

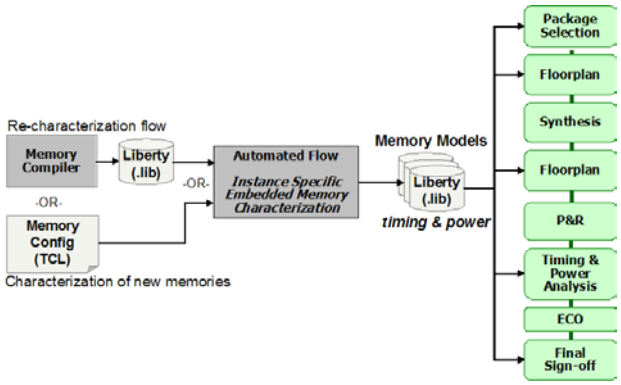


Figure 13: Improved Design Flow with Robust and Precise Embedded Memory Models.

The recommended flow utilizing memory characterization and modeling is shown in figure 13. Two flows that are applicable to SOC designers are shown. The first is a re-characterization flow in which designers take existing memory models (generated by a compiler) and perform a re-characterization step to improve accuracy of the models. Typically the model format will be in the public domain Liberty format (also known as .lib). With this methodology, users can characterize for unique PVT conditions required for their design. The second flow is for characterization of custom memories. These could be hand crafted memories or compiler generated memories that have been modified. In this case, there are no existing models. The suggested flow would involve creating a memory configuration file that is then read by the automated characterization system. This flow is also useful for re-characterization, when an existing memory model generated by a compiler may not be of good quality. In this case, model generation from scratch would be the preferred solution. The flows described liberate memory design teams to concentrate on circuit

design and memory architecture development rather than the tedious and time-consuming process of verifying and supporting existing designs.

7 Conclusion

Time-to-market, custom package tooling, die real-estate requirements, and the ensuing design closure issues create a complex decision making process that demands accuracy and flexibility. Embedded memories are often at the core of this process due to their increased dominance in SoC designs and can negatively impact a design flow. Traditional methods for memory model generation and characterization are often inaccurate and use inefficient processes. A new solution that leverages high capacity simulators offers the capability to generate accurate models with high throughput. This system was verified with test cases to demonstrate the capability to generate accurate models for various memory types. The system is flexible for acquiring data for timing and power. Data described in this paper showed a minimal 2% - 3% variance from SPICE. The use of this system in SoC design flows will enable improve quality of design. The commercial application based on this research, SiliconSmart MR from Silicon Metrics, showed that accurate timing and power memory models of any size can be generated with a rapid, precise, and pushbutton process. Thus enables designers to successfully implement high-speed and high-density embedded memory-based SoC designs that meet today's demanding cost and schedule requirements.

References

[1] Eric Hall and George Costakis, "Developing a Design Methodology for Embedded Memories", *Integrated System Design*, January 2000.

[2] Alberto Macii, Luca Benini, and Massimo Poncino, "Memory Design Techniques for Low Energy Embedded Systems", *Kluwer Academic Publishers*, 2002.

[3] Tegze Haraszti, "CMOS Memory Circuits", *Kluwer Academic Publishers*, 2001.

[4] "Advanced Library Format for ASIC Technology, Cells, and Blocks version 2.0", *Accellera*, 2000.

[5] "Library Compiler Reference Manual volumes 1-3", *Synopsys*, 2001.

[6] Sam Wang and An-Chang Deng, "Delivering a Full-chip Hierarchical Circuit Simulation and Analysis Solution for Nanometer Designs", *Nassda*, May 2002.

[7] Ravi Agarwal and Patricia Wong, "Error Inequalities in Polynomial Interpolation and Their Applications, *Kluwer Academic Publishers*, July 1993.

[8] Charles Longway and You-Pang Wei, "Automatic Memory IP Characterization", *EEdesign*, December 6 2000.

Building Design Process in a Startup Company

David A. Gates

ATI Research Silicon Valley, Inc., Santa Clara, CA, USA
gates@ati.com

Abstract

A startup company must build its design process up from nothing into a complete system in a very short period of time. However, a startup design team is confronted with several other challenges as well. The area of functional verification is one where complete design automation is essential to first-time design success. Built from scratch, the verification environment is gradually brought under fully automated control. DV, a verification flow manager, is the product of one such progression. Its software requirements and how it meets them are described.

1 Introduction

A startup ASIC design company has one big disadvantage and one big advantage when it comes to design process: the absence of one. Lacking a design process, a startup must build or acquire all the necessary pieces to complete a design. This can be an expensive and time-consuming proposition. On the other hand, the new company is not saddled with the compromises and tradeoffs made as a design process evolves to become the standard design environment of an established company. The startup starts with a clean slate, limited only by the experience and creativity of its early employees. The challenge during the startup phase is to build a design process that allows the company to get its first product out the door on time, while at the same time providing a solid basis for future design cycles.

In this paper, we describe the challenges facing a startup company that affect the way it goes about completing an ASIC design. These challenges place constraints on the way the design process evolves. In particular, focus is placed on the functional verification process, an area where comprehensive design automation is essential to project success. Evolution of the design process through various stages shows how an initially chaotic situation is gradually brought under the control of increasing automation. The end result of this evolution is a verification flow manager [1] that meets the requirements of a startup company. The architecture, implementation and use of such a tool called DV (for Design Verification) are described. Details and examples of how DV meets its requirements are presented.

2 Startup Challenges

Few, if any, startups choose to attack an entrenched competitor using commonly available means. Instead, a startup's goal is generally to introduce a new product or technology before any of its competitors. Such competitors can take the form of both other startups with similar goals and well-established companies working to improve their existing products.

2.1 Fluid Goals

A startup relies on its small size and lack of bureaucracy to adapt quickly to changing market conditions. It can also change goals quickly when faced with an insurmountable technical problem.

The design process must be able to tolerate large swings in the technical direction of the venture. For example, the team may be initially committed to a new technology node, only to realize that using the current node is the only way to hit a market window. A design process that takes weeks or months to reengineer could cause the startup to lose its competitive advantage.

2.2 Time Pressure

Quick reaction time is a response to the extreme time pressure startups face. If a competitor beats the startup to market, the most likely outcome is that the startup will cease to exist. Employees work long hours to compensate for the limited capital and personnel resources of the startup.

Time pressure impedes the development of a consistent and repeatable process. When a tradeoff is required between development time and process quality, time will generally win out. It is much easier for a weary engineer to focus on solving the problem at hand than to think about generalizing the solution. However, a design environment can mitigate this problem by capturing the design process as it develops. This captured form can then be reviewed and updated to establish standard practices.

2.3 Diverse Workforce

The engineering team has no doubt been recently assembled by hiring experienced individuals from different backgrounds. Each engineer brings their own training and experience to bear on the design process. Each will have strengths in some areas and weaknesses in

others. By drawing on the experiences of all engineers, the design process of the new organization can be built on the best practices of the group.

In some cases, two or more equally valid methods will be available. It may make sense to allow both to coexist, to choose one over the other outright, or to give each a trial to determine the preferred method. For example, in the first case, engineers working on different blocks of a design can use techniques most familiar to them to complete their designs. Forcing a common design process on everyone could introduce inefficiency that would lengthen overall development time. Instead, an extensible design environment is needed to accommodate process diversity.

2.4 The Leading Edge

Often the team will be facing technological hurdles on the leading edge of ASIC design. Innovative products have specifications that require the latest process node, a larger, faster ASIC, exotic process technology or all of the above.

In light of this, the design process needs to be able to adapt to handle these challenges. New tools and flows will be needed that no member of the team has dealt with before. It may be necessary to experiment with different solutions before an acceptable one is found. In addition, engineering teams working in parallel may end up solving the same problem in different ways. Only a flexible design environment will be able to cope with these issues.

3 Functional Verification

One area of ASIC design where complete design automation is essential is functional verification. Full verification of a complex, multi-million gate ASIC requires thousands of directed and random tests to be run. The majority of these tests are aimed at verifying the RTL representation of an ASIC design.

3.1 Example Verification Flow

Figure 1 shows a simple flow for testing an RTL design. It is used for examples in later sections. Common infrastructure is compiled first for use by later steps. The RTL source is compiled into a simulator. In parallel, a test stimulus generator is compiled and run to generate input vectors for simulation. During simulation, the input vectors are fed in and output results are captured. These results are then checked for correctness against 'golden' results. The golden results are assumed to be correct and can be obtained in a variety of ways.

3.2 Automation of Verification

Verification is performed at both the individual block level and at the chip level where all blocks have been integrated together. The exact details of how to run tests at each of these levels are different and should be hidden

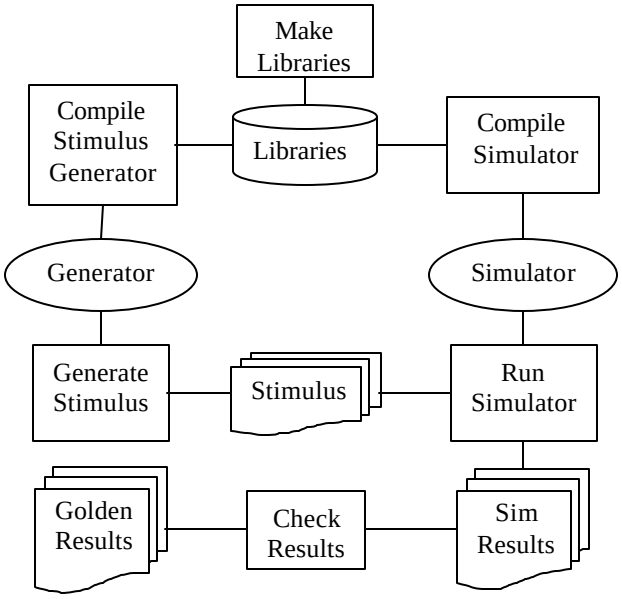


Figure 1: Example Flow

from the designer as much as possible. This makes it easier for the engineer to focus on design and debug of a block, yet still be able to test the block in context.

Abstracting the test process also helps when running test suite regressions. It is desirable to have a common environment that can run one test at a time interactively or entire test suites in the background. Batch queuing systems such as LSF [2] make the latter task feasible.

4 Design Process Evolution

When developing a new design environment, the design process evolves through several stages on a path towards comprehensive automation. In the following it is assumed that the team is already following certain common practices for good design: use of detailed specifications, well-documented code, source-code control with individual sandboxes for each engineer, and others [3].

4.1 Piecemeal Automation

Initially, individual pieces of the verification environment are developed in isolation. Engineers use commonly available tools such as Make, Perl, Python and TCL to automate building of the various components. The choice of which to use for each task is left up to the engineers, who make their decisions based on familiarity with a given tool or a desire to learn a new one. The net result is that the design environment is covered by a group of isolated areas of automation.

4.2 Manual Flows

In the next phase, engineers develop ad hoc flows to build multiple sections of the sandbox. For example, the following list might be used to run a simple flow:

```
# Make libraries.
1 > cd $TOP/lib && make
# Compile stimulus generator.
2 > cd $TOP/test/src && make gen
# Compile simulator.
3 > bld_sim.pl $TOP/sim/c1.conf
# Generate stimulus.
4 > cd $TOP/test && src/gen t1
# Run simulator.
5 > run_sim.pl –test t1 –conf c1
# Check results.
6 > chk_sim.csh –test t1 –conf c1
```

Note that this flow is a typical mix of different automation tools used at different steps. Initially, the flow is not fully automated because the complete list of steps needed is being discovered by trial and error. Recall that it is likely that no one person understands the complete system, since the pieces have been developed by several people working independently.

As an engineer makes changes to the sandbox such as modifying RTL source, fixing a library bug, or extending a test case, they must rerun parts of the flow to bring the sandbox up-to-date. In many cases, it may be so difficult to determine which pieces are out-of-date, that an engineer will choose to rerun all of them. The alternative is to risk working with an inconsistent sandbox which contains obscure, difficult-to-trace bugs.

One advantage manual flows have is being easy to debug, since the steps are run one at a time and the output of each step is immediately available for inspection.

4.3 Hard-coded Scripts

In short order, typical engineers become frustrated with the tedium of manually running through such flows over and over. In response, a hard-coded wrapper script is written that runs through the entire flow.

If the flow is simple enough, a target in a Makefile may suffice:

```
runtest1:
    # Make libraries.
    cd $TOP/lib && make
    # Compile stimulus generator.
    cd $TOP/test/test1 && make test1
    ...
```

However, in most situations, the dependency / update structure of Make interferes with efficient implementation, and a scripting language is used instead.

Running multiple steps automatically one after another can make it difficult to determine which output belongs to which step of the flow. For example, while errors will generally stop flow execution, mere warnings will only show up as messages embedded in the output stream. Working around this problem requires extra time and effort that might be skipped when developing many hard-coded scripts.

4.4 Generalized Scripts

Hard-coded scripts have the disadvantage of being very inflexible. Although they can be parameterized to run different instances of the same kind of test, a new script must be written for each major variation of a flow. Users must either learn the details for running each kind of test, or a wrapper must be written that calls the appropriate script as a subroutine.

What is desirable is a generalized form of scripting that combines the labor-saving of hard-coded scripts, the observability of manual execution and the flexibility to integrate new flows that neither provides. In the next section, we describe a tool called DV that has these three properties and more.

5 DV Architecture

The requirements for a functional verification tool meeting the special needs of a startup design team are summarized as follows:

- Flexible: can accommodate multiple flows and levels of design detail.
- Extensible: users can define new flows applicable to a particular part of the design.
- Formatted: test cases must be stored in a way that users can maintain.
- Automated: can run one or many tests interactively or as a batch.
- Observable: can easily follow the flow of control to spot problems.
- Modifiable: can be updated while in use as a project progresses.

DV (short for Design Verification) is a tool developed to meet these requirements. The architecture of DV is shown in Figure 2.

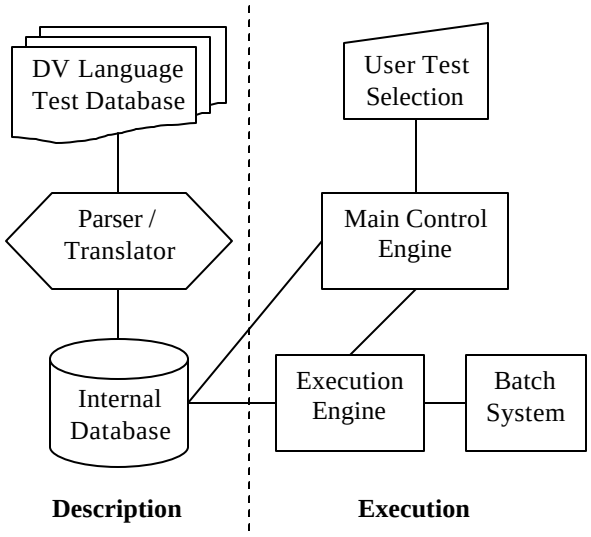


Figure 2: DV Architecture

The architecture is divided into two major pieces: one for describing tests and one for executing tests. Integral to the architecture is a database language (the DV language) that captures the testing process in an automatable way. This language is designed to make it easy for users to describe hundreds of tests for each block of a design and to account for the individual personalities of the blocks. A parser / translator converts the database for each block into a binary format that can be used during execution. Manual input from the user selects which tests to run and how to run them. The main control engine then extracts test information from the internal database and passes tests to the execution engine. The execution engine then interacts directly with the host system to run the flow or with an external batch system that manages resources throughout a network. The execution engine also interacts with the internal database to obtain parameters for the individual steps of a flow.

5.1 DV Language Descriptions

The DV language can describe four different kinds of things in the test database. They are configurations, tools, flows and tests.

Configurations describe the structure of the design under test. They define which source to include for blocks and for special stimulus and monitoring code. The configuration description for the example might look like this:

```
// Describe structure.
conf c1
  block1 = rtl
  block1_mon = live
endconf
```

Each configuration has a name followed by a list of component / view pairs. Users can define multiple configurations in the test database. For example, during gatelevel testing, the original configuration can be modified to use a synthesized netlist:

```
// Derived configuration.
conf c2 : c1 +=
  block1 = netlist
endconf
```

Here a copy of the RTL configuration has been used as a template for the netlist configuration. The new configuration then redefines the view for component 'block1'. The ability to derive things from pre-existing ones is a powerful mechanism for organizing a test suite.

Tools and flows describe the actions that should be taken when running a test. A tool is a detailed description of a job defining where and how to run it. Each tool must specify a directory to run in and a command to run. In addition, most tools need arguments to control the command's behavior. The descriptions for two of the tools from the example would look like this:

```
// Describe tools.
tool make_lib
  dir = $TOP/lib
  cmd = make
endtool

tool chk_sim
  dir = $TOP/test
  cmd = chk_sim.csh
  args  = -test $test
  args += –conf $conf
endtool
```

Notice that the 'chk_sim' tool has been parameterized by using DV variables so that it can be used for different tests. The 'args' property employs one of several operators that can be used to edit an existing property value.

A flow is a list of tools defining the order to run them in. Sequential execution, parallel execution and a limited form of branching are allowed. A flow can fork and rejoin as needed, and can call other flows as subroutines. The complete flow for the example is:

```
// Describe flow.
flow default
  make_lib
  par
    seq
      make_gen
      run_gen
    endseq
    bld_sim
  endpar
  run_sim
  chk_sim
endflow
```

Nested parallel / sequential blocks are used to identify that the generator can be built and run while the simulator is also being compiled. Due to the complexity of its syntax, flows cannot be derived from templates or edited the same way configurations and tools can.

Finally, tests are used to tie structure to action to form a complete test case. The example test description looks like this:

```
// Describe test.
test t1
  conf = c1
  flow = default
  when = daily
endtest
```

A configuration provides the what; a flow and its tools provide the how and where. An additional property 'when' is used as a key for selecting tests to run as part of regressions. A user can define many such keys as a way of organizing the complete set of tests into related groups.

5.2 DV Execution

The DV tool reads and executes tests from the database as selected by the user. The user chooses which tests to run via command-line options. Tests are organized into blocks, with each block database typically containing the tests needed to validate the design block of the same name. The user specifies a block and then a list of tests to run for that block. Alternatively, the user can supply a selection expression which is used to query the contents of the database to find tests that satisfy the expression:

```
# Select one or two tests by name.
1 > dv block1 t1 t2
# Select test with an expression.
2 > dv block1 –where when=daily
```

The main control engine parses the command-line options, assembles a final list of tests to run by consulting the database, and then sends the tests one by one to the execution engine.

The execution engine reads the contents of a test's flow and executes the tools in the order encountered. Procedures for both interactive or batch execution of tools are provided. Special care is taken to avoid running the same tool more than once with the same context.

6 DV Implementation and Use Experience

The DV architecture has been implemented as a pair of cooperating programs running on Sun Solaris and Microsoft Windows NT. The user interface, control and execution engines and the internal database are written in approximately 4000 lines of Perl. A scripting language was chosen over a systems programming language such as C/C++ in order to enable rapid development and evolution of the tool. However, for speed and implementation efficiency reasons, the parser / translator was written as a 1000 line lex / yacc / C program. It converts the DV database to a Perl script that is then invoked by the control engine to set up the internal database. The language description is written in a loose way so that new properties can be attached to the database objects without having to change the language. This makes it easier to implement new features in the Perl program. On Solaris, the execution engine has been interfaced to the LSF batch queuing system to provide parallel execution support.

The original DV implementation was used to verify the 3D graphics sections of the Flipper ASIC, the SOC core of the Nintendo GAMECUBE console. While initially targeted at block-level RTL functional verification, the DV databases were extended by adding configurations and flows to support chip-level verification as well as gate-level verification.

6.1 Phased Introduction

Many of the features of the language and tool were not part of the initial deployment. The requirements for the tool were extended on an ongoing basis as the Flipper project progressed. The basic interactive execution capability was available at rollout, and engineers started using the tool. A major update to add batch execution mode was needed as more tests were written. Parallel execution was added in order to improve performance when using the batch system. In order to support multiple regressions, the expression-based database query system was introduced. Deployment of new features was relatively painless because of DV's plain-text database format and the Perl implementation of the engines.

6.2 Process Capture

One of the common drawbacks of verification scripts is a proliferation of arguments and environment variables that modify the behavior of individual tools. This makes it difficult to transfer verification to other engineers, since one must also pass along these custom settings. DV resists this approach by encouraging users to write a short

DV database that modifies the default behavior. Here is an example of this feature used to enable waveform dumping during simulation of the example test:

```
test t1 +=
  tool.bld_sim.args += -pli_waves
  tool.run_sim.args += -wave_dump
endtest
```

The details of how to enable dumping are encapsulated inside this short database file. The default test description is retrieved and modified by this script. Test-specific tool arguments are used by the execution engine to temporarily modify the tool's execution. It is also possible instead to modify the default tool descriptions so that a set of tool changes applies to all tests.

6.3 Finding Patterns

One of the benefits of capturing tests in a database is the ability to search for patterns in the database. Identifying patterns can influence future efforts in two ways. First, awkward patterns signal a deficiency in the scope of the database language. By analyzing such a pattern, modifications can be made to the language to make it simpler to express the pattern. For example, early DV databases contained the following pattern:

```
// Base test.
test t =
  conf = c1
  flow = default
endtest

// First derived test.
test t1 : t +=
  tool.test_run.args = "-obj 1"
endtest

// Second derived test.
test t2 : t +=
  tool.test_run.args = "-obj 2"
endtest
```

This method for defining a series of related tests was quite common. However, because each new test requires four lines in the database, it is cumbersome to add a new test or maintain large sets of tests. In response, the language and tool were modified to allow a single-line format for this pattern:

```
// Derived tests.
test t1 : t += args = "-obj 1"
test t2 : t += args = "-obj 2"
```

The matching 'end' keyword can be omitted when modifying a single property, and the alias 'args' has been introduced for the commonly occurring 'tool.test_run.args' property.

Second, good patterns can be documented and communicated to other members of the team. For example, the following pattern can be used to derive a series of tests of escalating functional complexity:

```
// Escalating difficulty series.
test t1 : t  += args =  -func 1
test t2 : t1 += args += -func 2
test t3 : t2 += args += -func 3
```

Each new test is derived by copying its predecessor and enabling a new function. When fed input from a random generator, this series can test the interactions of each new feature against all previous ones.

7 Conclusions

A verification flow management tool called DV has been developed as a natural extension of the methods used to bring a fragmented design environment under control. The main benefit of such a tool is the ability of verification engineers to automatically run the thousands of tests required to ensure that an ASIC design is bug-free. The DV tool is architected on an underlying model of design verification that encompasses both the design structure and the verification flow. At the same time, its implementation is guided by the constraints of a startup company in a way that allows it to evolve with the company. By capturing the verification process in an executable format, it has become a foundation tool for ASIC design projects.

8 Acknowledgments

The author would like to thank the ArtX/ATI Flipper ASIC development team, the first users of the DV verification tool described in this paper. Their contributions in the form of ideas and idioms were invaluable to its early development.

9 References

[1] J.B. Brockman, *A Schema-Based Approach to CAD Task Management*. Ph.D. Thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering (Report CMUCAD-93-01), Jun. 1992.

[2] Platform Computing, Inc., *Platform LSF 5 Product Brochure*, URL: http://www.platform.com

[1] L. Bening and H. Foster, *Principles of Verifiable RTL Design*. Kluwer Academic Press, 2001.

