



# Data Management for a Mixed-Signal Design Project with Distributed Teams

Himadri De,  
Scott Humphreys,  
William Farlow,  
Matt Deig,  
Tammy Glascock

# Outline

- Introduction
- Historical Perspective
- Requirements
- Server Architecture
- Project Configuration
- Workarea
- Check-in/out in Virtuoso
- Tagging
- Introducing DM in Mixed-Signal Design
- Future Work/Conclusion

## Introduction

- Data management of mixed-signal SoC with distributed teams is a challenge
- Tools exist for distributed SW projects
- Pure ASIC designs can adopt SW methods
- Mixed-Signal design uses various other EDA tools and SW methods wouldn't work
- SOS from Clisoft was used as DM tool

## Overview

- Involves geographically distributed teams with project lasting over several years
- Project goes through different design phases with more designers joining the project
- EDA tool skill level/understanding of DM concepts varies amongst designers
- Revision history is critical
  - Historical reason
  - Allow different sub-system groups to work at different maturity levels of the design

## Historical Perspective

- Each designer works in his/her own library and must continually update their cds.lib to gain access to co-worker's libraries
- Rsync and tar/untar make the design process unproductive
- Creative naming conventions are used to distinguish different stages of a design

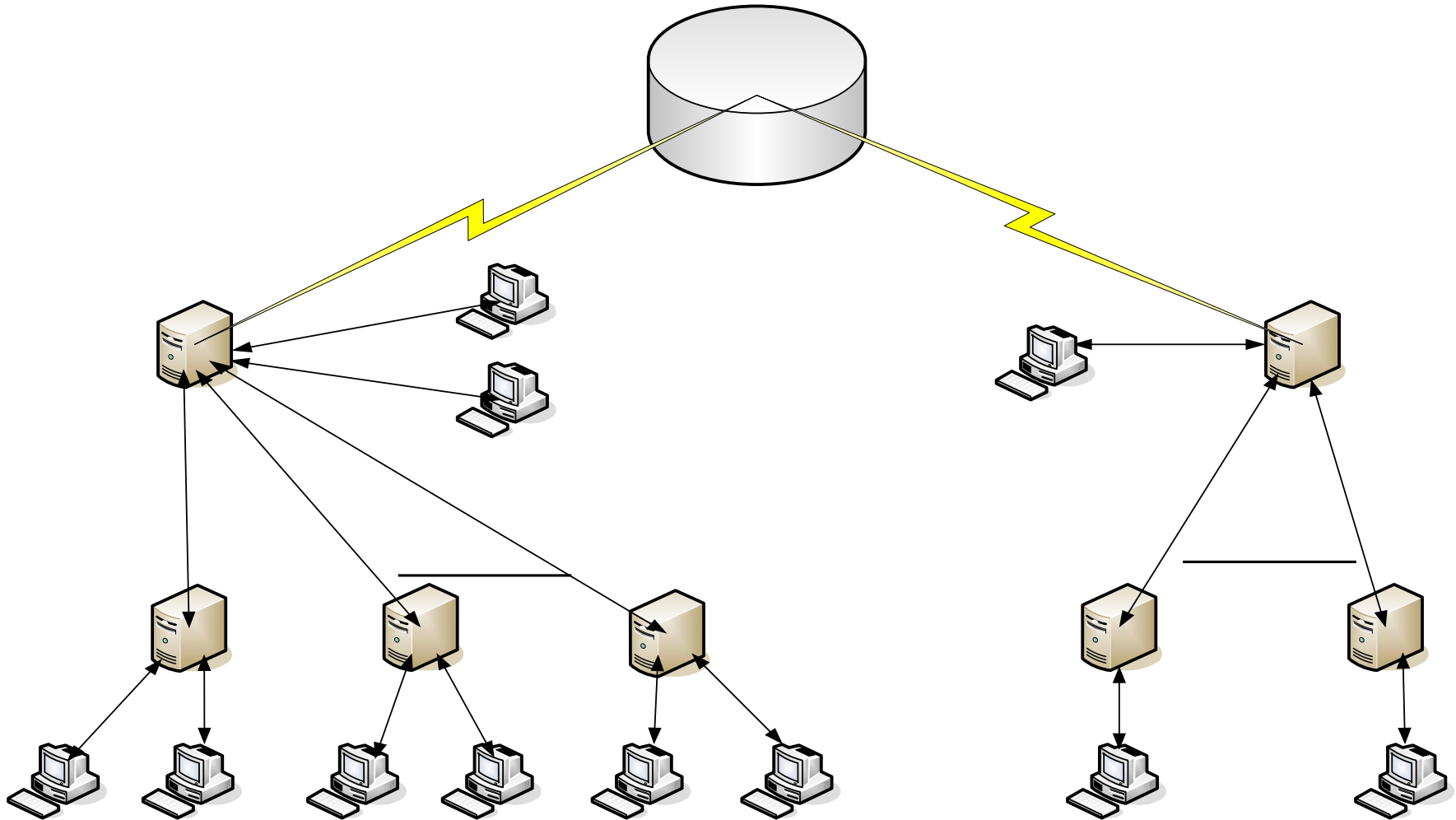
## Requirements of a DM Infrastructure

- Designers should be able to efficiently access design data
- Hardware requirements should conform to company's existing hardware platform
- Designers should be able to perform DM operations from within the design tools and at the same abstraction level
- DM tool should not modify EDA binaries

## Data under Revision Control

- Both RF/analog and digital data should be under same DM environment
- Rev control Cadence design & test fixture libraries, RTL, testbenches, Makefiles, stimulus vectors & environment settings
- Synthesis .db or gds from Encounter should be under rev control only at critical points
- Sim data should not be under rev control