Keynote 2:

Design Challenges & Solutions for 90nm/130nm Technology

David Lan (*TSMC*)

Design Challenges & Solutions for 90nm/130nm Technology

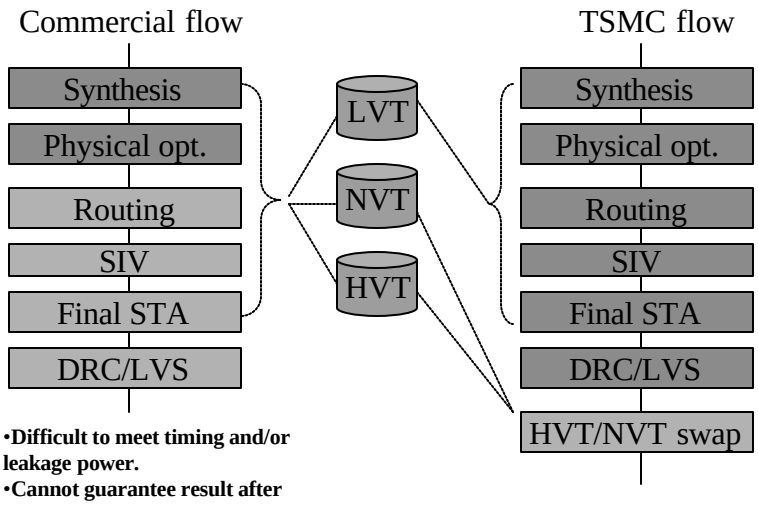
Contents

- Design Challenges for 90nm/130um process
  - Higher leakage current
  - Signal integrity concern
    - Xtalk
    - De-coupling cell insertion
  - DFM rules
    - Process variation modeling
    - Double via
    - Dummy metal insertion
    - LOD effect modeling
- TSMC Reference Flow 4.0
- Summary

Design Flow to Minimize Leakage Power

- Leakage power and dynamic power are of the same order in 90nm technology.
- Multi-Vt core cell libraries can be used for performance and leakage power optimization.
- Use only the fastest library during all phases of flow to simplify handling and allow tool to fully optimize for timing and area
  - Difficult to achieve fastest timing if several libraries are used during synthesis and optimization.
- After final routing and placement, perform swapping of cells to NVT/HVT without causing new timing violations

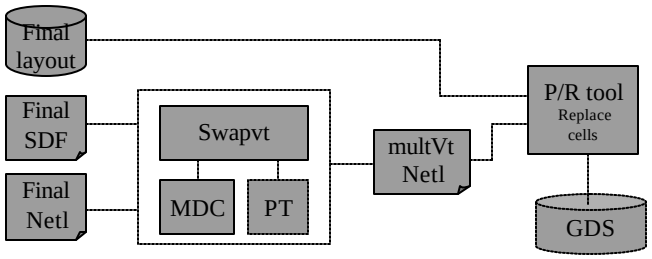
Design Flow to Minimize Leakage Power (Cont'd)



•Difficult to meet timing and/or leakage power.
•Cannot guarantee result after post-route iterations

Design Flow to Minimize Leakage Power (Cont'd)

- Advantages
  - No need to consider multiple libraries. Put all effort on optimizing area and timing
  - Simplifies implementation flow
  - Simple task to replace cells after final routing since it will not affect layout



Multi-Vt Flows Comparisons

2 DC iterations, long run time (5~10X longer)

	Lvt	Lvt -> Hvt --> 3vt (iteration 1)	Lvt -> Hvt --> 3vt (iteration 2)	Lvt swap Hvt/Nvt with our script		
Leakage power (nW)						
before P&R	**19.8**	**6.25**	**4.6**	**4.3**		
after P&R	**19.8**		**6.9**	**4.3**		
Instance Count (%)						
Hvt		43%	44%	65%		
Nvt		15%	23%	26%		
Lvt	100%	42%	33%	9%		
Timing (WNS)						
before P&R	0	0	0	0		
after P&R	0		-0.4	0		

A 90nm Test Case: ~100K cell instances with TSMC90 multi-vt libraries

Leakage and Performance Trade-off using Multi-Vt cells

A 90nm Risc Core

Vt options	Power (mW)		Frequency (mHz)	Cell distribution		
	Leakage	Dynamic		HVT	NVT	LVT
LVT	21.6	123.8	360			100%
LVT/NVT*	10.5	106.2	360		72%	28%
LVT/NVT/HVT*	9.7	105.5	360	43%	27%	28%
NVT	3.6	92.9	280		100%	
NVT/HVT*	2.2	91.5	280	73%	27%	
HVT	1.3	90.7	200	100%		

* TSMC Multi-Vt implementation approach was used

Note! No cells in paths with less than 0.3ns margin was swapped to NVT or HVT cells. If less margin is used, even more saving in leakage power can be achieved.

Signal Integrity Flows

- Xtalk
  - Prevention
  - Analysis
  - Violation fix
- De-coupling cell insertion
  - Capacitance calculation
  - De-coupling cell insertion

SI: Xtalk Prevention

- **Placement and optimization stage**
  - Set max transition (0.6 ~ 0.8 ns) to prevent weak victims
  - But sharp transition (0.4 ns) may create strong attackers
- **Post-placement stage**
  - FE: slew balance, congestion removal
  - Astro: gate up-sizing and buffer insertion for potential victims
- **Routing stage**
  - Nanoroute: global, track assign and detail route prevention
  - Astro route: global route and track assign prevention
- **Prevention effects**
  - Routing is more effective than placement in xtalk prevention
  - Can reduce xtalk violations 50% or more

SI: Xtalk Analysis

- Use CeltIc as our xtalk sign-off engine
- Library accuracy
- RC accuracy
- Design related information
  - Get timing window from PrimeTime
  - Identify constant nets and hazard-free net (spare cell nets, scan-in)
- Multi-Vt cells impact:
  - Xtalk sensitivity: HVt > StdVt > LVt

SI: Xtalk Fixing

- Flow 1: CeltIc -> FE -> Nanoroute -> CeltIc loops
  - Nanoroute is efficient in fixing glitch violations
  - FE is efficient in fixing timing violations
- Flow 2:CeltIc -> Astro-inroute -> CeltIc loops
  - Script for translating CeltIc violation report to scheme
  - Difference between CeltIc and Astro-xtalk:
    - Timing window, constant/floating nets, small cap filter
- Most Xtalk glitch and timing violation fixed in around 3 iterations.
- Manual script is used to fix remaining violations
  - High-strength victim nets (driven by D4 and larger cells)
    - wire spacing
    - buffer insertion
  - Low-strength victim nets (smaller than D4)
    - cell size up

Example Using Flow 1 (0.13u., 2M gates)

An example to show good noise convergence

	# of glitch violations	# of timing violations
Iteration 1	1437	622
Iteration 2	116	112
Iteration 3	18	40
Final Clean-up	0	0

De-coupling Cell Insertion

- De-coupling cap. cells are used to reduce dynamic IR drop on chip.
- C = P / f * V * ? V
  - C: total decoupling cap
  - P: total power consumption
  - f: operating frequency
  - V: operating voltage
  - ? V: voltage noise
- Decoupling cell cap =

  total decoupling cap – existing cap
- Existing cap sources:
  - Net cap, pin cap, power net cap
  - Internal pin cap, diffusion cap, nwell cap

Placement of Decoupling Cells

- Evenly distributed on power strap grids
  - Easy to implement and less effective on dynamic IR reduction
- Distributed based on power consumption
  - High power area
  - Clock buffers

DFM Scope and our Coverage

- DRM issues:
  1. Process variation modeling
  2. Metal over via/contact enclosure
  3. Redundant via insertion
  4. Dummy OD, Poly, and Metal insertion
  5. Poly/OD shape, orientation, dimension, and spacing
  6. Others

DFM: Interconnect Metal Process Variation Modeling

- Inter and intra-die process variation modeling
- Intra-die process modeling
  - Resistivity, metal width and thickness are function of width, spacing, and density.
  - Sample of resistivity, metal width and thickness adjustment table

	Width					
Spacing	0.20	0.40	0.60	1	2	3
0.21	2.653					
0.24	2.698					
0.42	2.339					
0.63	2.304					

S\W	0.20	0.40	0.60	1	2	3
0.21	0.389	0.389	0.389	0.389	0.389	0.389
0.24						
0.42						
0.63						
0.84						

S\W	0.20	0.40	0.60	1	2	3
0.21	0.2275	0.4275	0.6275	1.0275	2.0275	3.0275
0.24	0.2275	0.4275	0.6275	1.0275	2.0275	3.0275
0.42	0.2275	0.4275	0.6275	1.0275	2.0275	3.0275
0.63	0.2275	0.4275	0.6275	1.0275	2.0275	3.0275
0.84	0.2275	0.4275	0.6275	1.0275	2.0275	3.0275

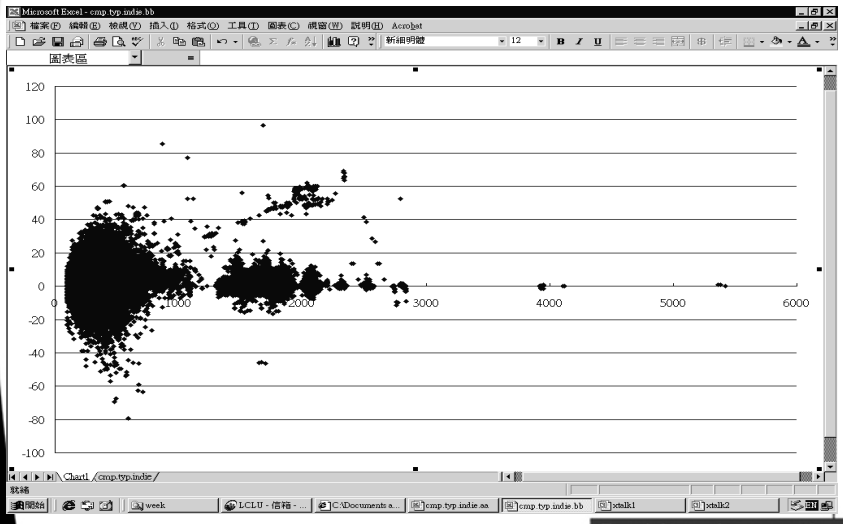
Resistivity

Thickness

width

Path Delay Comparisons of a 0.13um Example with and without Intra-die Metal Variation Modeling

x: path delay(ps), y: path delay diff (ps)



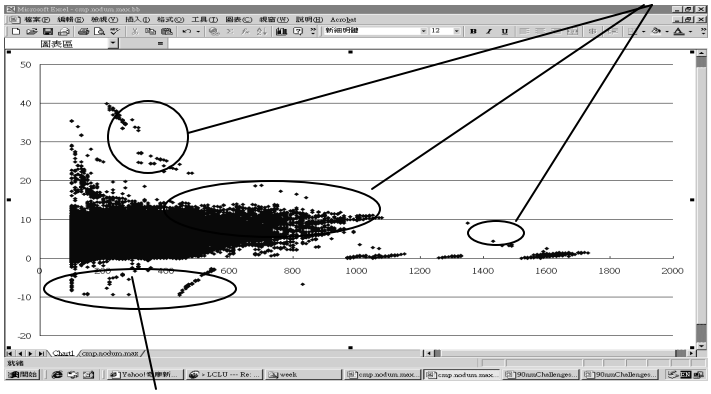
DFM: Inter-Die Metal Process Variation Modeling

- Process corners: too many combinations
  - Metal width (W), Metal Thickness, IMD thickness(IMD)
  - Which ones ?
    - **Cmax**(Wmax, Tmax, IMDmin), **Cmin**(Wmin, Tmin, IMDmax)
    - **RCmax**(Wmin, Tmin, IMDmin), **RCmin**(Wmax, Tmax, IMDmax)
  - Are all metal layers varied to the same corner?
- Current work-around for interconnect corner modeling:
  - Use extra margin for interconnect R and C
  - Use RC max/min corners
  - Use RC max/min and Cmax/min corners

A 90nm Example to Show Metal Process Corner Impact on Path Delay (Cmax corner vs. Typical)

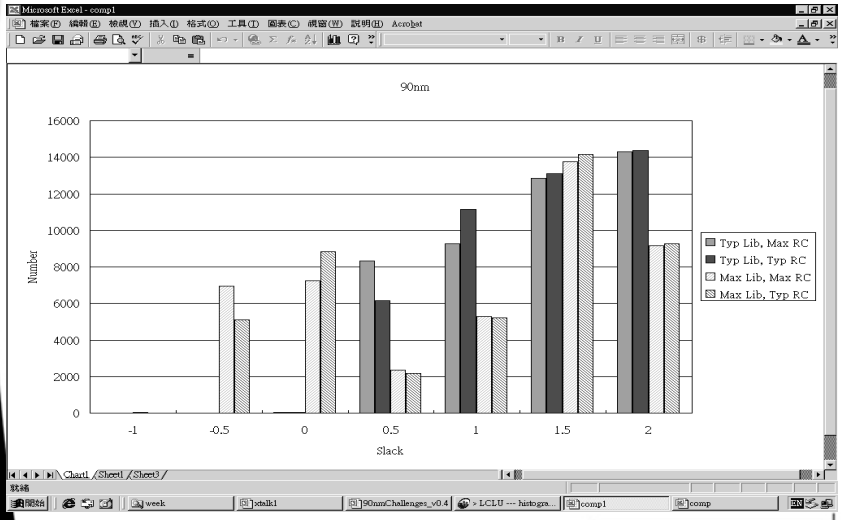
x: path delay(ps), y: path delay diff (%)

Design Margin if typ. Corner is used

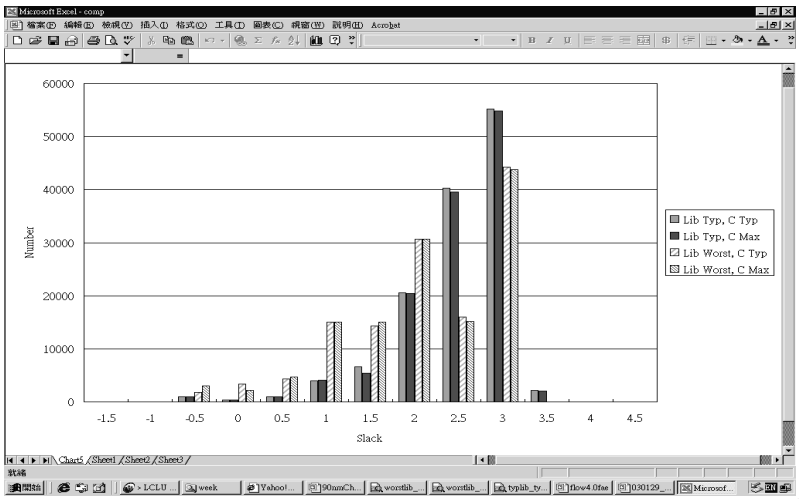


Extractor's accuracy limitation

Delay Impact Comparisons of library and Interconnect corners: A 90nm Example



Delay Impact Comparisons of library and Interconnect corners: An 130nm Example



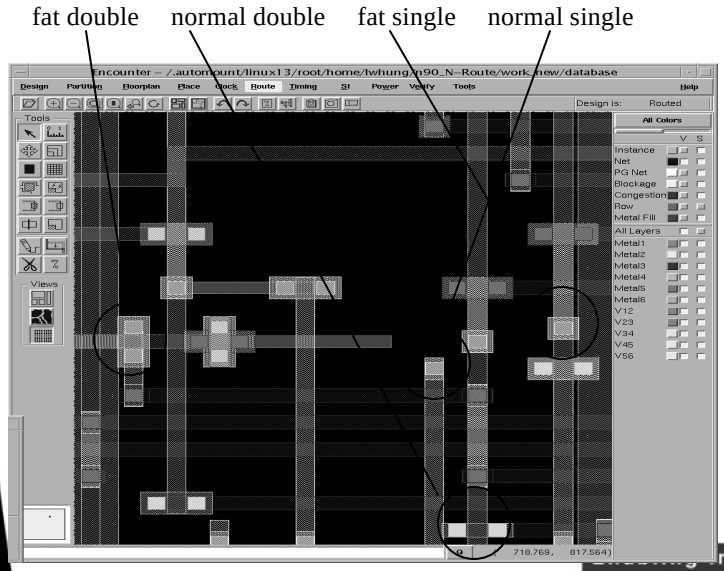
DFM: Redundant Via Insertion methodology

- In addition to add double vias on wide metal, suggest add redundant vias on normal signal for yield improvement.
  - Both Astro and Nanoroute support this feature now and are used in our 90nm projects.
- A 4-stage redundant via insertion flow:
  - fat double via > normal double via > fat single via > normal single via

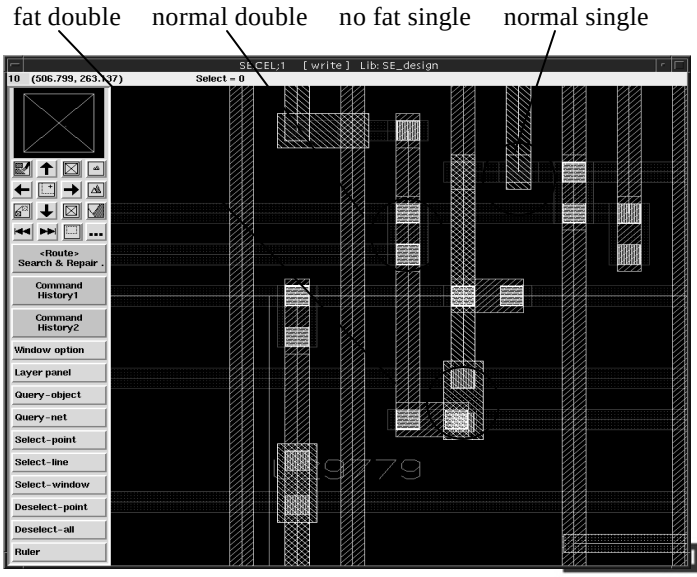
DFM: Redundant Via Insertion methodology (Cont'd)

- Redundant via insertion flow:
  - Define 4 via types in P&R tech. files
  - Step 1: route with single via
    Fat single via first if DRC OK
    Then, normal single via
  - Step 2: swap single via to double via
    Fat double via first if DRC OK
    Then, normal double via

DFM: Double Via Insertion in Nanoroute



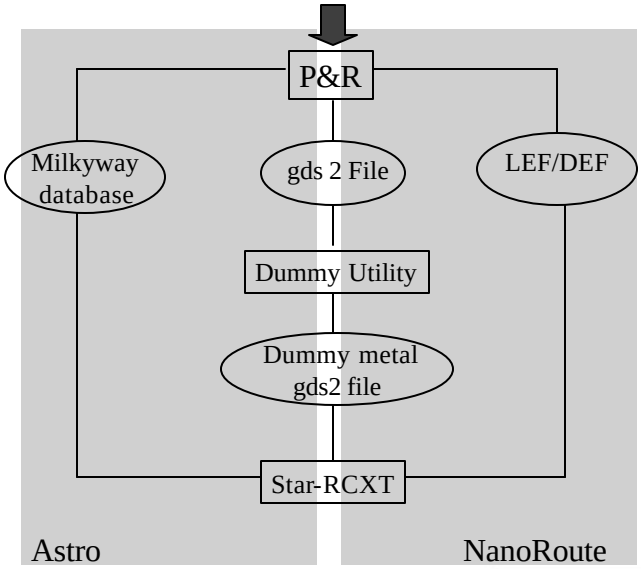
DFM: Double Via Insertion in Astro



DFM: Dummy Metal Insertion

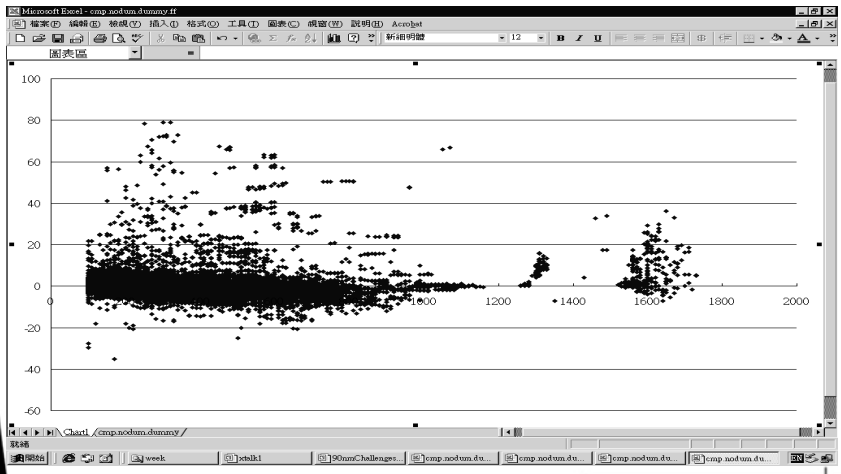
- Dummy pattern selection: size and spacing
  - must meet metal density requirement
  - minimize OPC impact
  - minimize induced capacitance
- Dummy metal flow:
  - Insertion of dummy metal
  - Induced delay analysis
  - Final DRC check
- Add dummy metal as a post-processing step
- Dummy metal induced capacitance can be extracted and verified in final timing check.

DFM: Dummy Metal Insertion Flow



A 90nm Example of Dummy Metal Impact on Delay

x: path delay(ps), y: path delay diff (ps)

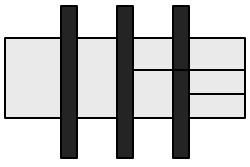


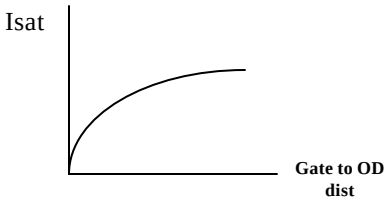
DFM Analyzer

- Cont/Via enclosure by metal.
- Poly corner rounding effect (“T’ shape).
- OD corner rounding effect.
- Cont. position and number affects Idsat.
- OD resistor increasing while gate is too close.

LOD Effect Modeling

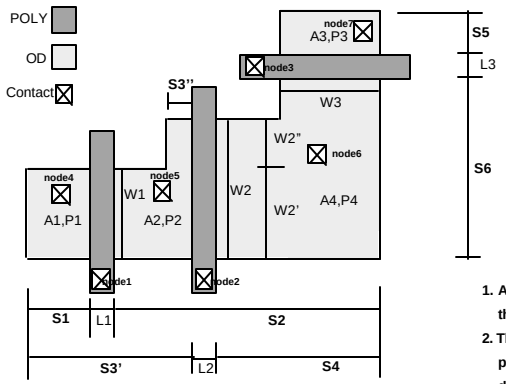
- Gate to diffusion edge distance must be extracted to model STI stress effect in design.
  - Spice model: available for early user now
  - Extractor: available now in Hercules/Star-rctx and Calibre/xCalibre flows





I_{sat} is a function of gate to OD edge spacing.

LOD Effect Modeling (Cont'd)



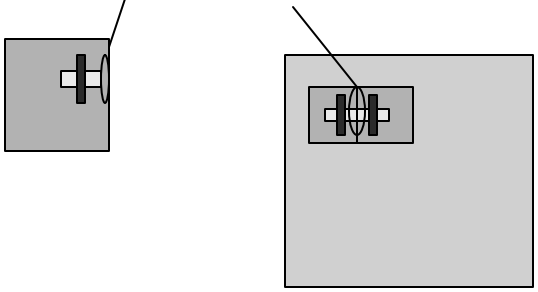
1. Add more instance parameters in the device.
2. The number of additional instance parameters is not a constant and is dependent on the shape of layouts.

Extracted Netlist of The Above Layout:

```
M1 4 1 5 0 n_ch L=L1 W=W1 AD=A1 PD=P1 AS=A2/2 PS=P2/2 SA=S1 SB=S2
M3 6 3 7 0 n_ch L=L3 W=W3 AD=A4/2 PD=P4/2 AS=A3 PS=P3 SA=S6 SB=S5
M2 5 2 6 0 n_ch L=L2 W=W2 AD=A2/2 PD=P2/2 AS=A4/2 PS=P4/2 SA=S3' SB=S4 SW=W2' SA=S3'' SB=S4 SW=W2''
```


LOD Effect Modeling (Cont'd)

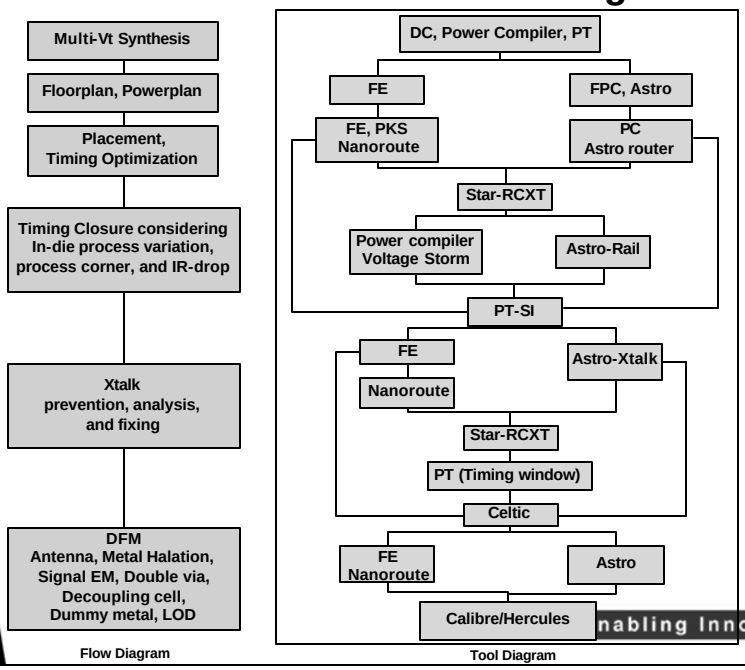
- IP design
  - Reserve space margin from OD to cell boundary to avoid OD abutment in chip placement.
- Chip integration
  - Make sure no OD abutment after cell placement.



TSMC Reference Flow 4.0

- Previously presented design solutions for noise, dynamic IR, DFM, and leakage power optimization flows will be integrated in TSMC Reference Flow 4.0
- Available on TSMC ONLINE in early May 2003.
- Several 90nm and 130nm chips were taped out using TSMC Reference Flow 4.0 + TSMC 90nm/130nm core cell libraries.
- Working on high frequency CTS design, flip chip area array flow, voltage island methodology now.

TSMC Reference Flow 4.0 Diagram



Session 6:

Methodologies of Industrial Usage of Formal Verification

Moderator: Sandeep Shukla (*Virginia Tech*)

Property Specification: The key to an Assertion-Based Verification Platform

Harry D. Foster
Chief Architect
Verplex Systems, Inc.
harry@verplex.com

Abstract

Assertion-based verification—that is, user specified properties and automatic property extraction combined with simulation and formal techniques—is likely to be the next revolution in hardware design verification. This paper explores a verification break-through prompted by multi-level specification and assertion verification techniques. The emerging Accellera formal property language, as well as the Open Verification Library standards and the important roles they will play in future assertion-based verification flows are discussed. Furthermore, automatic property extraction techniques are explored—and their important roles in validating semantic consistency in the context of an RTL signoff flow.

1 Introduction

A change is taking place in the way we design and verify our designs that will revolutionize the industry and result in the equivalent of a synthesis productivity breakthrough in verification. This change demands that we move from natural language forms of specification to forms that are mathematically precise and verifiable, and lend themselves to automation. Property specification is the key ingredient of this revolution, whose end result is improved verification through an *intelligent testbench*. An assertion-based verification platform is an integral part of an intelligent testbench, which consists of the following key components:

1. ***verifiable testplans*** through property specification (that is, functional coverage models),
2. ***hardware verification languages*** (HVLs) combined with property specification to raise the abstraction level of testbench generation
3. ***reactive coverage driven testbenches*** based on property specification (assertions and functional coverage),
4. ***automated block-level methodologies*** such as smart block-level simulation stimulus generation based on interface properties (i.e., constraints),
5. ***exhaustive formal verification*** and ***semi-exhaustive property checking*** techniques.

This paper discusses the important role of *property specification* and automatic property extraction techniques in the context of an assertion-based verification flow.

1.1 Standards

One organization that works to support improvements in verification methodologies is Accellera (see www.accellera.org). Their mission is to drive worldwide standards that enhance a language-based design automation process. Recently, the Accellera Formal Verification Technical Committee completed the 1.0 LRM, which defines its property specification language (PSL) [Accellera 2003]. This *declarative* property language supports *top-down* (that is, functional specification-driven) design methodologies. Declarative property languages are ideal for specifying architectural and global properties, as well as defining interface specification during block-level partitioning.

In addition to PSL, the Accellera Assertion Committee has developed a standard for specifying RTL implementation properties directly within the designer's HDL through the *Open Verification Library* (OVL) [Bening and Foster 2001] and the new *SystemVerilog* procedural assertion construct proposal [Foster, et al. 2002]. The OVL provides a template for expressing a broad class of assertions *structurally* within the designer's RTL, while the new assertion construct facilitates expression of assertions *procedurally* during RTL development. Both the OVL and the new assertion construct enable *bottom-up* (that is, white-box) verifiable implementation practices, which improve simulation-based methodologies while providing a seamless path to formal verification.

Combined, these powerful and expressive formal property languages enable engineers to:

- specify properties as assertions and constraints for formal analysis,

- specify functional coverage models to measure the quality of simulation and identify holes,
- specify pseudo-random constraint-driven simulation environments, similar to the research by Yuan, et al. [1999]; and Shinmizu and Dill [2002].

1.2 Monitor-based specification

While standardizing a property language, such as PSL, is integral to addressing increased verification complexity, it is not the entire solution. Equally important to this revolution in design verification is an effective methodology that unifies traditional and formal verification within an assertion-based verification framework. Recently, monitor-based methodologies have emerged as a technique for unifying traditional and formal verification (for example, *FoCs-Automatic Generation of Simulation Checkers from Formal Specification* [Abarbanel, et al. 2000]). Other approaches include creating a protocol bus-monitor that examines an agent's output signals (as the monitor's input) and then generates a Boolean $correct_i$ output signal, which is true when agent *i* is compliant to the specification (for example, *Monitor-Based Formal Specification of PCI* [Shinmizu, et al. 2000] and *A Specification Methodology by a Collection of Compact Properties as Applied to the Intel Itanium Processor Bus Protocol* [Shimizu, et al. 2001]).

An effective unifying methodology includes the Accellera Open Verification Library (OVL), which provides the systematic elements of the methodology. The OVL incorporates a consistent and systematic means of specifying RT-level implementation properties structurally through a common set of assertion monitors. The OVL monitors act like a template, which enables designers to express a broad class of assertions in a common, familiar RTL format. Furthermore, the OVL capitalizes on the various Accellera assertion techniques by unifying the PSL *declarative* form of property specification with the new SystemVerilog (and VHDL) *procedural* form of specification within the library. Finally, these monitors address assertion-based methodology considerations by encapsulating a unified and systematic method of reporting, that can be customized per project, and a common mechanism for enabling and disabling assertions during the verification process. The reporting and enable/disable features use a consistent process, which provides uniformity and predictability within an assertion-based methodology.

2 Property specification

Informally, a *property* is a general behavioral attribute used to characterize (that is, a collection of logical and temporal relationships between and among subordinate Boolean expressions, sequential expressions, and other properties that in aggregate represent a set of behavior).

When discussing properties, it is generally easier to view their composition as three distinct layers:

- the *Boolean layer*, which is comprised of Boolean expressions (for example, Verilog or VHDL expressions)
- the *temporal layer*, which describes the relationship of Boolean expressions over time
- the *verification layer*, which describes how to use a property during verification

Defining (or partitioning) a property in terms of the abstract view (that is, layer structure) enables us to dissect and discuss various aspects of properties. However, it is actually quite simple to express design properties; the three-layer view is merely a way to explain.

A property's *Boolean layer* is comprised of Boolean expressions composed of variables within the design model. For example, if we state that "*signal* en1 and signal en2 are mutually exclusive" (that is, a *zero-or-one-hot* condition in which only one signal can be active high at a time), then the Boolean layer description representing this property could be expressed in Verilog as:

```
(!(en1 & en2)
```

Notice that we have not associated any time relationship to the statement: "*signal* en1 and signal en2 are mutually exclusive". In fact, the statement by itself is ambiguous. Is this statement *true* only at time 0 (as many formal tools infer), or is it *true* for all time?

To remove all time ambiguities, a property's *temporal layer* describes the Boolean expressions' relationships to each other over time. For example, if *signal* en1 and signal en2 are always mutually exclusive (that is, for all time), then a temporal operator could be added to the Boolean expression to state precisely when the Boolean expression must hold (that is, evaluate *true*). This could be written in PSL as follows:

```
always !(en1 & en2)
```

There are many temporal operators in PSL (including *always*, *never*, *next*, *eventually*, *until*), which permit us to reason about very complex temporal relationships that potentially involve multiple Boolean expressions. The Boolean layer combined with the temporal layer form the basis of the *property*.

While a property's Boolean and temporal layers describe general behavior, they do not state how the property should be used during verification. In other

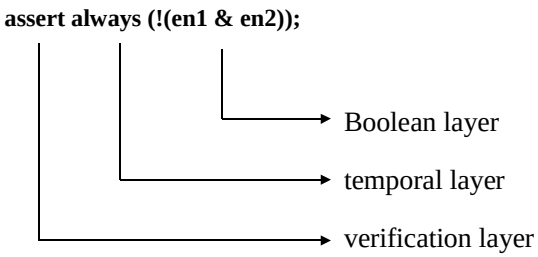
words, should the property be asserted, and thus checked? Or should the property be assumed as a constraint? Or should the property be used to specify an event used to gather functional coverage information? Hence, the third layer of a property, which is the *verification layer*, states how the property is to be used.

Consider the following definitions for an assertion and a constraint.

- **assertion:** A given property that is expected to hold within a specific design. An *assert* keyword could be associated with the property (depending on the property language).
- **constraint:** A condition (usually on the input signals) which limits the set of behavior to be considered. A constraint may represent real requirements (e.g., clocking requirements) on the environment in which the design is used, or it may represent artificial limitations (e.g., mode settings) imposed in order to partition the verification task.

Look again at *signal* en1 and signal en2. We can specify this property as an assertion in PSL using the *assert* keyword as shown in Figure 1. This states that the property is to be treated as an assertion during verification.

Figure 1: Multiple layers of a PSL assertion



2.1 Declarative versus procedural specification

Assertions (or constraints) may be expressed either *declaratively* or *procedurally*. A declarative assertion is always active, and is evaluated *concurrently* with other components in the design. A procedural assertion, on the other hand, is a statement within procedural code, and is executed *sequentially* in its turn within the procedural description. Hence, declarative properties, such as PSL, are natural for specifying block-level interfaces, as well as other properties that must hold concurrently within a system. Similarly, a procedural assertion, such as the new SystemVerilog assert construct, is convenient for expressing algorithmic properties that must hold in the context (and sequential scoping) of procedural code.

For example, using the Accellera *declarative* formal property language PSL, the designer could express that the 8-bit bus *cntrl*[7:0] must possess the property of zero or one-hot as shown in Example 1, below.

Example 1: PSL declarative assertion

```
assert always ((cntrl & (cntrl – 1))==0) @(posedge (clk))
```

In Example 2, the zero or one hot assertion (using the new Accellera SystemVerilog *procedural* assert construct) is embedded directly within some procedural code.

Example 2: SystemVerilog procedural assertion

```
always (en or cntrl) begin
  if (en)
    assert @( posedge clk) ((cntrl & (cntrl – 1));
end
```

A key difference between the *declarative* assertion (shown in Example 1) and the *procedural* assertion (shown in Example 2) is that the declarative assertion concurrently monitors the assertion expression, while the procedural assertion only validates the assertion expression during sequential visits through the procedural code. If the 'en' signal is never TRUE in Example 2, then the procedural assertion will never be validated. This might be the intended behavior. However, care must be taken when assertions are deeply nested within **case** and **if** constructs in procedural code to prevent over constraining the assertion expression.

Alternatively, if the 'en' signal is a required condition on the assertion, then a declarative assertion must include this as part of the assertion expression. For example, we would require Example 1 to be re-written as shown in Example 3.

Example 3: PSL declarative assertion.

```
assert always (en -> ((cntrl & (cntrl – 1))==0))
      @(posedge clk);
```

To continue this progression, Example 4 demonstrates how to *structurally* express the same property shown in Example 3 using the Open Verification Library:

Example 4: OVL structural assertion.

```
  assert_zero_or_one_hot #(0,8) (clk, en, cntrl);
```

Example 5: FIFO Over and Underflow OVL Assertion

```
module fifo (clk, fifo_clr_n, fifo_reset_n, push, pop, data_in, data_out);
  parameter fifo_width = `FIFO_WIDTH;
  parameter fifo_depth = `FIFO_DEPTH;
  parameter fifo_cntr_w = `FIFO_CNTR_W;
  input clk, fifo_clr_n, fifo_reset_n, push, pop;
  input [fifo_width-1:0] data_in;
  output [fifo_width-1:0] data_out;
  wire [fifo_width-1:0] data_out;
  reg [fifo_width-1:0] fifo[fifo_depth-1:0];
  reg [fifo_cntr_w-1:0] cnt; // count items in FIFO
 . . . .
// RTL FIFO Code Here
 . . . .
 `ifdef ASSERT_ON
// OVL Assert that the FIFO cannot overflow
     assert_never no_overflow (clk,(fifo_reset_n & fifo_clr_n),
                      ({push,pop}==2'b10 && cnt==fifo_depth));

// OVL Assert that the FIFO cannot underflow
     assert_never no_underflow (clk,(fifo_reset_n & fifo_clr_n),
                      ({push,pop}==2'b01 && cnt==0));
 `endif
endmodule
```

2.2 Top-down versus bottom-up specification

The previous section introduced the work of Accellera that promotes improved top-down and bottom-up verification methodologies with the PSL declarative property language and the OVL and SystemVerilog procedural constructs. This section discusses a usage model for combining these multiple forms of specification.

In a specification-driven design and verification methodology, the design architect begins by creating a high-level specification for key characteristics of the design (such as communication protocols or complex arbitration schemes) using a declarative property language like PSL. This enables specification consistency-checking prior to design and implementation, and eliminates architectural bugs.

As the architect partitions the design (top down) into block-level components, prior to RTL implementation, block-level interfaces should be specified to form verifiable contracts between multiple block-level components. The Accellera PSL formal property language declarative form of specification is ideal for this type of specification. These formal interface specifications serve as constraint models during block-level formal analysis, as well as interface monitors during simulation. Furthermore, through this process, interface misconceptions between multiple engineers are resolved prior to RTL coding.

As the engineers begin RTL-development, they should take advantage of *assertions* and add this form of specification for any potential corner case concerns. For example, using the OVL assertions the designer can specify that a queue or FIFO will never *over* or *underflow* (see Example 5)—or that a set of signals is *one hot*—or that a specified expression (condition) will never occur. Engineers should embed these specifications directly within the RTL.

The declarative block-level interface specification (developed during top-down specification) combined with RTL implementation assertion (created during bottom-up RTL development) facilitates block-level formal analysis. Similarly, the designer will leverage the block-level interface specification in the future for pseudo-random constraint-driven simulation vector generation.

3.0 Automatic property extraction

Commercially available formal verification tools are emerging as a key advancement in the design verification revolution. These tools enhance verification methodologies by automatically extracting many design properties from the engineer's HDL model. The tool then exhaustively verifies the properties using formal techniques. This enables the engineer to verify many design properties early in the design cycle without the need to create testbenches and test vectors. The following sections provide examples of properties that can be automatically extracted and verified by tools. This set of properties includes bus contention, bus floating, set/reset conflicts, RTL (Verilog) X-assignment "don't care"

checks, full or parallel case "don't care" checks, and clock-domain crossing checks.

3.1 Semantic consistency property checking

Transformation verification, using formal combinatorial equivalence checking, has become mainstream for many design projects over the last few years. This process is a critical component of the design and verification flow that ensures *logical consistency* between transformed models within the flow. Transformation verification minimizes the need for gate-level simulation. However, when creating an RTL signoff methodology, the process of ensuring *semantic consistency* between the pre- and post-synthesis model must be addressed as well [Bening and Foster 2001].

The following definitions offer some insight into the significance of addressing logical and semantic consistency and X-state optimism.

Definition: Logical consistency
Referenced and revised design models are *logically consistent* if their combinational logic cones driving the next state latches and the output ports are functionally equivalent, modulo their don't-care space.

Definition: Semantic consistency
Referenced and revised design models are *semantically consistent* if, for all possible sequences of input vectors, the simulation results observed at all the latches and output ports are the same.

To illustrate the impact of semantic consistency, consider a particularly insidious situation in which design errors cannot be demonstrated during RTL simulation, yet are easily revealed using gate level simulation for logically equivalent circuits [Bening and Foster 2001]. This occurs because it is possible for an RTL and gate-level model to exhibit *logical consistency*, yet behave *semantically inconsistent*. That is, an equivalence checker proves that the two circuits are equivalent from a Boolean perspective. However, the pre- versus post-synthesis simulation results differ. To comprehend this phenomenon requires an understanding of RTL X-state optimism, which is explained below.

Definition: X-state optimism
Optimistically exercising the **default** (or **else**) branch, from multiple alternative branches, within a procedural **case** (or **if**) statement due to an X evaluation of the statement's test expression.

The following Verilog code demonstrates RTL X-state optimistic behavior:

```
case (d)
  2'b01:   e = 2'b10;
  2'b10:   e = 2'b01;
  default: e = 2'b00;
endcase
```

If '*d*' evaluates to *2'bXX*, the case statement will optimistically evaluate the default branch, assigning *e* the known value *2'b00*. Given an *XX* initialization state for *d*, then only one of the four possible branches during the start-up condition is tested, which can result in missing a functional bug.

The functional class of bugs hidden by RTL X-state optimism is generally the result of coding styles that yield semantic inconsistency, and include: (a) X assignment usage; (b) bit-vector select range overflow; and (c) synthesis pragmas, such as *full_case* and *parallel_case*. Semantic inconsistency problems for (c) are discussed in the following section.

3.2 Full case semantic problem

Example 6 illustrates a full case semantic problem. Examine the following RTL code, which contains a *full_case* synthesis pragma.

Example 6: RTL code for one-hot mux

```
module mux (a,b,S,q);
  output      q;
  input       a, b;
  input [1:0] s;
  reg         q;

  always @(a or b or s)
  begin
   case (s) //synthesis full_case
      2'b01: q = a;
      2'b10: q = b;
   endcase
  end
endmodule
```

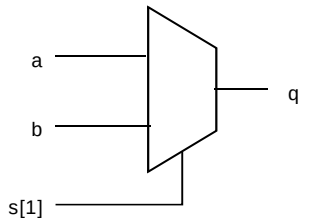


Figure 1: Synthesized gate for mux

Notice the RTL code in Example 6. If the *s* input variable ever assumes the unspecified values of *2'b00, 2'b11* or *2'bXX* (due to a functional bug in the logic

driving s), then the *q* output variable will improperly behave as a latch during RTL simulation (that is, retain its previous, valid assignment). However, the synthesized logic in Figure 1 does not contain a latch.

For Example 6 and Figure 1, notice the case when *s==2'b11*. The circuits are semantically inconsistent since the RTL *q* output simulates to *a*, while the gate *q* output simulates to *b*. However, these two circuits will prove to be logically equivalent, since the *full_case* pragma instructs the equivalence checker (and synthesis tool) to treat any value of *s* other than *2'b01* or *2'b10* as a don't care.

It is possible to automatically extract properties derived from the designer's HDL code, and then apply formal techniques to validate the designer's intent. For example, in Figure 1 the property—*'s' will never equal 2'b00 and 's' will never equal 2'b11*—can be automatically extracted from the RTL code, and then formally verified to ensure semantic consistency.

Note that some designers might be tempted to assign the q variable in example 6 a default value of X. However, this can result in optimistic behavior further down stream in the RTL during simulation—which means that a bug might go undetected. Automatic property extraction is a better technique for validating the safety of all X assignment, since formal verification will not encounter X-state optimism.

3.3 Parallel case semantic problem

Even if the **case** statements are fully specified, it is still possible to encounter semantic inconsistencies due to the general use of synthesis pragmas within the RTL. Example 7 demonstrates a dangerous semantic inconsistency problem when a *parallel_case* pragma appears in the RTL. The problem occurs if there is a functional bug in the logic outside the module, that generates illegal values for the variables *a* and *b* (for example, *{a, b} == 4'b1111*). The RTL simulation always behaves as a priority encoder, while the synthesized gate-level model would encode both *y* and *z* in parallel (since synthesis uses the pragma to identify *don't care* values for optimization). Hence, the functional bug could be missed during RTL simulation since the RTL model optimistically behaves as a priority encoder, while the gate-level model behaves differently (as shown in Figure 2).

In Example 7, the designer's intent can be formally verified by automatically extracting the property—*it is not possible for both 'a' to equal 2'b11 and 'b' to equal 2'b11*.

Example 7: RTL Code

```
module encoder (y, z, a, b);
  output       y, z;
  input  [1:0] a, b;
  reg          y, z;

  always @(a or b) begin
    {y, z} = 2'b00;
    casez ({a, b}) //parallel_case
      4'b11??: z = 1'b1;
      4'b??11: y = 1'b1;
    endcase
  end
endmodule
```

Examples 6 and 7 demonstrated semantic problems that you may encounter through pragma use. In addition to these problems, general variable X assignments can exhibit optimistic behavior when evaluated as a test expression within a case or if statement. This can result in semantic inconsistency between the RTL and gate-level models.

Ensuring semantic consistency is a key component within an RTL signoff methodology. Hence, automatic extraction and formal analysis of RT-level design intent properties complements Boolean equivalence checking to ensure that both *logical consistency* and *semantic consistency* are preserved during design flow transformations—minimizing the designer's dependency on gate-level simulation.

Figure 2: Synthesized Gates

5.0 Conclusion

In this paper we discussed the need for declarative, structural, and procedural mechanisms for expressing assertions. The emerging Accellera PSL, OVL, and SystemVerilog procedural assertions were introduced. Techniques for combining these languages were proposed, creating a top-down interface specification methodology with a bottom-up RTL implementation assertion methodology, which creates an effective assertion-based verification flow. This flow facilitates formal analysis, while improving traditional simulation methodologies. Finally, we demonstrated the important

role automatic property extraction (and verification) plays within an RTL-signoff methodology.

Formal verification technology will become a key component within an assertion-based verification framework. This framework will enable the designer to verify a set of user defined properties (specified with the new Accellera PSL and OVL standards)—while supporting automatic property extraction and verification techniques.

References

[Abarbanel, et al. CAV 2000] Y. Abarbanel, I. Beer, L. Glushovsky, S. Keidar, Y. Wolfsthal, "FoCs: Automatic Generation of Simulation Checkers from Formal Specifications," *Proceedings of the Computer-Aided Conference* (CAV), pp. 538-542, 2000.

[Accellera 2003] Accellera Property Specificatoin language http://www.eda.org/vfv/docs/psl_lrm-1.0.pdf

[Bening and Foster 2001] L. Bening, H. Foster, *Principles of Verifiable RTL Design*, Kluwer Academic Publishers, May 2001.

[Foster and Coelho 2001] H. Foster, C. Coelho, "Assertions Targeting A Diverse Set of Verification Tools," *Proceedings of the 10-th Annual International HDL Conference*, March, 2001.

[Foster et al. 2002] H. Foster, P. Flake, T. Fitzpatrick, "Adding Design Assertions to SystemVerilog", *Proceedings of the 11-th Annual International HDL Conference, March 2002*.

[Shinmizu et al. 2000] K. Shimizu, D. Dill, A. Hu, "Monitor-Based Formal Specification of PCI," *Proceedings of the Third International Conference on Formal Methods in Computer-Aided Design*, pp. 335-353, November 2000.

[Shimizu et al. 2001] K. Shimizu, D. Dill, C-T Chou, "A Specification Methodology by a Collection of Compact Properties as Applied to the Intel Itanium Processor Bus Protocol," In *CHARME'00*, Springer Verlag, pp. 340-354, 2001.

[Shimizu and Dill 2002] K. Shimizu, D. Dill, "Deriving a Simulation Input Generator and a Coverage Metric From a Formal Specification," *Proceedings of the 39-th Design Automation Conference*, June, 2002.

[Yuan, et al. 1999] J. Yuan, K. Shultz, C. Pixley, H. Miller, A. Aziz, "Modeling Design Constraints and Biasing in Simulation Using BDDs, " *Proceedings of the IEEE International Conference on Computer Aided Design*, pp. 584-589, November 1999

A Verification Synergy: Constraint-Based Verification

By Carl Pixley (Synopsys Inc.) and John Havlicek (Motorola, Inc.)

Introduction

Functional verification (as opposed to verification for timing, power, manufacturability and so forth) is a bottleneck in design. We know why this is so. IC's have become so complex that it is very difficult to specify and verify their behaviors. In the last ten years, the semiconductor industry has moved from directed simulation and directed random simulation, based solely on golden models, to more creative verification solutions, including comprehensive testbench tools, temporal assertion- and constraint-based verification, emulation, widespread use of automated Boolean equivalence checking, formal property checking and various hybrid schemes for verification.

It has also become obvious that verification technologies can have synergistic relationships. For example, we know that by writing constraint assertions in a certain way, they can be used as random simulation drivers for unit level verification and then can "flip" to become assertion monitors when units are combined with other units. These constraint assertions can also be used in a formal verification environment. There is a huge cost savings when information is used for more than one purpose. Historically, a similar synergy was discovered when it was realized that a subset of simulation language could be used for synthesis. This synergy was the basis for a revolution in IC design. But synthesis of models for verification (rather than design) is also attractive for getting emulation models running early in the design cycle. Typically, behavioral models of a design exist before detailed RTL models are available. Being able to synthesize and emulate these models is another attractive synergy. Several companies embed test generation and assertion monitoring in an emulator so that verification can proceed at emulation speed. Language can facilitate synergies of technology. The important point is that language is not just about features for a single use (e.g., simulation, formal verification or synthesis) but also about how it facilitates synergy among several technologies. For example, an ideal assertion specification language would facilitate use with formal, simulation, and emulation engines (and even synthesis engines!) and be convenient for designers to use. However, language without supporting algorithms is not very useful. Verification algorithms also benefit from technology synergies. For example, Binary Decision Diagrams (BDDs), Boolean satisfaction algorithms (SAT), ATPG algorithms, uninterpreted functions, word-level algorithms and so forth and so on have been combined in various ways to create hybrid systems superior to any individual algorithm.

These synergies are having profound consequences for how verification is done. One issue that always haunts the design community is the cost effectiveness of a design strategy. Currently, most companies do the maximum verification that they can afford – counting time to market, human effort, computer costs and so forth. Very few companies claim that any IC claim has been "totally verified" -- whatever that means. With very large expenditures some companies such as Intel have been able to totally formally verify certain key parts of their IC's, such as floating-point units and memories. One way to reduce the cost of verification is to share information among different tools and methods. This implies a "capture once, use repeatedly" strategy for precious design information.

Constraint-Based Verification

One example of verification synergy is constraint-based verification. To some degree, all commercial test bench tools allow constraints to be used in the generation of stimulus vectors during simulation. One way, pioneered at Motorola, is to capture constraints, which define the "environment" of a unit being verified. The idea was very simple, given that the design is in a certain state there is a set of vectors that are appropriate inputs for that state. At Motorola, we captured this as a Boolean formula, i.e., a constraint – that depends, of course, on the state of the design or of some useful auxiliary finite state machine. This was natural, since constraints were already built into our model checker, Verdict [kaufmann98], as a native capability. For example, we used constraints to define the valid set of initial states for model checking purposes. Actually, we used precisely the same constraint syntax for our Boolean equivalence checker, MET [park00], our switch-level extraction tool [jolly02] and other tools in our verification suite- another nice synergy. Since we used both formal model checking and informal, simulation-based verification, it was natural for us to use constraints for both. On the one hand this was obviously useful because we could use constraints to represent the environment of a unit for formal verification, such as SAT-based bounded model checking. And as an added benefit, such constraints (i.e., assertions formulas) could easily become assertion monitors when the DUV was connected up to its real chip environment. This became the basis of an

assume/guarantee methodology whereby assertions about a unit could be proven under constraint assumptions and then “flip” and become a property to check in a larger context.

However, how would we use these constraint assertions in simulation? There was no obvious method for taking Boolean formulas and generating solutions on the fly that would not excessively slow down simulation. We would have to solve a SAT problem (np-hard) each clock cycle of simulation! So we invented a tool called SimGen [pixley99, yuan99] that could be used for non-backtracking, on-the-fly stimulus generation. The idea was actually pretty easy: first we compiled the constraints into Binary Decision Diagrams (BDDs) involving state variables and input variables of the design. As everyone who has used BDDs knows, there is always the possibility of BDD blowup so we had to think of some clever ways to ameliorate that problem. However, if you can compile the BDDs, the rest is easy. At each clock cycle, the state of the design is sampled and then a “walk” of the BDD is performed in linear time to get a satisfying assignment of inputs that satisfies the constraints. One then drives the inputs to the design with the input values just calculated, toggles the clock and does the same thing next clock cycle. We’re leaving out lots of details but that is the general idea. It is possible to bias the inputs (even depending upon the state of the design!) so that one got a good mix of inputs [yuan99].

It was quite a breakthrough when we discovered a “biasing” scheme. This solved two problems: how to bias inputs to the design and (surprisingly) how to allow for efficient BDD ordering. We proved that the biasing scheme was independent of BDD order, which allowed us to interleave state and input variables in any order. Of course, the whole point of this research was to find a way to generate inputs to the design that satisfied the constraints and was not inordinately time consuming. SimGen’s overhead during simulation was usually far less than 20% based upon the complexity of the constraint set. It turned out that most of the time was spent in the PLI. These days, we all know a lot more about how to make the compile phase and the runtime much more efficient. The interesting thing is that when we tried the new scheme on real verification problems, it turned out to be amazingly effective in generating interesting corner cases. So what evolved is a methodology in which SimGen was used on module, block and unit levels of the design to quickly locate bugs. This was called “maturing the logic early”. Then when the bug rate fell off, we used our model checker -- *with precisely the same constraints* -- to find the more subtle bugs that were harder to find in any sort of simulation. Of course, there was the possibility of actually *proving* an assertion with the model checker as well. The advantage of unit verification is that one is not finding silly bugs during full chip or SoC verification. Also, constraints-as-checkers (i.e., assertions) were left all over the design as booby traps to catch bugs at their source. My dream is that various reusable parts, like bus interfaces, can be fully verified at the factory and that verification IP, in the form of assertions and constraints can be delivered with the design IP.

Another very important synergy concerns the language that is used to define assertions and constraints. Our colleagues at Motorola Israel devised a language call CBV (Cycle-Based Verilog) that is extensively used directly by designers to express properties about a design. As the name implies, CBV looks as much like Verilog as possible. For example, the expression language of CBV <u>is</u> the Verilog expression language. So learning time is very short. We have known designers to be writing useful CBV with a half day of training. The language is block structured. The block structure encourages the writing of a comprehensive specification about a block rather than just an ad hoc set of isolated assertions.

The goal of CBV was to give designers an intuitively easy way to write the kinds of properties that are of the most practical interest. Most practical properties are expected to hold all the time. Therefore CBV provides an implicit "always" on all top-level assertions. The most common way to combine assertions is by "and", so the "begin-end" operator in CBV is interpreted as a conjunctive fork. CBV encourages a forward-looking style of coding, in which nesting of many variations of temporal "if" operators can be used to define the precondition of an assertion. The "if" operators also allow limited expressions of disjunction. Negation and liveness operators in CBV are limited to Boolean expressions. The resulting language is richly expressive of safety properties and offers limited liveness. It covers the very important intersection of what is most commonly of interest to designers with what is efficient to implement, e.g. in thread-based simulation monitors or on-the-fly model checkers.

The thread-based semantics of CBV provides a unified conceptual framework for writing temporal assertions without having to rely on traditional temporal logic operators or regular expressions. The underlying multi-threading features give the user a convenient foundation: new threads begin every cycle due to the implicit top-level "always", and forking is as easy as opening a "begin-end". Since there is no join in CBV, the scoping for threads is particularly simple--each thread has its own local namespace, inherited initially from its parent, but free to evolve independently of its siblings. Put another way, the various concurrent CBV

threads grow as a *tree* [theoretically, this tree is nothing more than a computation tree of a forall-automaton], and as a result, CBV threads can carry local variables very intuitively.

CBV also has functions and tasks for modularity. The tasks can be temporally recursive, which makes the coding of ongoing assertions based on patterns of events routine.

These days, assertion languages based on regular expressions have become quite popular (Synopsys' OVA, Intel's ForSpec, IBM's Sugar, and the emerging PSL and SVA from Accellera). Regular expressions provide a convenient way to define temporal patterns, or "sequences", which can then be combined in various ways to define assertions. Admittedly, CBV does not offer the same convenience as regular expressions, but CBV does go a long way in letting the user define temporal patterns. The idea is to use the same intuitive thread-based semantics from assertion writing when defining the patterns. The user puts the pattern matching code into "matching tasks" and uses the leaf-level "return" statement in such a task to indicate that the pattern has been matched. The matching task can then be called in the condition of an "if" statement, and every thread that matches the pattern will return to execute the body of the "if". This makes the "if" work like a "sequence implication" in Sugar or the "triggers" operator in ForSpec. The neat thing in CBV is that the thread semantics that the user thinks about is the same for defining patterns and properties.

By the way, there are some challenges to adding local variables to regular expressions. Regular expressions provide "and" and "or" operators that cause somewhat different kinds of joining. At the end of an "and", there is typically a true join of the threads of the branches, so there must be defined a method for resolving inconsistent local variable assignments in the different threads. At the end of an "or", there may not be a true join, since it might be that only one branch matches. In case both branches match at the same time, one must decide whether to join the local variables as in the "and" case or let the threads continue separately. Whatever decisions are made to deal with these joins, it seems preferable to have static (compile time) rules about how local variables pass through joins. The unbounded repetition operator poses another challenge for supporting local variables in regular expressions.

A regular expression

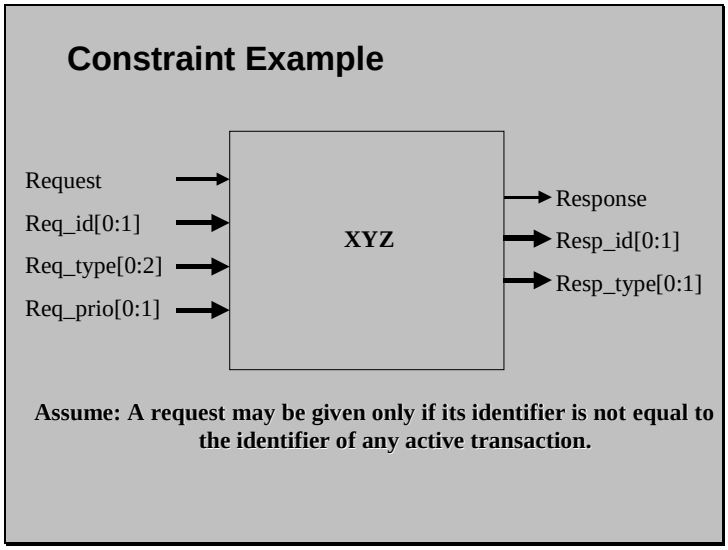
```
(!p*[0:inf]; !p(local x[31:0] = e[31:0]) ; !p*[0:inf]; p && (y == x));
```

samples x at a non-deterministic point prior to the first assertion of p. Thus, this regular expression matches provided that at the first assertion of p (after the start of the match), y has a value that appeared in e at some prior point (the prior point also being after the start of the match). Checking for matches of such a regular expression in simulation can be very expensive.

We do not wish to get involved in a language controversy but we would like to make the point that CBV (a) is easy to learn, (b) is used on real designs by both verification engineers and designers, and (c) fit into our total verification flow. Which brings us back to the main point of this talk. It is synergy among languages, tools and methodology that has maximum impact on design.

A Simple Constraint Example:

Assume the following bus interface unit. Attached are two simple constraints on the interface to the unit: the environment is not allowed to generate a request for a request id that is already active.




```
module xyz;

  /* Definitions Block */

  function activate(id[0:1])[0:0]   = request  & (req_id == id) ;
  function deactivate(id[0:1])[0:0] = response & (resp_id == id) ;
  function active_next(id[0:1])[0:0] =
  (
    deactivate(id) ? 1'b0    :
    activate(id)   ? 1'b1    :
                     active[id]
  ) ;

  var active[0:1] =
  {
    active_next(0),
    active_next(1),
    active_next(2),
    active_next(3),
  } ;
```

Constraint: A request may be given only if its identifier is not equal to the identifier of any active transaction

```
    constraint(request ? ~active[req_id] : 1'b1) ;

endmodule
```

SimGen Constraint Generation

As noted above, simulation generation from constraints a la SimGen [2] works very simply. At compile time the user supplies a set of constraints to the SimGen compiler. The constraints are compiled either into a

Verilog module [kukula00] suitable for emulation or into a C program [pixley99] that runs during simulation. During simulation a user can give a set of biases to the runtime program to control the likelihood that an input bit will get set to 1. The user can also set an initial state using Verilog directives or can give the design a synchronizing sequence. At any point the user can call a SimGen task, which turns the simulation over to the SimGen executable. After that point SimGen generates simulations compliant with the constraints and biases every clock cycle.

There is a logical (and very real) possibility that the simulation will encounter a deadend state, i.e., a state for which there is no solution for the inputs. At that point the simulation will stop and the offending trace will be generated. The cause of a deadend state can either be an error in constraints (for which we had a simple debugger) or an error in the design that allowed it to get into an unanticipated "bad" state. In either case, the deadend must be analyzed.

Summary

The thesis of this paper is that synergies of verification technologies are most effective. As an example, constraint-based verification was examined. Constraints can be used (1) to generate simulations, (2) to monitor simulations, (3) as formal environment definitions for model checking, (4) properties for model checking. Therefore, constraints can be used in an assume/guarantee methodology with both simulation and formal verification. A few key advantages of this approach are that constraints can be developed incrementally starting with inexpensive early animation of designs. They can be introduced into an existing verification methodology easily. Constraints will mature while the design matures. Most importantly, constraints can easily be "flipped" to become monitors as well as generators. Hence, they support reuse.

Bibliography

[pixley99] C. Pixley, K. Shultz, J. Yuan, "Integrated Formal and Informal Design Verification of Commercial Integrated Circuits", Proc. of the international Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), pp. 1061-1067, June 28, 1999.

[yuan99] J. Yuan, K. Shultz, C. Pixley, H. Miller, A. Aziz, "Modeling Design Constraints and Biasing in Simulation Using BDDs", ICCAD'99, pp. 584-589.

[kaufmann98] M. Kaufmann, A. Martin, C. Pixley; "Design Constraints in Symbolic Model Checking", CAV'98, pp. 477-487.

[park00] J. Park, C. Pixley, M. Burns, H. Cho, "An Efficient Logic Equivalence Checker for Industrial Circuits", JETTA vol. 16:1/2, pp. 91-106, 2000.

[jolly02] S. Jolly, A. Parashkevov, T. McDougall, "Automated Equivalence Checking of Switch Level Circuits", DAC'02, pp. 199-205.

[kukula00] J. Kukula, T. Shiple, "Building Circuits from Relations" CAV 2000.
We thank to Jim Kukula and Robert Damiano who commented on this paper.

Formal Interface Compliance Verification

C. Norris Ip

ip@tempusf.com, Tempus Fugit, Inc., Fremont, CA

Abstract

This paper discusses the role of formal interface compliance verification in a design flow, and how it can significantly improve the design and verification process of complex SoC.

1 Introduction

Most complex SoCs and reusable IPs being designed today work around standard interfaces. And as design complexity increases, proprietary interfaces are also being developed more vigorously within large companies to partition a complex design and for reuse across multiple product generations.

The verification problem for designs using these interfaces is critical [1]. Frequently, during the design stage, it is not known which specific systems the SoC or the reusable IP will be interacting with. Furthermore, it is possible that different designers reading the same English specification may interpret various scenarios differently.

Since 1999, the team at Tempus Fugit has been formally verifying complex interfaces, for a wide range of designs. To prepare for these verification efforts, we first captured the interface requirements in an executable verification module. We will use PCI [2], AGP [4], and PCI-Express as examples in this paper. Each of these interfaces contain more than a hundred requirements, some come from the actual compliance checklist from PCI-SIG [3], and some come from the English specification documents.

Once the requirements were captured, we reused them for a wide range of designs. Using TempusQuest, an advanced formal verification tool developed at Tempus Fugit, we analyzed each complex design for compliance with these requirements, exhaustively catching all corner cases that violate these requirements. These efforts can fit into any existing methodology and lessen the effort in writing simulation test cases to hit these corner cases.

Finally, to obtain the maximum benefit of formal technologies, we also apply our tool to the interfaces at the individual blocks of a design, utilizing *an assume-guarantee* approach to verify the interaction among multiple components. We also applied formal techniques on some of the end-to-end requirements for the components. In doing so, we can replace a significant portion of ad-hoc unit-level simulation tests, resulting in shortened integration time, and perform formal regression to make sure last minute changes have not broken the design.

PCI, AGP, PCI-Express and other standard or proprietary interfaces contain complex requirements. While there are several commercial assertion-based verification products, their focus is typically simple local assertions instead of complex interface requirements. If a requirement is too complex for the automated analysis to formally verify, the requirement is validated in simulation instead (as stimulus generator and event checker). However, we have been able to consistent verify them formally with our verification engine developed at Tempus Fugit. In fact, the tool is powerful enough to verify not only interface compliance, but also end-to-end functional requirements.

In section 2, we summarize various aspects of functional verification of these complex designs, and then in section 3 we move on to discuss the potential productivity gain through the consistent use of formal techniques.

2 Functional Verification of RTL Designs

The various aspects of functional verification can be summarized as follows:

Local Assertions

A local assertion is a statement to declare the implementation decision with respect to the code in the surrounding vicinity. Examples include one-hot encoding, queue-overflows, etc.

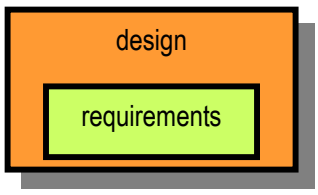


Figure 1. Local Assertions

Interface Requirements

An interface requirement is a statement to declare how two entities communicate with each other. Each

entity can be a complete ASIC design, an SoC design, a reusable IP, or a block within a design.

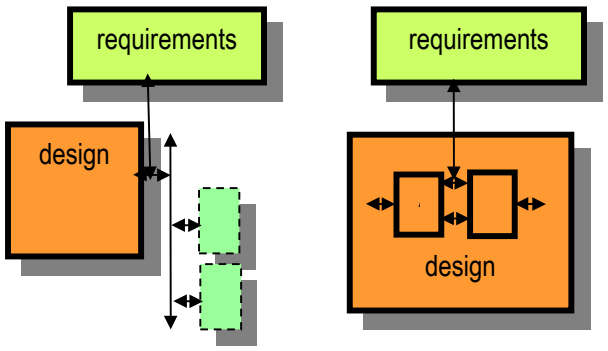


Figure 2. Interface Requirements

Let's take the PCI and AGP interfaces as examples. PCI has more than a hundred requirements, and one requirement is that the master must always assert byte enables on the upper bits of CBE when the master asserts REQ64# and the slave has not responded. AGP has a similar number of requirements, and one of the requirements is that, if any AGP read or write grants are pending, the core logic must not start PCI or fast-write transaction.

End-to-End Functional Requirements

An end-to-end functional requirement is a statement to declare what an entity should generate with the supplied data. The entity can be a complete ASIC design, an SoC design, a reusable IP, or a block within a design.

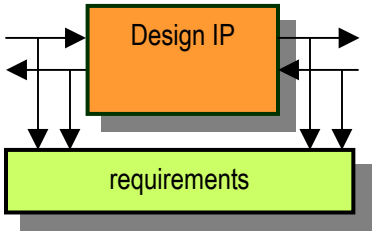


Figure 3. End-to-End Functional Requirements

Let's take the link layer of the PCI-Express interface as an example. For each packet going into the design, we need to verify proper sequence number assignment to the corresponding outgoing packets, their data integrity, proper ordering, and that the design does not drop or duplicate packets.

System-level Requirements

A system-level requirement is a statement to declare the overall behavior of the design across time. Sometime it involves less tangible items such as typical throughput, etc.

The complexity of these requirements increases as they move from local assertions to system requirements. Because local assertions cover the code in the surrounding vicinity, these local assertions tend to catch local problems.

Because interface requirements describe how two entities communicate to each other, they typically involve the control logic of the design. These requirements become the contract between the two entities. The verification of these requirements on each entity is challenging because of potential ambiguity in the specification and because of the complex corner cases due to the possibly arbitrary behavior of the other entity.

While the implementation within a module may change from time to time, a well-designed interface is less likely to change. This is especially true for standard interfaces and reusable cores. Furthermore, it is usually difficult to manually create simulation test benches to hit all the corner cases with respect to the interface requirements, so it naturally calls for a formal solution.

On the other hand, end-to-end functional requirements are related to the data-path and the overall functionality of the design. Formal verification of these requirements is often more difficult, but it has the potential of significantly contribute to the ASIC and IP sign-off process.

Finally, less tangible system requirements are probably best done by system-level simulation, which is outside the topic of this paper.

3 Using Formal Interface Compliance Verification

The ability to formally verify interface requirements (and end-to-end functional requirements) can have a big impact in the design process. Without it, a design flow may look like the diagram in Figure 4.

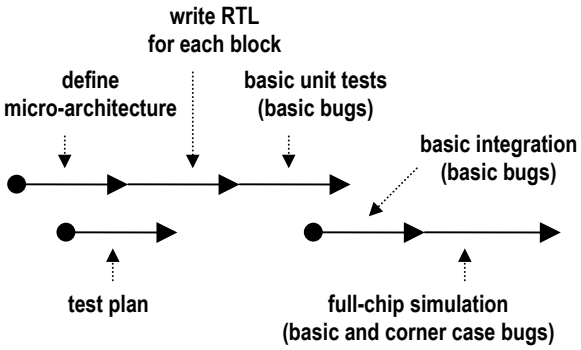


Figure 4. Basic Design Process

In this process, the designer would write ad-hoc unit-level simulation tests to verify some functionality of the blocks, but the main effort in finding corner case bugs would have to wait until full-chip simulation is up and running. Because of the loose interaction among the development of individual blocks, a significant amount of time is also spent in the integration of the units before full-chip simulation is ready.

With a formal tool capable of verifying complex interface requirements, significant improvement in the development process can be achieved with minimal changes to the existing methodologies. For example, it may be shortened into something like the diagram in Figure 5.

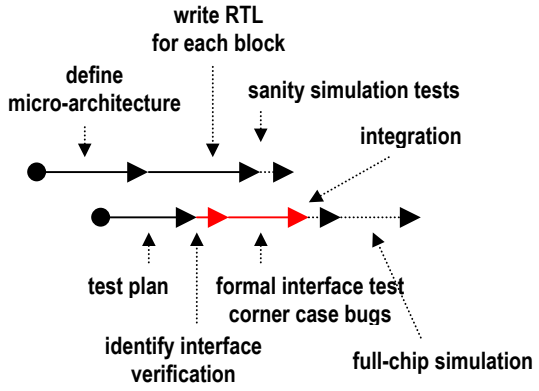


Figure 5. Formal Interface Verification

The steps relevant to the use of formal interface compliance verification in this diagram are:

1) Work out the overall architecture of the design.
2) While defining individual components in the design, create or buy appropriate *interface verification module* for formal analysis.
3) While developing the RTL code for individual components, debug the code interactively and formally with respect to the requirements in the verification modules.
4) During the process of developing the RTL code, exchange assumptions among the designers and verify them formally.
5) When full-chip simulation is up and running, maintain the day-to-day correctness by running formal regressions.

In this process, the interface requirements are used as a contract among the blocks in the design. If each block has been verified using the same interface verification module before integrating into full-chip simulation, the integration step would be much shorter.

Furthermore, the full-chip simulation step would also be shorter, because corner case bugs with respect to the interfaces and blocks are detected early during the unit-level verification. While the designers are busy writing the RTL, the verification engineers can write or obtain appropriate interface verification modules. We have also designed our tool so that the verification of the unit may start even before the RTL of the block is complete, because our tool will regard the missing pieces as black boxes.

Finally because formal verification achieves 100% coverage for the requirements, the proofs performed during the early steps can be executed again as an effective formal regression during the later steps, while enhancements and non-interface bug fixes are made to the design description.

The remainder of this section will go over each step in more details.

3.1 Interface Verification Modules

Very often, it is possible that different designers reading the same English specification may interpret various scenarios differently. It is important that the designers or the verification engineers consider all possible interpretation of the other side of the interface.

Writing the interface requirements in an executable format would reduce the possibility of misunderstanding. We tested our verification modules for various standard interfaces in a wide range of designs, and therefore, have resolved most of the misunderstanding. For proprietary interfaces, the architects or the verification engineers responsible for the design may write the verification modules. Because these verification modules are written independently from the design efforts, discrepancy in the interpretation of specification by the architects and the designers can be resolved through the formal verification process.

The actual languages in which these requirements are specified are not critical to the design flow, as long as they have clear semantics associated with them. In many cases, a hardware description language such as Verilog and VHDL is sufficient. Open Verification Library (OVL) can be used a convenient layer on top of Verilog for a more concise assertion specification.

There are other advanced languages being standardized in the industry. A declarative property specification language is being standardized by the Accelera Formal Verification Technical Committee, and a procedural assertion construct proposal is being evaluated by the Accelera SystemVerilog Committee. Both of them contain constructs that are more expressive than Verilog and VHDL, useful for liveness properties that describe infinite behavior.

One of the difficulties in getting the designer to use assertion-based verification is to persuade them to write assertions. By supporting HDL languages, they can start capturing requirements without learning a new language.

By concentrating on interfaces, the verification module can be created by anyone, not necessarily the designer. Using it as a contract between two components can avoid misunderstanding, and free up precious simulation time for validating the transaction-level activities of the design.

3.2 Formal Block-level Verification Before Simulation

Because the formal analysis does not depend on the creation of a test bench, designers can start verifying their blocks without waiting for the simulation environment to be up and running. Furthermore, formal verification may start even when the RTL code has not been completely developed. Any missing code in the RTL can be regarded as a black box that generates arbitrary outputs. If the existing RTL can be formally verified with these arbitrary behaviors, the RTL is guaranteed to behave correctly when the missing pieces are added.

3.3 Simplifying the Integration Process

Traditional unit-level simulation tests are usually done in an ad-hoc way with restricted stimulus. Therefore, they usually test the basic behavior of the block only. Furthermore, as shown in Figure 6, the stimulus generator and the event checker for individual blocks are typically developed independent of each other, and it is difficult to determine whether the generator for block A generates all legal outputs from block B and vise versa.

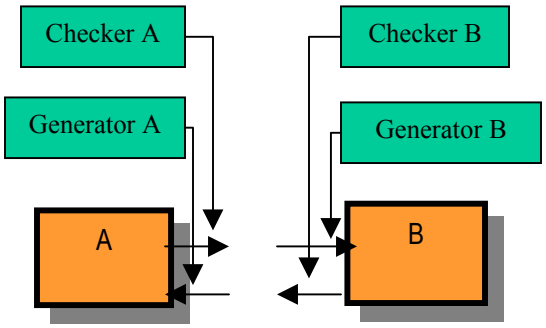


Figure 6. Unit-level Simulation Tests

When formal verification is used, the same interface verification module can be used for the verification for both block A and block B, as shown in Figure 7.

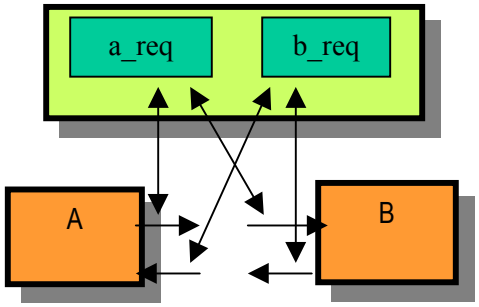


Figure 7. Formal Unit Tests

During the verification of block A, the proof can be set up as follows:

```
% assume b_req
% prove a_req
```

and during the verification of block B, the proof can be can be flipped as follows:

```
% assume a_req
% prove b_req
```

This style of proof structure is known as the *assume-guarantee approach*. Because formal verification is exhaustive, the verification of block A may also detect extra assumptions about block B in order for block A to function correctly. These assumptions can be appended to the verification module so that they can be verified in block B. A semi-formal or simulation methodology would be less effective

The fact that these requirements can act as a contract has serious implications. As shown in Figure 8, it can be used to verify an IP core while capturing its assumption about the environment. The same requirement can be packaged with the IP so that the IP user to verify those assumption (formally or informally) to make sure the SoC is using the IP correctly.

3.4 Trust-worthy Regressions

The verification modules and the set up scripts for the proofs at the unit level can also be reused as a formal regression through out the later steps in the design process. Because the addition of a new feature or the fix for other functional requirements may introduce new corner cases, it may break existing features. Because of the new corner cases, existing simulation tests may not be able to detect this problem. New tests have to be written for both of the old and the new features. With formal verification, the formal proofs can automatically

adapt to the new design description and verify the new corner cases as well, so the designers only need to verify the new features.

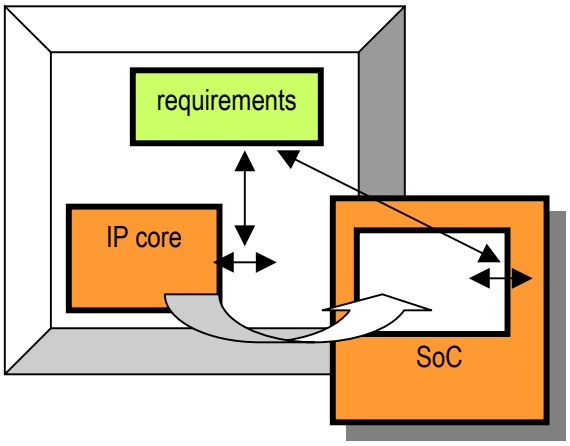


Figure 8. IP and Its Operating Environment

4 Conclusion

In this paper, we identified the various aspects of functional verification and described how formal verification can significantly impact the design process. Our experience indicates that applying formal tools for the interface compliance verification (and end-to-end functional requirement) can lead to a significant productivity gain.

Reference

[1] Vigyan Singhal and Joseph Higgins, Compliance Verification for SoC and IP interfaces. DesignCon 2002.

[2] PCI Special Interest Group. PCI Local Bus Specification Rev 2.2. December, 1998.

[3] PCI Special Interest Group. PCI 2.2 Compliance Checklist. www.pcisig.com.

[4] Intel Corporation. Accelerated Graphic Port Interface Specification, Revision 2.0.

Applying Formal Methods to Protocol Specifications and System Architecture

Ching-Tsun Chou

Multi-Processor Architecture
Enterprise Platforms Group
Intel Corporation

Disclaimer

The views expressed in this talk are the presenter's alone and not necessarily those of Intel Corporation

Why formal methods?

- Architectural specifications contain complex distributed protocols whose correctness is nontrivial to establish
  - Examples: Directory-based cache coherence protocols, forward-progress mechanisms, variations of sliding window protocols, ...
  - Goal: Get protocol specifications correct before implementations commence
    - The earlier a bug is found, the easier it is to fix it, and the more flexibility there is in possible fixes
  - Formal verification (FV) is a body of powerful techniques for achieving this goal
- Formal modeling promotes clear thinking and minimizes misunderstanding and misinterpretation of specifications
  - In the early stages of protocol design, more bugs are found during formal modeling than by model checking
    - Protocol design and formal modeling should go hand in hand
  - Formal modeling produces unambiguous "golden models" of at least some aspects of complex protocols
    - Executable reference models can be generated from formal models
- Experience shows that formal methods work
  - Already a standard industry practice: Intel, Sun, IBM, Compaq, SGI, ...
  - As this talk hopes to demonstrate

Formal vs simulation-based verification

- Simulation-based verification:

  Check a small fraction of all possible behaviors of a large model

  + Very large and relatively complete model

  + Model need not be simplified or abstracted

  – Only a very small part of the state space is explored

  – Need to generate tests and collect coverage feedback

  > Results only as good as your *tests* and *checkers*

- Formal verification:

  Check all possible behaviors of a small model

  + All states are exhaustively explored

  + No tests are needed and coverage is 100%

  – Only very small models can be handled

  – Often need drastic simplification and abstraction

  > Results only as good as your *models* and *properties*

Moral: There is no free lunch!

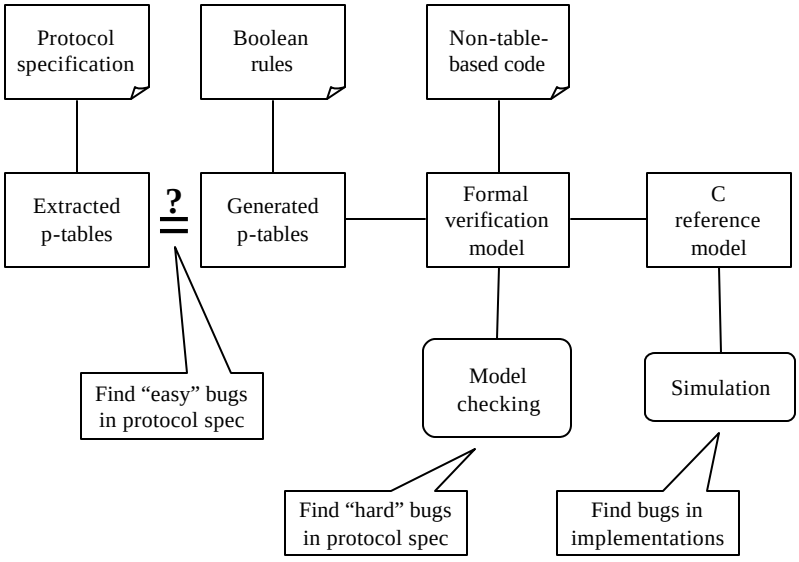
Overview of Intel's Scalability Port (SP) architecture

- Designed for mid-range shared memory multiprocessors
- Employ high-speed point-to-point interconnect that provides good scalability for mid-range to high-end systems
  - Shared buses are neither cost-effective nor scalable beyond limited number of processors due to signaling, thermal, mechanical, and other challenges
- Support flexible system architecture
  - Enable cost-optimal small systems to scalable high-end systems
  - Enable system vendors with proprietary system interconnects and components to use Intel building blocks
- An instance of SP was implemented in Intel's E8870 chipset

Overview of SP cache coherence protocol

- Make **no** assumption whatsoever about the relative timing of events or the ordering of messages
  - Completely asynchronous, event-driven specification
- Directory-based, though the directory:
  - is optional (no directory = null directory)
  - may or may not be physically distributed
- A generalization of the invalidation-based MESI protocol, but different caching agents may do MESI-state transitions at different times
- Employ mechanisms to resolve *conflicts* — collisions of requests from different requesters to the same cache line — in a *distributed* manner
- Table-based specification with ~1,000 rows in all tables
- >20 transaction types, each of which has a different behavior by itself and can interact with every other transaction type

SP cache coherence protocol validation flow



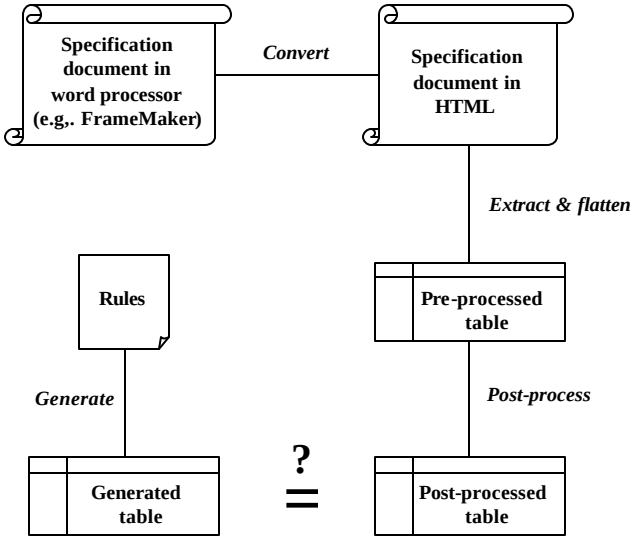
Properties verified

- Data consistency:
  - If a cache's state is valid (i.e., S, E, or M), then its data is up to date
- Cache and directory state consistency:
  - If any cache is in state E or M, the other caches must be in I
  - If a presence bit in directory is 0, the corresponding cache must be in I
  - If the directory state is I, all presence bits are 0 (and hence all caches are I)
  - If the directory state is S, the caches whose presence bits are 1 are in I or S
  - If the directory state is E, there is exactly one presence bit being 1
- Weak liveness properties: AG EF (cs = CS), for each "control state" cs and each possible value CS of cs
  - Excellent guard against missing rows in protocol tables and other unexpected cases
  - Detect both global and local deadlocks, but not livelock or starvation
  - Do not rely on fairness assumptions

Results of SP cache coherence protocol FV

- An SP cache coherence protocol has >10^33 reachable states for a configuration containing 1 cache-line address, 1 home node, 2 caching nodes, and all >20 transaction types
- Each property takes (on the average) 4~5 hours to model-check on a 700 MHz Pentium III Xeon machine with 4 GB of physical memory
- Many interesting bugs were found in successive versions of SP cache coherence protocol by both formal modeling and model checking
  - In fact, more bugs were found by the former than by the latter in the early phase of SP protocol design
  - Not surprisingly, most problems were found when SP was first designed and during major revisions (e.g., when new transaction types were added)
  - But even minor revisions could introduce problems
  - Moral: As far as cache coherence protocols are concerned, unaided human reasoning should not be trusted

Rule-based table checking flow



Why rule-based table checking works

- Tables and rules take two fundamentally different but complementary views:
  - Tables are *row-centric* and enumeration of cases (row = case)
  - Rules are *column-centric* and expression of relationships between columns
- By comparing the two views against each other, the chance of a bug escaping is minimized
  - Ideally, tables and rules should be constructed by two different persons
- Expression of complex relationships between visible columns is simplified by means of *hidden columns*
  - "Cause-and-effect" metaphor: Hidden columns are the "ultimate" but invisible "causes" of visible columns
  - Hidden columns are hidden by existentially quantifying them away
- Hidden columns are used to increase further the difference of the two views

Results of rule-based table checking

- Coded boolean rules for SP protocol tables and checked them against each other
  - Typically dozens of errors were found before tables and rules agree
  - Most errors were trivial (e.g., typos), but some were more serious (e.g., missing cases or systematic misunderstanding)
- Maintained the agreement between tables and rules over 2 years and tens of major and minor protocol revisions
  - Changing rules to keep up with tables almost never required more efforts than changing tables themselves
- Rule-based table checking is our first line of defense, flushes out virtually all "easy" bugs, and has very low computational overhead
  - It takes < 5 minutes to extract and verify by rules all SP protocol tables
- We are ***not*** advocating that "code review" of tables be eliminated
  - "Code review" is still a must at the beginning
  - We *do* advocate that insights from "code review" be captured and codified by rules and re-used later when tables are changed

Novel applications of binary decision diagrams

1. Rule-based table generation and checking
   - Boils down to enumerating satisfying "truth" assignments of boolean expressions over enumerated types
2. Search for minimal deadlock-free wormhole routing scheme
   - A wormhole routing scheme is deadlock-free ⇔ Its channel dependency graph is acyclic ⇔ The transitive closure of the graph contains no self-loop
   - Hence reducible to BDD fixpoint computation
   - Details in our FMSD paper
3. Search for fault-tolerant link initialization sequences
   - Details in our FMSD paper

- Observations:
  - "Formal methods thinking" leads to new ways of looking at old problems
  - A little BDD goes a long way:
    - BDD is an efficient representation of boolean rules (1. above)
    - BDD supports exhaustive search of a "solution space" (2. & 3. above)

Lessons learned

- Formal modeling steered us toward more precise and concrete protocol specifications than we would have written without it
  - Even an abstract formal model requires one to spell out what exactly one means by each protocol structure and action
  - Formal modeling also turned out to be an excellent way to help architects articulate their ideas
- Formal verification gave us much higher confidence in the correctness of our protocol specifications than we would have without it
  - Certain distributed protocols (e.g., directory-based cache coherence protocols) are too complex for unaided human reasoning alone to get correct
  - Formal verification makes it less risky to modify protocol specifications
- Architecture definition affords a rich and fruitful area for the application of formal methods
  - Avoid state explosion with a high level of abstraction
  - Get to bugs at the earliest possible stage
  - Encourage architects to choose more "validation-friendly" schemes
- Applying formal methods to a specification enables the exploration of design spaces that are beyond the scope of any particular implementation
  - Especially important for Intel, which defines architectures that will be implemented by multiple vendors over multiple product generations

Acknowledgements

- Mani Azimi, Jay Jayasimha, Akhilesh Kumar, Victor W. Lee, Phanindra K. Mannava , Seungjoon Park, and Aniruddha Vaidya all contributed to the work described above.

Non-presented papers:

Topics:
System-level Design
Design for Manufacturability

PROGRESS: PROcess GRaph For Embedded Systems and Software: Combining Top-down and Bottom-up System Design Methodology

James R. Armstrong Shekhar S. Agrawal Hiren D. Patel Sandeep K. Shukla

Bradley Department of Electrical and Computer Engineering
Virginia Polytechnic and State University
Blacksburg, VA 24061
Email: {jra, ssa, hipatel2, shukla}@vt.edu

Abstract

In this short position paper we briefly describe a methodology under development at Virginia Tech, for combining specification oriented design methodologies and IP-reuse based design methodologies for designing embedded systems. Specification oriented paradigms are ***top-down****, and refinement based, and most of them have sound mathematical modeling, abstract representation, and supporting formal methodologies. Ptolemy or SpecC can be cited as examples of such methodologies and frameworks. However, in the industry,* ***cut-and-paste reuse****, and IP* ***component based reuse*** *are used quite often. These reuse methodologies are often* ***ad-hoc*** *in the absence of tools and techniques to create abstract models from the reusable components, and often due to the fact, that in practice part of the design is done in a specification oriented manner, while some parts are taken from existing code base within a company or from purchased IP-cores. We propose using an abstract intermediate representation of component functionalities, namely, Process Model Graphs (PmG). Various manipulative operations on such representations can be viewed us ways to trim, merge, and join IP blocks for systematic reuse. This methodology will allow us to manipulate reusable cores in the abstract, and make top-down specifications merge with PmG abstractions of cores, and then either map them to existing reusable components, or generate required components with tool support. This would lead to a design methodology which successfully combines two different methodologies into one and enables a framework that can benefit from both the approaches.*

I. Introduction:

In the changing world of networked and ubiquitous computing, increasingly sophisticated handheld and bio-medical devices, embedded computing is becoming pervasive and increasingly complex. Furthermore, Moore's Law is imposing an even greater challenge by the continuous improvement in silicon technology at an exponential rate while the design productivity of engineers and computer aided design tools are not increasing at the same pace. This 'Productivity Gap' in semiconductor design industry has necessitated research and development in design and validation methodologies that can enhance productivity, and render validation reliable and inexpensive. A number of approaches have been proposed in this context with the recent developments in system level design languages and models of computation which influence the choices of reuse of intellectual properties and provide efficient simulation and validation environments. However, almost all proposed solutions to the problem of system engineering have addressed one of the two design flows. They have either been *Top-Down specification oriented* using languages such as Ptolemy or SpecC or a *Bottom-up IP reuse approach* using languages such as VHDL and SystemC.

In the Top-Down design flow, the process begins with a specification from which an abstract mathematical model of a system or a portion of a system is constructed. This model is validated to ensure that the specification is interpreted accurately and different algorithms for implementing the system behavior are explored. Several stages of refinement follow this step adding more details to the abstract model. On the other hand, the bottom-up approach exploits design reuse to achieve productivity necessary to build complex systems. Historically, Chip Design companies have always reused designs in going from one product generation to another, but the efficiency of bottom-up design is enhanced by the reuse of IP-cores that a company can buy from an outside source. The challenges with using these IP-cores is designing

interfaces between the cores and other logic as well as being confident about their correctness.

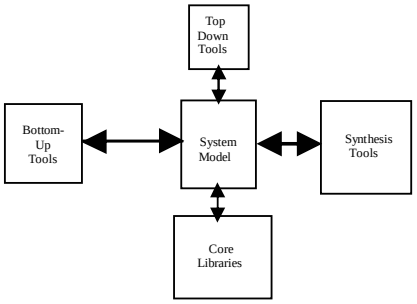


Figure 1. Integrated Top-Down/Bottom-Up Design Approaches

II. Problem Statement

It is a commonly held belief among system designers and CAD practitioners that one-of-a-kind domain-specific construction methodologies for such systems will not be sufficient. Radical advances in systems engineering are needed to allow composition, reasoning and validation of these novel systems and applications. In fact, such combining is practiced in the industry in an unorganized manner in almost every product design cycle. However, lack of formalization of this combined methodology leads to lack of tools and CAD environments, which could lead to better productivity in the design teams.

Figure 1 shows a schematic of our view of industrial design practice, which we believe is an ad-hoc combination of both top-down and bottom-up design styles. The center block shows the system model that integrates the designs from the top-down and bottom-up design flow processes and allows validation of its correctness. For example, when designing a new microprocessor, a top-down model is first built for quick evaluation of performance and feasibility but due to the existence of code and IP-cores in house and off the shelf, eventual implementation is often largely based on existing IP blocks. Unfortunately, in most design cases, this reuse is often cut-and-paste based rather than being based on a solid methodology. Therefore, in our opinion, neither a top-down nor a bottom-up methodology by itself is an adequate design approach, but a combination of the two can lead to new system engineering tools and techniques that will enable System Designers and Engineers to design systems from IP-cores in a specification oriented manner. This approach will allow the development of semi or fully automated refinement methodologies targeting IP-cores or parts of IP-cores to be automatically extracted from IP blocks. However, is it possible to develop a sound and robust methodology and design environment that can combine a top-down specification and models extracted from cores into a system model from which one can map to existing cores, synthesize common protocols between cores and synthesize parts of glue logic?

A comprehensive, robust methodology and environment to combine the different design approaches has not yet emerged due to the lack of a single uniform semantic foundation for the specification of either the high level models or the cores represented at low level. To the best of our knowledge, no work has been done in combining the specification oriented approach with the core reuse based approach because of this lack of uniform semantic foundation. In this paper, we introduce a robust system engineering environment for embedded hardware/software system that brings together the best of both worlds. This environment is currently under development.

III. Related Work:

The approach presented here is based on advances in specification oriented top-down and bottom-up IP-core reuse based hardware and software design methodologies. Academic projects and some CAD/EDA vendors have supplied some tools and techniques, such as Ptolemy [10] and Metropolis projects [9] from UC Berkeley, VCC from Cadence [4], Co-Centric System Studio from Synopsys [16], Rational Rose [11] and other visual tools for software development. These tools and methodologies guide development of complex hardware/software systems through visual programmatic means, which can be viewed as top-down development from high-level specification enabled by a manual or semi-automatic refinement methodology. On the other hand, extensive bottom-up IP reuse [17] either in the form of *cut-and-paste* of existing HDL code or software code, or to some extent by propriety standards for creating IP blocks is occurring in the industry. Evidently, we see a need to combine these two design approaches and

construct a comprehensive and robust methodology and environment.

Attempts to construct a robust methodology began with Kahn in 1974 by proposing the use of processes as fundamental to modeling. In 1974, Kahn presented the idea of Process Networks, i.e., sequential processes with blocking read

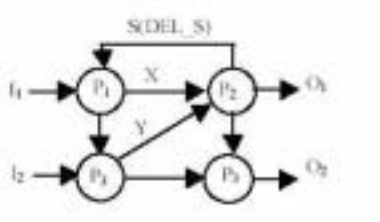


Figure 2. Process Model Graph

interconnected by infinite capacity FIFOs [6]. Process Behavior was implemented using Kahn's Language constructs. VHDL and Verilog employ process mechanisms [5, 12]. SystemC has the notion of processes defined as methods, threads and clocked threads. In [1, 2] we advocated the use of PmGs to represent VHDL behavioral models. Formal properties of processes were developed in [7]. Recently, we performed hardware/software co-design of a GSM system using the PmG representation. [3, 13, 20]

IV. Our Approach

The basis of our semantic foundations for core representation and top-down specification refinement is proposed in the form of ***Process Model Graphs (PmGs),*** and operational semantics of each operation at the tool level are given in the form of operations on PmGs. We provide details of the denotational and operational semantics for PmGs in [22] and limit our paper to a brief overview of PmGs. Firstly, a Process Model Graph (PmG) is a digraph [21] whose nodes represent processes and whose arcs represent signals as shown in Figure 2. Signal arcs are labeled with the signal names they represent, e.g., X and Y. Sometimes the name of the generic delay for a signal is added to the signal name, e.g., S(DEL_S). The PmG represents a module. The input arcs I1 and I2 represent module inputs, and the output arcs O1 and O2 represent module outputs. Other arcs represent internal feed-forward (X and Y) or feedback (S) signal flows.

The operations that can be performed on a PmG are ***join, trim*** and ***merge***. Joining and trimming are most frequently employed when IP-cores are ported into the System Model PmG. Cores are either hard or soft cores. For ASICs, hard cores consist of the IC layout and for FPGAs the bit file. Soft cores consist of an RTL description in VHDL or Verilog. Hard cores must be accompanied by behavioral models so the core can be joined with other models for simulation purposes. Soft cores may not have behavioral models because the RTL model can be used for this purpose. IP-cores are more marketable if they offer a range of capabilities. However, when constructing the PmG system model we wish to use specific instances of the core. Thus, before joining the core model to the PmG, the core must be trimmed of redundant behaviors. Below we discuss the nature of the trim and join operators. We present these concepts in the context of an example where core models are to be inserted into a system model implemented via SystemC.

Figure 3 shows a core based model which interfaces a FIR filter to a two channel CODEC. There are two cores in the model: 1) the FIR filter is a hardcore which is produced by Xilinx's Coregen Program [19] 2) the Clock Generator and the Channel are soft VHDL cores provided by the board manufacturer [18]. The CODEC is fixed hardware and the core based model is a VHDL model to be implemented in FPGAs.

Trim Operation: The trimming operation generally results to three forms of elimination of redundancy in the existing cores. For example, one might have an ALU core, that does both integer, and floating point computation. If the required use, does not require floating point operation, one can use program slicing techniques to eliminate the parts relevant to floating point operations. In the example of Figure 3, we have used three kinds of trimmings as follows:

Functional Elimination: three processes are totally eliminated because high level knowledge of the model shows that these processes are not required.

Eliminated By Code Trimming: instantiation of the parameter values trims the code of process such that the process merely passes inputs to outputs without modification and without delay, resulting in the elimination of that entire process.

Internal Code Trimmed: instantiation of the parameter values in trims the code of process by a certain percentage.

For example, The FIR filter is a Serial Distributed Arithmetic FIR Filter. It performs the

required multiply – accumulate operations serially to save chip area. The core can either be used by itself or a number of them cascaded together. It can be specialized in a number of ways depending on the nature of the FIR filter being designed. Accompanying the core is a multi-process, VHDL behavioral model. Applying the Trimming operation for this core, it is found that a total of 10 functions are eliminated and the resulting core had 39% less inline code than the original core.

Join Operation: As shown in Figure 3, an Interface Module is used connect the CODEC

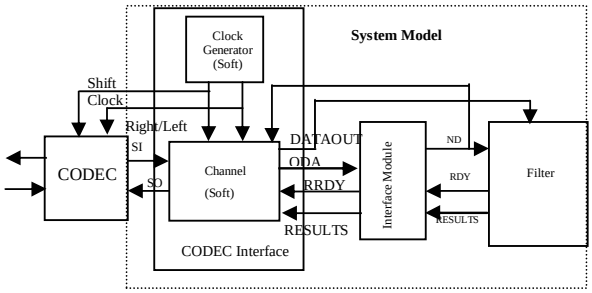


Figure 3. IP-core system model

Interface Core to the FIR core. It performs two functions: a) converts an Output Data Available (ODAV) level to a ND single clock period pulse, and b) registers the filter results output which is stable only for one clock time. Implementation of this interface module is an example of a *join*. In order to automate the joining process, one may have to use *behavioral typing* [8] of core interface signals. In the system illustrated in Figure 3 two data types are used: 1) for control: std_logic, and 2) data: std_logic_vector. As part of an automation process, one can define two behavioral types: *control* and *data*. The join operation consists of type conversion between behavioral types, e.g., the Interface Module in Figure 3 performs two behavioral type conversions: a) type 1_level to type p_pulse and b) type momentary to type registered. Certain assumptions underlie this type hierarchy, i.e., pulses last for one clock cycle as does momentary data.

Merge Operation: Often two cores may be merged in the context of eliminating redundant signals [3].. External signal storage is replaced by internal variables. The read/update mechanism for variables is designed to ensure the same computation.

Mapping Operation: A process whereby two different PmGs in distinct domains can be related to each other by certain mapping definitions provided in [22]. Rigorous definition of mapping is needed to automate the process of mapping a PmG to PmG models of existing modules.

Example

To test our PmG structure and the operations associated with it, we developed a System PmG and an associated SystemC model for the system shown in Figure 4 which is an example for PmG mapping. We utilize the cores described in Figure 3. The CODEC is a two channel, 20 bit A/D, D/A converter, with serial input and output. The CODEC Interface performs a serial-to-parallel (read) and parallel-to-serial (write) conversion. In the system shown in Figure 4, only the input from the CODEC on one channel is used. Thus, the CODEC Interface core is trimmed to a single channel, read only configuration. The CODEC Interface Core is RTL VHDL. The hardcore FIR filter is modeled by behavioral VHDL and is also trimmed to match the specifications. The join operator is in VHDL. The BUS is a high speed serial bus implemented in SystemC, the FFT and Format and Display actors are Ptolemy actors operating in the SDF domain.

We translated the above mapping example into a SystemC model as shown in Figure 5. The experiment validated the usefulness of our PmG operations. It was conducted manually, but automation of this process is under investigation.

Our PmG approach not only provides a robust and comprehensive methodology for design but the integration of hardware/software co-design is a direct byproduct of the proposed methodology. It is very important that modeling, hardware/software co-design, and synthesis be integrated. To put it in a nutshell, the system procedure can be split into 5 different steps as:

1. The Process Model Graph (PmG) is extracted from the SystemC system level model.
2. Each Process P_i is characterized in terms of its software (C_{soft}) and hardware (C_{hard}) complexity. An Instruction Set Simulator (ISS) or profiler for the core processor is used to determine C_{soft} for

each P_i whereas a Behavioral Synthesis Tool determines C_{hard}.

3. Using the process complexity values from the Process Library and deadlines from the Constraint Library, the Graph Partitioning Algorithm partitions the graph into hardware and software sub-graphs
4. In step 3: the Graph Partitioning Algorithm generates a new set of process complexities for those processes assigned to the hardware sub graphs. Also, whenever a signal crosses the boundary between the hardware and software sub-graphs, delay overhead is incurred because of bus delay and processor task switching time. This information is back annotated to the PmG where new delay values are assigned to nodes in the hardware sub-graph and process nodes are added to the graph at the partition boundary to account for bus and processor task switching delays.
5. The Graph Partitioning Algorithm then rechecks the deadlines.

This approach was applied successfully to hardware software co-design of a system which implements the GSM transmitter and receiver sections. The software target was the StarCore 140 Processor; the hardware targets the Synopsys SystemC compiler. The system automatically identified partitions between hardware and software that meet all GSM timing constraints [3].

Case Study

Ptolemy is a top-down design based tool that supports heterogeneous modeling, simulation, and design of concurrent systems [10]. Different Models of Computations (MOCs) that deal with concurrency and time can be modeled using this tool. Each MOC gives an interaction mechanism for components in the model. In PtolemyII, each MOC is represented in a different *domain*. The choice of the type of Director dictates which particular Model of Computation is to be used. The chosen Model of Computation provides execution semantics for actor components of the model and the method of communication between actors. The model of communication is implemented in the Actor receiver, which is specific to the Model of Computation. In SystemC, the execution semantics is implemented by the simulation kernel. The method of communication is chosen by the way in which the Channel between actors is implemented. In Ptolemy, the receiver is buried inside an actor, thus communication is implicit in the Model of Computation. In SystemC the Channel is an explicit component. The SystemC kernel implements a two phase simulation cycle: 1) EVALUATE PHASE: Evaluate the set of currently triggered processes 2) UPDATE PHASE: Update Channels to values within the same evaluations cycle (zero time delay), after delta delay, i.e., updated in the next update cycle and results are available at the next evaluation cycle, or updated at a later time in the simulation during an update cycle. During the evaluation phase, processes notify events and during the

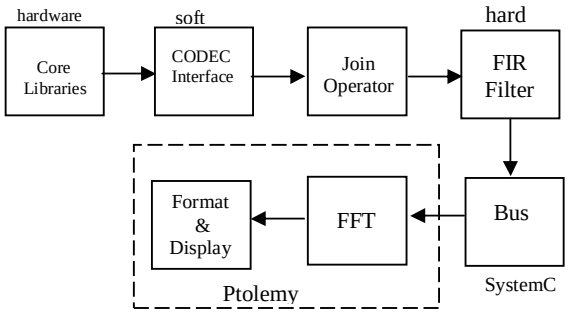


Figure 4. Comprehensive PmG Mapping Example

update cycle those events are triggered. These Ptolemy Actors can be coded in a behavioral fashion and modeled as PmGs. We have conducted work in [22] where we provide an example of an automation of transforming a Ptolemy Actor into a SystemC module. Current work in constructing PmGs from SystemC modules is also underway, allowing for a rendez-vous for Ptolemy Actors to be brought down to the PmG level and IP-cores to be brought up to PmG level given that IP-cores were provided in SystemC modules. Once having constructed the PmG, the above described operations can be performed on the PmG. This trimmed, joined and merged PmG can then be then mapped onto an IP-core for future industrial reuse.

V. Future Work:

Automation of this new methodology will be the key to faster and better reuse of cores, together with merging top-down and bottom-up methodologies. Our focus is to automate the extraction of PmG models from SystemC cores, implement a framework to apply operators, namely, Trim, Join and Merge, followed by

creating a robust design framework that facilitates design reuse. Development of a GUI for easy user interaction and reduced design time is also one of the long term goals of our work. We also plan on adding validation capabilities in the framework.

VI. Conclusion:

In this paper, we have introduced a new development methodology that brings together two different design flows, the top-down and bottom-up flows. The methodology makes use of Process Model Graphs (PmGs) to achieve this goal. The various operations that PmGs support, namely Trim, Join and Merge and the ways in which these operations help in developing the model has been shown. The entire design methodology is illustrated by a comprehensive example that makes use of all the design aspects. In summary, PmG based representation of system models as well as core models and automatic extraction and manipulation of such representation will enable us to create an environment and methodology to combine a core-reuse based methodology with specification driven methodology.

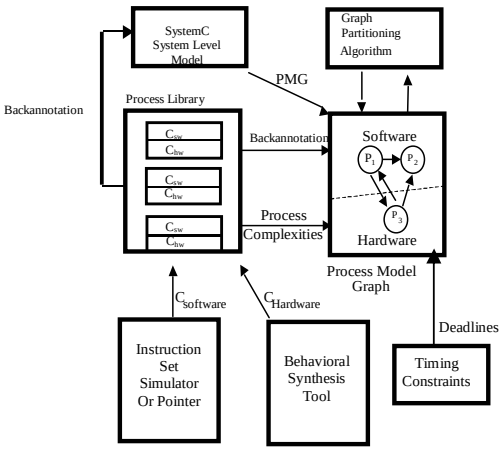


Figure 5. PmG Based Partitioning, and Synthesis System

VII. References:

[1] J. R. Armstrong and F. G. Gray, VHDL Design Representation and Synthesis, Englewood Cliffs, N.J., Prentice Hall, 2000.

[2] J. R. Armstrong, Chip Level Modeling With VHDL, Prentice Hall, June 1988,

[3] J.R. Armstrong, J. M. Baker, and P. Adipathi, "Model and Synthesis Directed Task Assignment for Systems On A Chip," Proceedings of the 15th International Conference on Parallel and Distributed Computing Systems, Louisville, KY, September 2002

[4] Cadence Website on VCC, http://www.cadence.com/products/vcc.html

[5] IEEE Standard 1076-93 Language Reference Manual.

[6] G. Kahn, "The Semantics of a Simple Language For Parallel Programming", Proceedings of the IFIP Congress 74, 1974, North Holland.

[7] A. Lee and A. Sangiovanni-Vincentelli, " A Framework for Comparing Models of Computation," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, Vol. 17. No 12. December 1998, pp 1217-1229.

[8] Edward A. Lee and Yuhong Xiong, "Behavioral Types for Component-Based Design," Memorandum UCB/ERL M02/29, University of California, Berkeley, CA 94720, USA, September 27, 2002.

[9] Metropolis Project Home page, http://www.gigascale.org/metropolis/

[10] "Ptolemy II, Heterogeneous Current Modeling and Design In JAVA," Memorandum UCB?ERL M)1/12, March 15,20001

[11] Rational Rose Webpage, http://www.rational.com/products/rose/index.jsp

[12] Sagdeo, V, The Complete Verilog, Book, Kluwer Academic Publishers, Boston, MA, March 1998,

[13] B. Sirpatil, J. M. Baker, J.R. Armstrong, "Using SystemC to Implement Embeded Software," Proceedings of HDL 2002, San Jose, March 2002, pp 41-45.

[14] J. E. Stoy, Denotational Semantics: The Scott-Strachy Approach to Programming Language Theory, Cambridge, MA: MIT Press, 1977.

[15] SystemC Consortium: www.systemc.org

[16] Synopsys Co-Centric Studio product webpage, http://www.synopsys.com/products/cocentric_studio/cocentric_studio.html

[17] Virtual Socket Interface Alliance Webpage, http://www.vsi.org

[18] www.xess.com

[19] Xilinx , www.xilinx.com

[20] A. Varma, J. R. Armstrong, and J.M. Baker, "A SystemC GSM Model for Hardware/Software Co-design," Proceedings of HDL 2002, San Jose, March 2002, pp 41-45.

[21] Wilson, Introduction to Graph Theory, Academic Press, NY, 1972.

[22] J. R. Armstrong, and S. Agrawal, "Denotational and Operational Semantics of Process Model Graphs," Submitted. Available at http://www.visc.vt.edu/armstrong/jra_vt_pmg.pdf

Total Hot Spot Management from Design Rule Definition to Silicon Fabrication

Soichi Inoue, Toshiya Kotani, Shigeki Nojima, Satoshi Tanaka, Kohji Hashimoto, and Ichiro Mori

Process & Manufacturing Engineering Center, Toshiba Corporation, Semiconductor Company
8, Shinsugita-cho, Isogo-ku, Yokohama, Japan

Abstract

The total flow consisting of manufacturability checks (MCs) was proposed to obtain stable pattern formation without any hot spots under low-k_1 lithography condition. The MC in the lithography design stage finds out the hot spots in the model patterns created by a compaction tool, and helps to define complex design rule. The MC in design stage is important to refine the design rule and improve PPC using the actual layout patterns. Even after placement and routing, there are some risks of remaining hot spots due to topology change at the cell boundaries. It is considerably necessary to feed forward the hot spots to the mask process and wafer process. Total hot spot management from design rule definition to silicon fabrication enables to clean up the "hot spots".

1 Introduction

The shrinkage of large-scale integrated circuits (LSI) has kept around 75 % per 1.5 years constantly. However, pattern formation technologies recently don't catch up with the requirement due to their technical difficulties. Needless to say, an optical lithography is a core technology in them, which resolution is determined by the exposure wavelength, λ, and numerical aperture of the exposure tool, NA. The normalized minimum line width of the LSI, k_1, is written by

$$k_1 = L/(\lambda/NA),$$

where L is the minimum line width of the LSI. The k_1 is decreasing year by year. The low-k_1 lithography gives large optical proximity effect resulting in remarkable corner rounding, line end shortening, and critical dimension (CD) variation through pitch, etc. Besides, the loading effects not only on the optical matter but also on other pattern formation processes, naming generically a process proximity effect (PPE), are not negligible with decreasing in the feature sizes. Accordingly, a process proximity correction (PPC) technology is indispensable to obtain sufficient CD accuracy and pattern fidelity the designers need. However, the PPC technology does not always give solution for any patterns they draw unfortunately. The issue is that it is considerably difficult to make a perfect design rule for any pattern topology to satisfy condition for the solution by the PPC. This is due to the complexity of PPE resulting from all physical effects from mask making to silicon fabrication. There are some risks of "hot spots", which means a critical point having a critical process window for manufacturing, if the designers draw layouts following the imperfect design rule.

In this report, a new methodology from design rule definition to silicon fabrication in low-k_1 lithography condition was presented to manage the hot spots totally. Especially, design rule definition and manufacturability check (MC) in design stage are key technologies in the method.

2 Design rule crisis in low-k_1 lithography

Figure 1 shows available optical lithography technologies as a function of minimum line width. The normalized line width, k_1, is also shown in the figure. Optical lithography has been making great effort to catch up with the requirements for resolution, resulting from severe shrink of CDs of LSI circuits, by shortening the exposure wavelength as well as enlarging NA of exposure tools; nevertheless, k_1 factor is decreasing as the pattern size is shrinking. Figure 2 shows the impact of low-k_1 on pattern fidelity. The pattern fidelity is decreasing as decreasing in

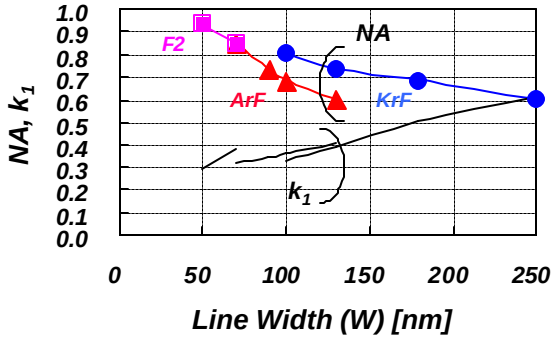


Fig. 1 Available optical lithography technologies as a function of minimum line width

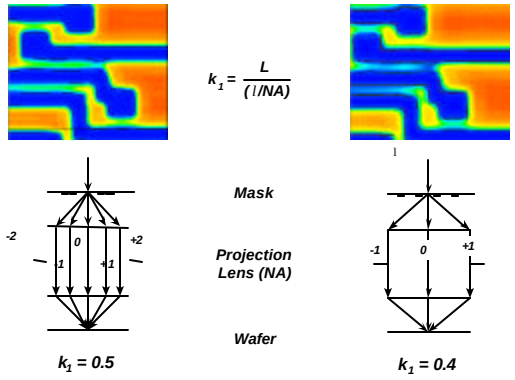


Fig. 2 Impact of low-k_1 on pattern fidelity

k_1. It is found that there is a hot spot even if the layout is drawn without violating a minimum spacing rule, and is corrected by a perfect PPC. The issue is that the positions and degrees of the hot spots don't depend only on the space width and/or neighboring line width. A simple minimum spacing and line rule can't clean up the hot spots at all. Optical lithography simulation tells us the positions of the hot spots, but it is critical to define the adequate rule for the dangerous topology; not too aggressive, not too moderate.

3 Strategy and the total development flow for DfM

What total development flow from design rule definition to silicon fabrication should be under the condition? Figure 3 represents the schematic illustration of the strategy for design for manufacturability (DfM). The horizontal axis shows the k_1 factor, and the vertical axis is the complexity of the design rule. Conventionally, the design rule was simple, e.g. minimum line width, minimum spacing, etc., due to the small PPE in the large k_1 factor world, around 0.5. In low-k_1 lithography, the simple design rule can't guarantee the resolution of all kind of pattern variations due to its large PPE. There are two main approaches for solving the issue. One is the complex design rule approach, and another is the simulation based MC approach. MC compares a target layout and a simulated image of the PPCed mask data, and highlights portions of greater difference than a criterion between the two as hot spots. The former approach yields few hot spots due to the complex design rule, but it takes long time to define the design rule. Besides, the complex design rule makes designers feel hard to draw layout. On the contrary, the latter approach yields relatively many hot spots due to the simple design rule, and makes the designers waste time and labor to repair the hot spot, but it takes short time to define the design rule. The realistic approach should be the combination of the two approaches described above. First of all, lithographers should define as accurate design rule as they can, using as many pattern variations as they can; not only simple patterns but also pseudo-real device patterns at the stage of design rule definition. Secondary, the design rule should be refined by the MC in the design stage using the actual device patterns. For memory device case, there is an advantage of the MC because it is important for memory device to reduce cost by shrinking the chip size.

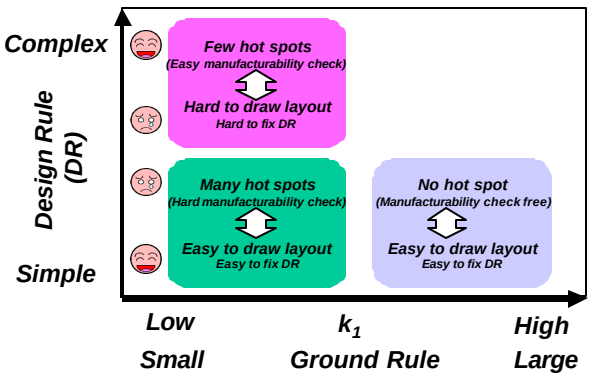


Fig. 3 Schematic illustration of the strategy for design for manufacturability (DfM)

Figure 4 shows the total development flow for DfM based on the concept described above. First of all, device engineers and designers show lithographers CD tolerance

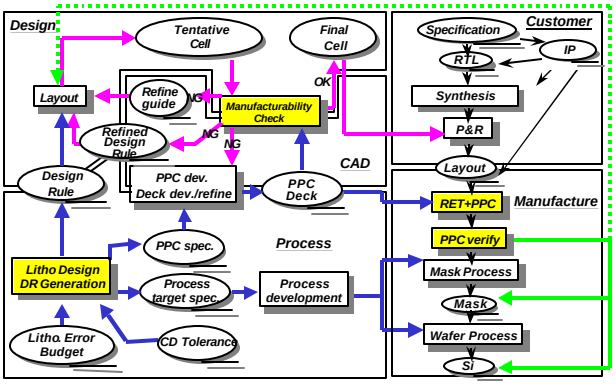


Fig. 4 Total development flow for DfM

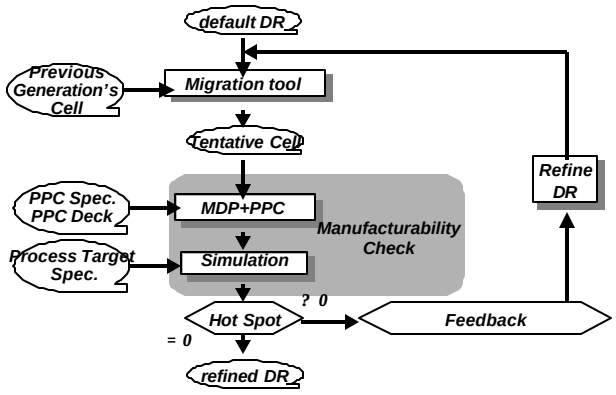


Fig. 5 Flow of design rule definition

for next generation's device. In order to obtain sufficient CD accuracy, the process engineers, especially lithographers carry out the simulation and experiment to determine the lithography method and conditions having sufficient CD accuracy less than the estimated CD tolerance. They define the drawing regulation, forbidden rule to obtain sufficient accuracy of pattern formation under the estimated lithography method and conditions. At the same time, they require target specifications for all kinds of process technologies, e.g. resist and etching performance, PPC method and algorithm, RET technologies, and so on. We call this procedure "lithography design." The designers start to layout according to the agreed design rule. The balance between the design rule and the target process technologies are considerably significant. If the target process technologies are too severe and the resultant design rule is too aggressive for the next generation's device fabrication, it is hard to make. On the contrary, the chip is not highly competitive if the target process technologies are not severe and the resultant design rule is too moderate. In this timeframe, there is no actual layout. We use previous generation's layout as the model patterns of the next generation to achieve accurate design rule. The detailed flow is described later.

MC for the tentative cells following the design rule is carried out in the design stage. The MC flow consists of PPC and simulation for pattern formulation including lithography. There are three feedback passes in case there are some hot spots in the cells. One is to improve PPC deck and PPC parameters, another is to repair the hot spots, and the other is to refine the design rule itself. The sort of device, timing, and so on dominates which pass is a main stream.

Place and route system uses the complete cells to build full-chip physical layout. Design rule checker (DRC) finds out the design rule violation and MC finds out the hot spots in the full chip layout data. MC compares a target layout and a simulated image of the PPCed mask data, and highlights portions of greater difference than the criterion between the two as hot spots. There is an opportunity to occur the hot spots even in this stage. In this case, there are few chances to feed back the hot spots to the design procedure. Therefore, it is considerably necessary to feed forward the hot spots to the mask process and wafer process. A metrology tool to navigate any hot spots portion and to measure the pattern quickly is essential for the flow in low-k_1 lithography world.

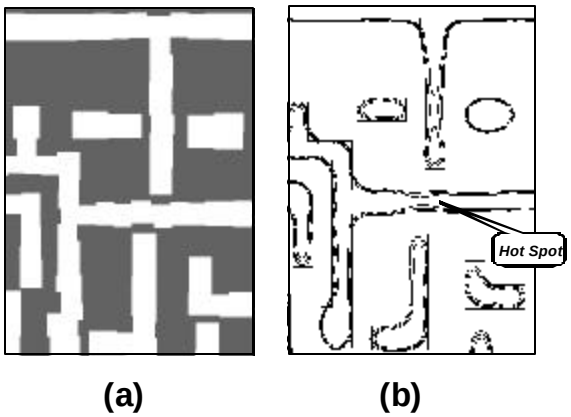


Fig. 6 (a) PPCed mask patterns for the shrunk patterns from 130-nm node to 65-nm node following tentative design rule for 65-nm technology node , (b) the simulation result for the PPCed mask patterns overlaid by the target patterns

4 Lithography design and design rule definition

Figure 5 shows the flow of design rule definition. In this timeframe, there is no actual layout. We use previous generation's layout as the model patterns of the next generation to define accurate design rule. After determining lithography method, condition, and tentative design rule with simple representative patterns, the previous generation's primitive and/or macro cell patterns are shrunk by any compaction tools, e.g. migration tool, following the tentative design rule. The shrunk patterns are converted into PPCed mask patterns. The PPCed mask patterns are transferred by simulation for pattern formulation including lithography, and compared with the target patterns to find the hot spots. If there are some hot spots, the design rule is refined to eliminate the hot spots. The iteration of the procedure makes the design rule eliminate the hot spots. Figure 6 shows the example of the procedure. Figure 6(a) shows PPCed mask patterns for the shrunk patterns from 130-nm node to 65-nm node following tentative design rule for 65-nm technology node. Figure 6(b) shows the simulation result for the PPCed mask patterns overlaid by the target patterns.

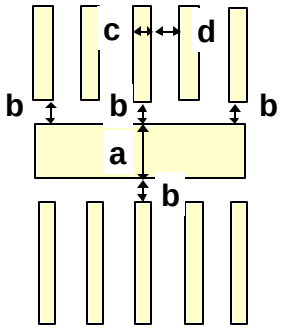


Fig. 7 Simplified and modeled patterns of the layout shown in Fig. 6 to obtain critical CD of line width "a"

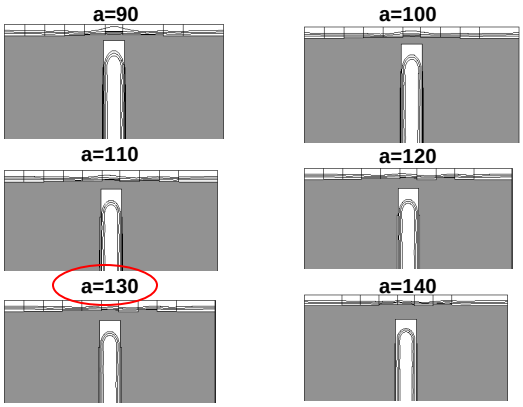


Fig. 8 Simulation results of the feature "a" for b = 1000 nm

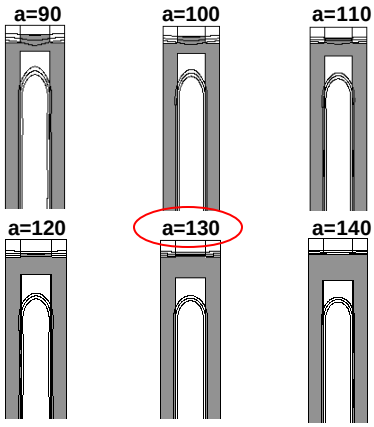


Fig. 9 Simulation results of the feature "a" for b = 90 nm

There is a hot spot for line narrowing at the portion nipped by two bars. The pattern fidelity at the hot spot is getting worse than the other portion with increasing in dose error. Next, it is necessary to find the adequate line width not to be line narrowing even with dose error. The hot spots patterns are simplified and modeled as shown in Fig. 7, and the adequate design rule is obtained by the parametric running of the patterning simulation. Figures 8 and 9 show the results. It is found that the line width of the design layout, a, should be more than 130 nm to maintain the corresponding line width on the wafer. The design rule has grown to cover as many pattern variations as the lithographers can in the lithography design stage according to the procedure.

5 Manufacturability check

Figure 10 shows the detailed procedure for MC in design stage. The designers draw the tentative layout following the design rule defined by the lithography design process. The patterns are converted into PPCed patterns, and the PPCed mask data are checked by simulation based verification tool to find the hot spots. There are three-feedback pass in case there are some hot spots in the cells. One is to improve PPC deck and PPC parameters, another is to repair the hot spots, and the other is to refine design rule itself. The improvement of PPC deck and PPC parameters to eliminate the hot spots should be prioritized rather than the other feedback passes. However, it is impossible to say definitely that there is no probability to feed back to the repair of the hot spots and/or the refinement of the design rule in the development stage. Figure 11 shows an example that changing the PPC parameter on the minimum jog length can eliminate the hot spot. The upper figures show the PPCed patterns of gate level of SRAM cell for 90-nm technology node overlaid with the active patterns, and the bottom ones are the corresponding simulated images on the wafer printed by ArF exposure tools. The longer jog length was not enough to control local CD variation at the curved

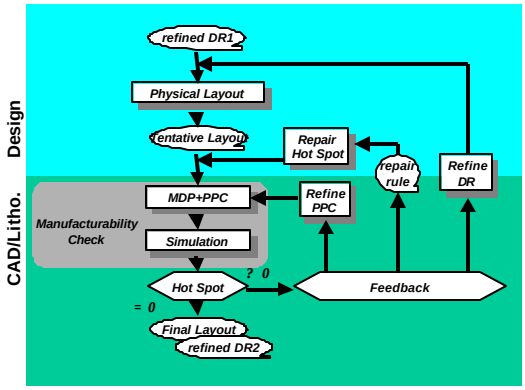


Fig. 10 Detailed procedure for MC in design stage

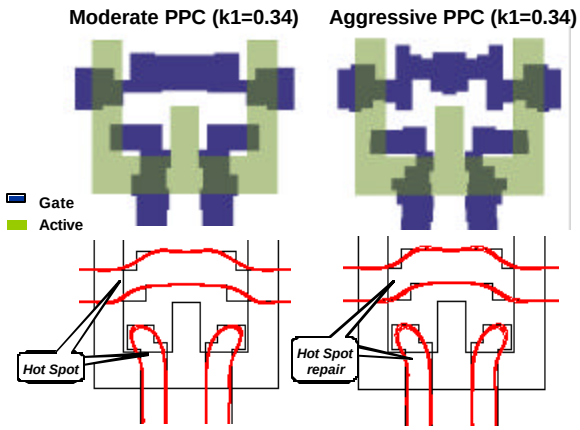


Fig. 11 Hot spot elimination by shortening jog length

portion of the gate patterns. The shorter jog length improved the local CD controllability. However, the shorter jog makes the reticle writing time elongate for a variable shaped beam type electron beam writer. It also makes other issues for reticle manufacturing. Shortening the minimum jog length is relatively easy way to avoid the conflict between design layout and PPCed mask layout. However, it should be determined very carefully by total optimization for efficiency and manufacturability. The MC procedure during the development stage makes the design rule robust. For volume production stage, there will be a rare case for hot spot repair or design rule refinement due to the robust design rule.

6 Hot spot management on reticle and wafer

Even after placement and routing process, there are some risks of remaining hot spots. This is because cell placement changes the pattern topology at cell boundaries against the individual cell. The change of pattern topology results in the change of PPE. In this timeframe, there are few chances to feed back the hot spots to the design procedure. Therefore, it is considerably necessary to feed forward the hot spots to the mask process and wafer process [1][2]. The feed forward pass should be a way of emergency evacuation for rare case of unavoidable hot spots. The accurate and robust design rule defining procedure being consistent with the manufacturability is extremely important for the low-k_1 lithography.

Figure 12 shows top-down scanning electron micrographs of hot spots pattern searched by the navigation system with the hot spot map on the CAD layout. The CAD layout is overlaid with the micrographs. The system makes it possible to run the production lots under the ultra-narrow lithography window condition.

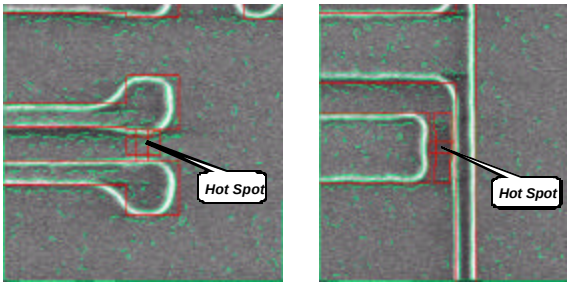


Fig. 12 top-down scanning electron micrographs of hot spots pattern searched by the navigation system with the hot spot map on the CAD layout. The CAD layout is overlaid with the micrographs

7 Concept of tolerance based PPC

To begin with, what are the criteria of the hot spot? The designers and device engineers usually give the lithographers the allowable CD error of +/- 10% of the main feature sizes. Is it really necessary for any pattern edge of any level to position within +/- 10 % of the target position? The device does work well even if the part of achieved edge positions are out of the tolerance. This is the reason why there are too many pseudo-hot spots, that is noises, which does not affect on the real device performance. Figure 13 shows the schematic illustration of LSI design with the tolerance of each pattern edge being analogous to the mechanical design form. The accurate tolerance of each edge position is quite necessary to eliminate the noises for MC [3][4].

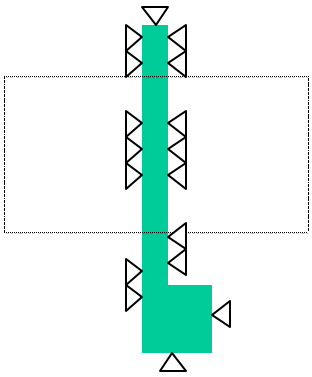


Fig. 13 Schematic illustration of LSI design with the tolerance of each pattern edge being analogous to the mechanical design form

8 Summary

The total development flow consisting of three MCs was proposed to obtain stable pattern formation under low-k_1 lithography condition. It is hard to define accurate and robust design rule under the condition because there are large and complicate PPE. Of course, there is no actual process and layout in the design rule defining stage. The compaction tool helps to make model patterns of the next generation. MC finds out the hot spots in the model patterns, and helps to define complex design rule. However, the design rule might be imperfect due to large PPE. MC in design stage is important to refine the design rule and improve PPC flow using the actual layout patterns. It is essential for the designers not to have a heavy load. Even after placement and routing, there are some risks of remaining hot spots due to topology change at the cell boundaries. In this timeframe, there are few chances to feed back the hot spots to the design procedure. Therefore, it is considerably necessary to feed forward the hot spots to the mask process and wafer process. Total hot spot management from design rule definition to silicon fabrication makes it possible to obtain higher yield for 65 nm and beyond generations.

References

[1] R.Matsuoka, *Electronic Journal (in Japanese)*, pp.104-105, July 2000

[2] K. Hashimoto, T. Sutani, T. Ikeda, S. Nojima, S. Inoue, "Experimental verification methodology for model-based process proximity correction (PPC)", *Proc. SPIE* Vol.5040, 2003, to be published

[3] M. Rieger, J. Mayhew, J. Li, and J. Shiely, "OPC strategies to minimize mask cost and writing time", *Proc. SPIE* Vol. 4562, 2002, pp.154-160

[4] R. Lugg, D. Beale, J. Haung, and M. Rieger, "Adaptive OPC with a Conformal Target Layout", *Proc. SPIE* Vol. 4691, 2002, pp. 1091-1096.

Code Generation from a Single Source Structural Specification

Claus Schneider, Wilfried Gänsheimer, Erich Schwenk
Micronas GmbH
Frankenthalerstr. 2
81539 Munich, Germany
<first>.<last>@micronas.com

ABSTRACT
A method and a tool to automatically derive top-level hardware description, low-level driver software and documentation out of a single source structural specification is presented. In contrast to system-level design tools this approach focuses on structural properties like interconnect, hierarchy and control/status registers instead of refining a functional specification. A table-based specification format was chosen to enable efficient data entry and review. Due to automatic code generation consistency between specification and design is guaranteed and error-prone manual transformations can be avoided. This paper focuses on specification and hardware generation.

Keywords
Single Source, Structural Specification, Code Generation, Table, Spreadsheet

1 INTRODUCTION

System-level design is a highly interactive process. A huge amount of experience is needed and many trade-offs have to be made during design space exploration. Therefore, it is extremely difficult to automate this process, and most of the refinement steps from specification to design have to be done manually. Depending on the purpose of a model different languages may be used. In reality an executable specification is only available for parts of the design. These models were designed to perform algorithm evaluation and may serve as reference models for verification, but in most cases they are not suitable for direct refinement to a design realization. In addition, initialization is done differently in the abstract model than in the concrete design, and IO and test structures are not covered at all.

Due to these constraints we decided to focus on structural properties like interconnect, hierarchy and control/status registers instead of refining a complete functional specification.

In this approach, functional blocks are treated as black boxes for top-level integration. The control and status registers are separated from the functional block to enable different address mappings and integration into different chip architectures (e.g. parallel control bus vs. serial control channels). Hierarchy information is separated from interconnect specification to enable easy hierarchy changes and flat (source to sink) interconnect analysis across hierarchies. Another structural issue is the multiplexing of a huge number of internal signals to a limited number of external IO pads.

A tabular specification format was chosen for top-level interconnectivity, hierarchy, IO multiplexing and control/status registers. The major benefit of this approach is to maintain a single source of specification information. All code and documentation that can be directly derived from the specification is generated automatically to avoid any error-prone manual interaction and to guarantee for consistency between specification, documentation and design.

2 RELATED WORK

The proposed table-based structural specification is closely related to traditional text-based or schematic design entry but is intended as a method to bridge the gap between concept and design. In the first step the concept engineering captures the register specification and basic architecture definition (major control and data flow). Afterwards the specification is extended by implementation details to be ready for automatic code generation.

Concept engineers prefer spreadsheets whereas designers often stick to their favorite text or schematic editor to capture the design. For this reason, the major issues related to these methods will be discussed briefly. While text-based design entry requires focus on syntax and formatting in schematic entry much time is spent on doing layout of blocks and wires. Both methods require a significant amount of effort, which is not necessary to reach the goal – a compact specification of logical interconnect. On one hand block diagrams are very useful to understand and to discuss structural design issues but on the other hand they are very time consuming to create and to maintain. A solution to this problem is the creation of HDL code out of a compact logical interconnectivity specification and the visualization with generated block diagrams.

Meanwhile, sophisticated debug tools [1] generate acceptable hierarchical block diagrams and allow analysis

of fan-in and fan-out cones of signals across hierarchies. Unfortunately, the analysis capabilities are restricted to debugging and tracing of single signals. They are neither intended to a systematic review of all top-level interconnections nor for design entry and code generation.

The table-based capture of structural specification information fills this gap between concept and design. It provides a means for a compact interconnect specification without having to deal with HDL syntax nor schematic layout issues. The HDL code is generated automatically according to predefined coding guidelines and naming styles. Schematics can be generated dynamically on demand by commercial tools.

The hierarchy that is required by text and schematic entry to handle the complexity is kept separate from the interconnect definition. Filter capabilities in spreadsheets instead of hierarchy can be used to deal with complexity. By keeping hierarchy separate from interconnect it can be changed easily and signals can be traced at a flat top-level from source to all sinks across hierarchies. While text and schematics are block-oriented the table provides also an interconnect-oriented view.

A commercial interface-based design tool [2] provides a table-based interconnect entry but there is no separation of interconnect and hierarchy nor there is dedicated support for register and IO multiplex specification. Especially the support for register, IO, hierarchy and interconnect specification in a single source for automatic generation of derived code and documentation is seen as the key benefit of the proposed tabular specification.

3 MAPPING OF SPECIFICATION OBJECTS TO DESIGN INSTANCES

The single source structural specification consists of four tables (Interconnect, Hierarchy, IO, Register). Out of these tables, top-level hardware structure, HW/SW interface (register structure + low-level driver software) and documentation are generated.

3.1 Software and Documentation

The lowest layer of the driver software is generated out of the register sheet only. The purpose of this layer is to hide the mapping of logical control/status parameters to physical bit slices of one or several registers. The higher software layers can access these control parameters without having to care about address space mapping and masking operations. For documentation purposes various tables are generated that list registers ordered by address, alphabetically or in functional order. In addition an IO mapping table is generated to summarize the IO multiplex options.

This paper focuses on the generation of the top-level interconnectivity. The basic architecture shown in Figure 1 will be taken to explain the method.

3.2 Basic Target Architecture

Typically, our chips are structured in the following way:

- CHIP: Pad frame containing the pad instances
- TOP: IO control Cells (IOC) that perform multiplexing, grouped into N/S/W/E for layout purposes
- CORE: Subsystems consisting of functional blocks, registers and interface blocks. For mixed VHDL/ Verilog designs wrapper blocks are necessary.

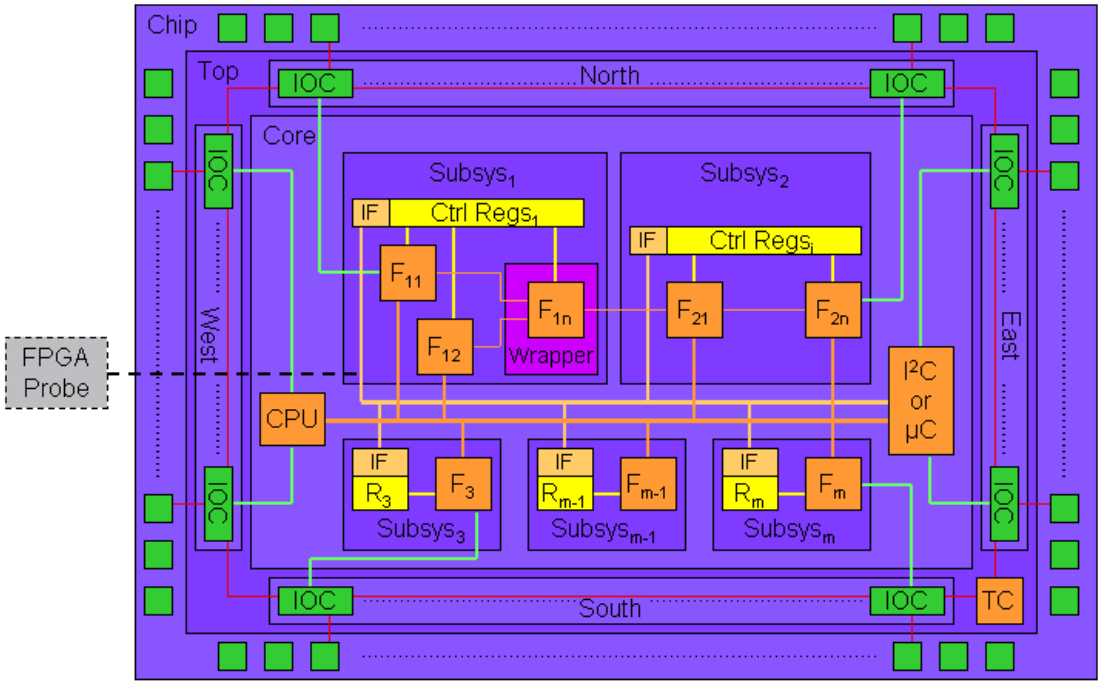


Figure 1: Basic Target Architecture

Control-oriented designs may consist of an internal CPU and a micro controller, both with their own bus. Designs for signal processing may only consist of a dedicated data path and a serial programming interface (e.g. I^2C). A test controller (TC) and a chain connecting all IOCs (e.g. boundary scan or NAND tree) denote the presence of test logic.

Not all connections have to be specified explicitly in the interconnect table for the generation of the top-level interconnectivity. Connections are automatically generated between core signals and IOC blocks by referencing signals from the interconnect table inside the IO table.

An example is the connection between F_m and IOC (green). Additionally, so-called implicit signals are generated by assignment of a register to a functional block in the register table, e.g. the connection between R_m and F_m (yellow). All other connections have to be specified explicitly (orange). User defined macros, e.g. for signal bundles, are supported to increase the abstraction level of the specification. Further reduction of typing can be achieved for regular interconnect structures, like boundary scan chains, by the usage of regular expression matching for instance names (red). These features will be described in more detail in the following sections.

All interconnections are described directly between functional blocks, registers, IO cells and pads. These blocks are treated as black boxes for top-level integration. The methodology can be applied in a hierarchical manner because there is no assumption on the granularity of the leaf blocks. A module designer may treat adders, FIFOs or FSMs as leaf blocks. For chip-level integration these internally developed modules or intellectual property blocks from third parties may be treated as leaf blocks (black box). Intermediate hierarchies, e.g. CHIP, TOP, CORE and subsystems, or simulator dependent wrapper frames between VHDL and Verilog are not considered for the interconnectivity specification. The hierarchy is specified in a separate table and used only for HDL code generation. While hierarchy is required for text or schematic capture it is replaced by filter capabilities of spreadsheets. Therefore, several attributes, assigned to the signals, can be used to focus on a selected aspect of the design. Besides a bundle name to group signals a class attribute is provided to select between data, control, test, clock or reset signals. This capability is especially useful for systematic reviews: All signals that belong to a certain category, e.g. test, can be traced from source to all destinations across all hierarchies. This can be done in a systematic manner rather than randomly one signal after another in a schematic tool.

3.3 Interconnect

There are four different types of interconnect specification objects that directly correspond to VHDL or Verilog design objects. A signal, in Verilog called net, is a connection between leaf blocks that has at least one driver (source) and at least one load (destination). A port describes an interface of the top-level and can be of mode *in*, *out* or *inout*. Constants are used to parameterize instances. Generics provide a means to pass constant parameters to leaf modules. The basic source of interconnect specification is the interconnect table, a list of interconnect objects. An example is shown in Figure 2.

The first row holds the tags identify the columns. This allows for swapping or inserting columns without the need to modify the table parser. There are two additional columns that are not shown in Figure 2: There is a *description text* column and an *ignore* column that can be used to mark a row as a comment line. A mode attribute distinguishes between different objects: Signal, port,

::gen	::bundle	::class	::clock	::type	::high	::low	::mode	::name	::out	::in
	RGB	Data	CLKF	logic	7	0		SIG_R	MODA/R	MODC/R
	RGB	Data	CLKF	logic	7	0		SIG_G	MODA/G_LO(3:0)=(3:0), MODB/G_HI(3:0)=(7:4)	MODC/G(5:1)=(7:3), MODD/G(7:5)=(2:0), MODE/G=(0)
	RGB	Data	CLKF	resolved	7	0		SIG_B	MODA/B(7:0), MODB/B(7:0)	MODC/B(7:0)
$i (1..147), /IOC_(\w+)_$i/	NAND_TREE	Test	TCK	logic				NAND_OUT_$i	IOC_$1_$i/NAND_OUT	IOC_$1_{$i+1}/NAND_IN
/IOC_(\w*O)_(\d+)/	PAD_CTRL	IO		logic				PAD_DO_$2	IOC_$1_$2/DO	PAD_$2/DO
MH		StdIF	$1	YCrCb_$2_$3_$4					$5	$6
MD	$5_$6	StdIF C	$1	logic				$5_AP	$5/AP	$6/AP
MD	$5_$6	StdIF C	$1	logic				$5_HSYNC	$5/HSYNC	$6/HSYNC
MD	$5_$6	StdIF C	$1	logic				$5_VSYNC	$5/VSYNC	$6/VSYNC
MD	$5_$6	StdIF D	$1	logic	$4	0		$5_CB	$5/CB($4:0)	$6/CB($4:0)
MD	$5_$6	StdIF D	$1	logic	$3	0		$5_CR	$5/CR($3:0)	$6/CR($3:0)
MD	$5_$6	StdIF D	$1	logic	$2	0		$5_Y	$5/Y($2:0)	$6/Y($2:0)
MX		StdIF	CLKD	YCrCb_8_7_7					BLE	CST
				integer			C		6	FIFO/DEPTH
/(IOC_\w+_\d+)/	TOP	IF		integer			G	WIDTH	8	$1/WIDTH
	TOP	IF	CLKF	logic	WIDTH-1	0	I	DATA_IN		MODC/Y

Figure 2: Interconnectivity Table

generic and constant. The name of each interconnect object has to be unique for the flat interconnect specification. Objects are further described by a type and range (for vector types), bundle name and class attribute for sorting and filtering purposes and an associated clock for signals and ports. At the beginning of the table different examples for signal definitions are shown. In the single bus example the width of the ports can be omitted when identical to the signal width. The split bus example shows a signal where one bit slice is driven from module A while the second slice is driven from module B. At the destinations the signal branches to modules C, D and E. For this connection the bits 7 down to 3 of the signal are mapped to bits 5 through 1 of the port G of module C. Tri-state bus connections can also be specified. The usage of regular expression matching and a counter loop for a very compact specification of a regular interconnectivity structure is shown for a NAND tree.

The algorithm to generate the regular interconnectivity structure is shown in Figure 3 in PERL syntax.

```
foreach my $i (1..147) {
  foreach $inst (%inst_hash) {
    if ($inst =~ /IOC_(\w_\w+)_$i/) {
      &add_conn(NAND_OUT_$i,
                IOC_$1_$i/NAND_OUT,
                IOC_$1_{$i+1}/NAND_IN, .... )
    }
  }
}
```

Figure 3: Regular Expression Matching

For each instance matching the regular expression within the specified loop a connection is added. The variables $1 (matched part of instance name), $i (loop counter) and any expression like {$i+1} will be evaluated and replaced by their actual value. For simplification the evaluation is not shown in Figure 3. After the NAND Tree an example for multiple point-to-point connections of the data out ports between all IO control cells and pad cells is shown.

For standardized interfaces like micro controller busses or data path connections macros can be defined to reduce typing effort and support abstraction. In Figure 2 a data path example is shown consisting of control signals (AP, HSYNC, VSYNC) and data signals (CR, CB, Y). The clock parameter ($1) of the macro header (MH) is just copied to all signals of the bundle in the macro definition (MD). The data widths ($2, $3, $4), the source ($5) and destination ($6) blocks will be replaced in the macro execution (MX) by the actual values.

Finally, a constant definition to parameterize a FIFO depth, a generic definition to parameterize the width of IO cells and an input port whose width depends on the generic parameter are shown in Figure 2.

3.4 Hierarchy

Interconnectivity and hierarchy are separated to enable flat analysis of top-level interconnectivity. For each level of hierarchy all instances are listed with a reference to the parent hierarchy. The most important instance attributes like entity name, configuration name and language (VHDL or Verilog) are specified, Figure 4.

In the hierarchy table regular expressions for instance matching can be used for grouping of instances. This feature is shown for pads and IO control cells. In the PADS hierarchy all instances matching "PAD_" followed by a number between zero and 147 are grouped. These instances were generated from the IO table described in the next section. The IO table also defines the pad type, i.e. the entity name. To avoid duplicated information and to enable grouping by regular expression matching the entity table cell for pad instances is left empty. The generator tool gets the correct entity name for each pad instance from the IO table to expand the hierarchy table. A similar mechanism applies to the IO control cells. Here, the tool automatically generates an entity name according to the given naming rules.

Another feature of the hierarchy table is the specification of design variants. A special column (::variant) in the table is reserved for a list of variant names for each instance. Using this feature the PROBE module can be included in the FPGA variant for debugging purposes and can be dropped in the final Chip (Default) or Emulation version. We also used variants to replace various data path modules by bypass modules to reduce the gate count for the FPGA prototype. In this scenario modules that were not relevant for some tests were bypassed.

::gen	::variants	::parent	::inst	::lang	::entity	::config
	Default	TESTBENCH	MYCHIP	VHDL	MYCHIP	MYCHIP_RTL_CONF
	FPGA	TESTBENCH	PROBE	VHDL	PROBE	PROBE_RTL_CONF
	Default	MYCHIP	PADS	VHDL	PADS	PADS_RTL_CONF
	Default	MYCHIP	IO_SOUTH	VHDL	IO_SOUTH	IO_SOUTH_RTL_CONF
	Default	MYCHIP	TOP	VHDL	TOP	TOP_RTL_CONF
$i (1..147), /PAD_$i/	Default	PADS	PAD_$i	VHDL		${::entity}_structural_conf
$i (1..42), /IOC_(\w_\w+)_$i/	Default	IO_SOUTH	IOC_$1_$i	VHDL		${::entity}_RTL_CONF

Figure 4: Hierarchy Table

3.5 IO Structure

Most of the designs are pin-limited and require IO multiplexing in mission mode as well as in test mode. While multiplexing options can be described very efficiently in tabular format the manual HDL implementation of all interconnections is very inflexible, time consuming and error-prone. The automation of this task is based on predefined IO Control cells (IOC). Different types of interfaces (e.g. GPIO, DRAM) require different IOCs.

An example for an IOC is shown in Figure 5.

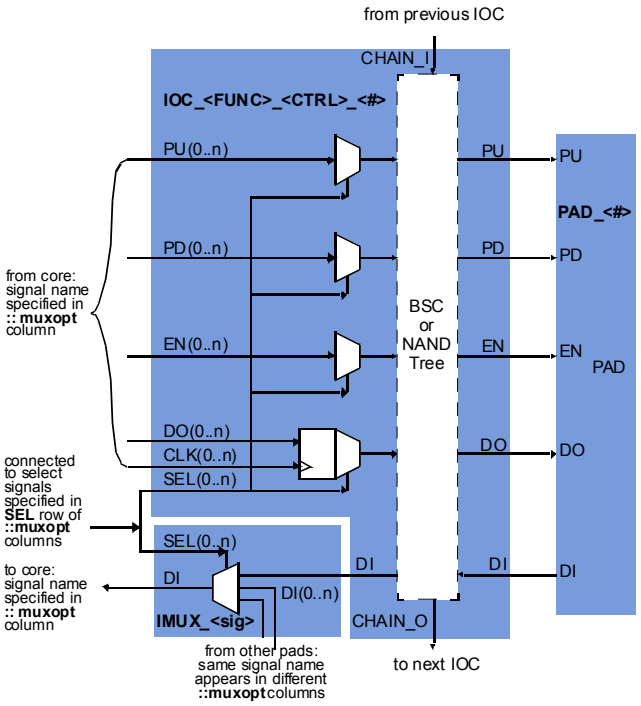


Figure 5: IO Control Cell

The IOC contains multiplexing logic, output registers and it may contain also test logic like a NAND tree or a boundary scan chain. Regular interconnect structures like the NAND tree and the connections between IOC and pad can be explicitly described in the interconnect sheet using regular expression matching and loops. All connections of core signals to the IOC multiplexer inputs have to be described in the IO table. A small example of an IO table is shown in Figure 6. It defines an input pad TM, two supply pads (VDD, VSS), an IO pad (GPIO) and another IO pad with pull up (PCK). Each row of the table describes one pad, its attributes, the IO control cell instance and the IO multiplexing.

::pad	::type	::iocell	::port	::name	::muxopt	::muxopt
			SEL	PAD	IOSEL_0	IOSEL_T
1	WC3I80	IOC_G_I	DI	TM	TESTMODE	TESTMODE
2	WVV3IO			VDD		
3	WVV0IO			VSS		
4	WC3B60	IOC_R_IO	DI, DO, EN	GPIO	GPIN, GPOUT, GPEN	SCAN_IN
5	WC3BC0	IOC_R_IOU	DI, DO, EN, PU	PCK	, PCK, PCK_EN	TCK, , , '1'

Figure 6: IO Table

For the multiplexing a list of ports (::port) of the IOC is assigned to multiple lists of core signals (::muxopt). The direction of the pad is determined by the ports of the IOC, that are used in this multiplex option: Data In (DI), Data Out (DO), Output Enable (EN). In addition, Pull Up (PU) and Pull Down (PD) resistors in the pad can be controlled. A special row defines the control signals of the select inputs (SEL) of all IO multiplexers for each multiplex option (e.g. IOSEL_0, IOSEL_T). These select signals have to be explicitly defined in the interconnect sheet. The signals referenced in ::muxopt columns can be either explicitly defined signals from the interconnect table or implicitly defined signals that will be generated or expanded by the tool (e.g. signals from/to register blocks).

3.6 Register

Control and status registers are separated from the functionality of the processing units. This approach often referred to as interface-based design [3] or separation of synchronization and functionality [4] gives the maximum flexibility to adapt the units to different on-chip communication structures like control busses (e.g. µC) or control channels (e.g. ::type = I2C). An example of the register table is shown in Figure 7. Each row of the table defines the bits (::b) of a control parameter and it's attributes. The generator supports different access modes (::rw), auto address increment (::auto), update mechanisms (::sync) and reset values (::init). Several control parameters can be merged to a register by using the same sub address (::sub). Furthermore, registers can be grouped into register blocks (::interface). The grouping may be based on clock domains, update domains, or logical function. The HDL code generation works similar to the approach for the IO control cells. Different types of predefined registers with various generic parameters are instantiated and

::type	::sub	::interface	::block	::rw	::auto	::sync	::clock	::reset	::b	::b	::b	::b	::b	::b	::b	::b	::init	::view	::comment
I2C	30	I2C_VIDEO	RGBF	W	Y	VS_F	CLKF	RSTF			BRT.5	BRT.4	BRT.3	BRT.2	BRT.1	BRT.0	20	Y	Brightness Adjust
I2C	30	I2C_VIDEO	RGBF	W	Y	VS_F	CLKF	RSTF		DIGSEL							0		Select digital or analog input
I2C	30	I2C_VIDEO	RGBF	W	Y	VS_F	CLKF	RSTF	YUVSEL								1	Y	Select YUV or RGB input

Figure 7: Register Table

parameterized according to the register attributes defined in the table including clock and reset signals. The control parameters of the registers are connected to the functional blocks (::block) that are specified in the table. These are implicit connections and the corresponding ports at the functional blocks are automatically added. During this generation step parameters that are spread over multiple registers are combined to one signal. The hierarchical register blocks are also automatically generated. Finally, the register table includes attributes (::view, ::comment) for driver software and documentation generation.

4 IMPLEMENTATION

4.1 Data Model

The ***M**icronas **I**nterconnect Spec e**X**pander* converter tool (**MIX**) closely models the required high-level design objects like *instances*, *signals, IO cells* and *registers. Perl* provides powerful regular expression matching operators and text processing capabilities. Design objects and other data structures are mapped to hashes and arrays. The relation between different design objects is shown in an UML-like format in Figure 8.

A chip consisting of instances and signals can be seen as an abstract class. Only a concrete variant can be generated. Instances and signals are linked by ports. Hierarchical blocks can be nested while leaf blocks are not further decomposed. A register block is hierarchical, consisting of registers. IOCs, pads and functional blocks are leafs in the hierarchy tree. There are two types of signals: Implicit signals are automatically generated out of the control parameter specification of the register definition. These are the signals between functional blocks and the bit-fields of the control/status registers. Explicit signals are directly specified in the interconnect sheet as plain signals or they are expanded from macros or regular expressions.

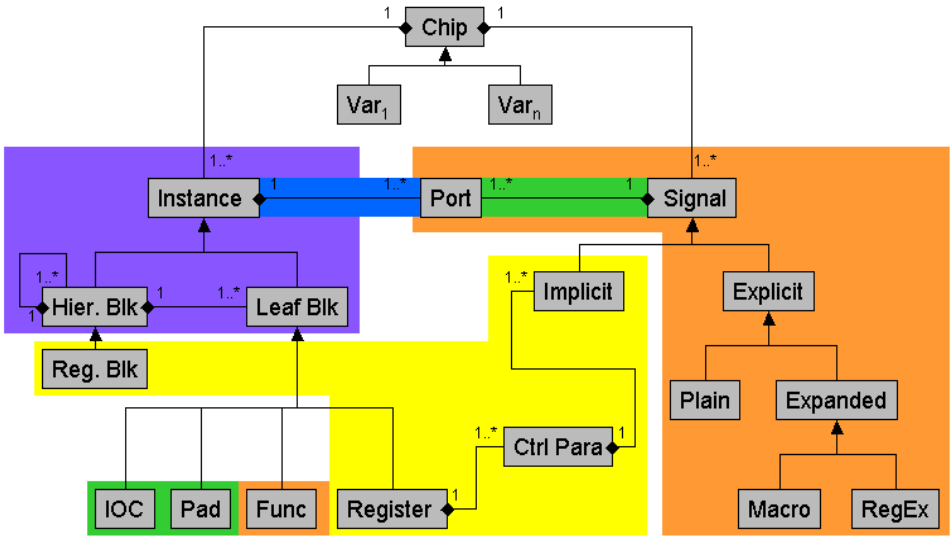


Figure 8: Data Model

4.2 Step 1: Read Input Description, Generate Intermediate Data

The converter tool starts up with some basic initialization. A set of company and project specific design guidelines and templates are read in and added to the tool environment. Design guidelines define standard naming conventions (e.g. pre- and postfixes) and other rules applied in later phases or used to check the generated data.

The input tables are checked to make sure all required columns are available. Default values are applied to empty cells if needed. Macro definitions and *Perl* match operators are preprocessed and stored in separate intermediate hash arrays to be applied later on.

The signal descriptions from the EXCEL tables are stored in the hash array `%conndb` modeling the non-hierarchical connection matrix CONN. The signal names are used as hash keys. Each array element itself is another hash array addressing the input data columns. Thus a signal and its properties can be accessed easily by its name. A unique, flat name space for signals is guaranteed. The signal `SIG_R` from the example in Figure 2 is stored as shown in Figure 9.

```
$conndb{'SIG_R'} = {
'::bundle' => 'RGB', '::class' => 'Data',
'::clock' => 'CLKF', '::type' => 'logic'
'::in' => [{'inst'=>'MODC', 'port'=>'R', … ],
'::out' => [{'inst'=>'MODA', 'port' => 'R', … ],
'::descr' => 'single bus', …};
```

Figure 9: Signal Data Structure

Macro expansion helps to specify regular or repetitive connections like standard on-chip buses very efficiently. The CONN data structure gets extended when additional information for a signal is defined in later stages.

For the instance specification a similar approach is taken. A HIER data structure keeps track of the hierarchy information by using tree objects. The instances make up a hierarchical view of the design data. The top node of the tree equals the design top cell. Additional instances and connections are added according to the specification in the IO.xls and Regs.xls input files.

Various checks verify the consistency of the design hierarchy and connection matrix, e.g. detect instances not connected to the main hierarchy or drivers and loads. The check routines print out warning

messages and flag the parts that conflict with the design guidelines. Based on these comments refinements to the input files should be applied by the designer. A rerun of the generator quickly shows the effect of the changes. A table-based approach for protocol specification is proposed in [5]. Even if the applications are different, both table-based approaches share the same benefits. Tables are relatively easy to read by both humans and machines and "easy" errors in tables can be caught by checks based on Boolean rules.

Ports are added to the hierarchical blocks for signals crossing hierarchies. The resulting HIER and CONN data structures represent the top levels of the chip design. The data is written to disk in an internal data format, ready to be read in by the step 2. On demand an EXCEL workbook can be generated for review and for documentation purposes.

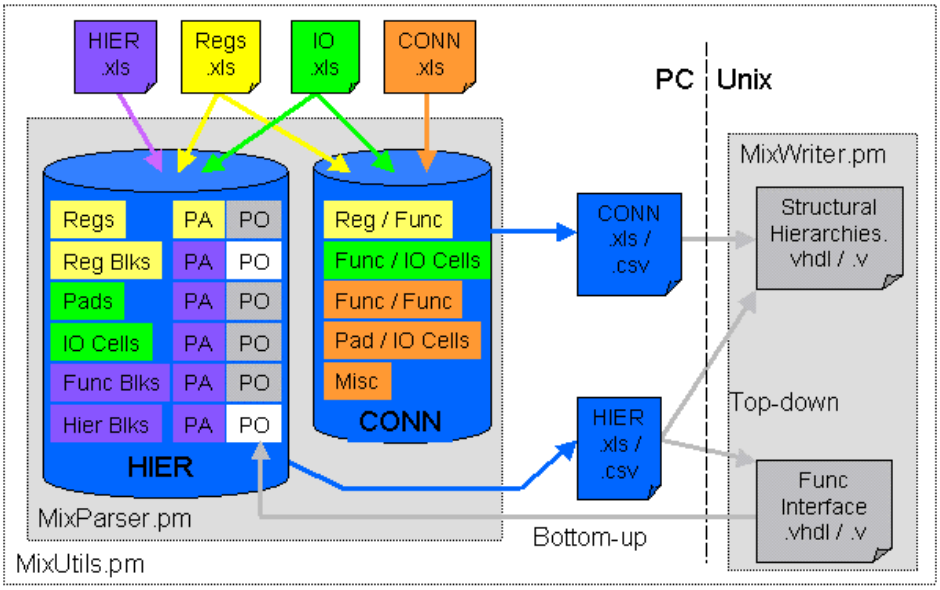


Figure 8: Dataflow from Specification to Design

4.3 Step 2: Generation of HDL Files

After transfer of the dumped intermediate data to a UNIX environment, the HDL files are generated. In case of VHDL output the primary design units Entity, Architecture and Configuration are derived and written. Additionally, HDL dependent extensions or wrappers and tool specific extensions are added and written into files.

4.4 Cross-Platform Implementation Summary

The base module MixUtils.pm provides functions for reading and writing files in various formats and for the tool configuration environment. It hides all operating system details from the upper levels, see Figure 10. MixParser.pm and MixWriter.pm correspond to step 1 and 2. Simple scripts use the functions of these modules, hiding all implementation details and data structures from the user.

5 APPLICATION RESULTS

The proposed methodology and the prototype tool was successfully applied to a display processor and scaler for LCD TV applications [6]. The design was modeled using 60 leaf modules at 7 levels of hierarchy connected with 500 explicit signals, about 250 generated register instances with 500 parameters resulting into the same number of generated implicit signals. Further, the design consisted of 90 digital pads with 4 functional and 2 test multiplex options each, resulting in around 1000 generated connections between pads IO cells and core signals. Out of 4 specification tables more than 25000 lines of VHDL code in 78 files were generated.

6 CONCLUSION

The fact than only a quarter of all connections were specified explicitly and that the rest was generated automatically, demonstrates the efficiency of the approach. But even more important than the productivity increase is the single source specification principle that guarantees consistency between all generated parts: HDL code, low-level driver software and table-based documentation.

REFERENCES

1. Sandler, S.: Expanding Debug Technology; Electronic Engineering Design - UK, Nov. 2002; http://www.electronicengineering.com/story/OEG20021105S000.
2. Edwards, C.: Mentor extends HDL portfolio with interface design; EE Times UK; 2001; http://www.electronicstimes.com/story/OEG20010309S0027.
3. Rowson, J.; Sangiovanni-Vincentelli, A.: Interface-Based Design; Design Automation Conference, 1997.
4. Schneider, C.; Ecker, W.: Stepwise Refinement of behavioral VHDL Specifications by Separation of Synchronization and Functionality; EURO-DAC with EURO-VHDL, 1996.
5. Mani Azimi, Ching-Tsun Chou, Akhilesh Kumar, Victor W. Lee, Phanindra K., Mannava, and Seungjoon Park, "Experience with Applying Formal Methods to Protocol Specification and System Architecture", Formal Methods in System Design, Vol. 22, pp. 109-116, 2003.
6. Hahn, M.; Schu, M.; Keller, S.; Schneider, C.; Carpentier, D.: System-On-Silicon for Liquid Crystal Displays; IEEE International Symposium on Consumer Electronics, ISCE, 2002.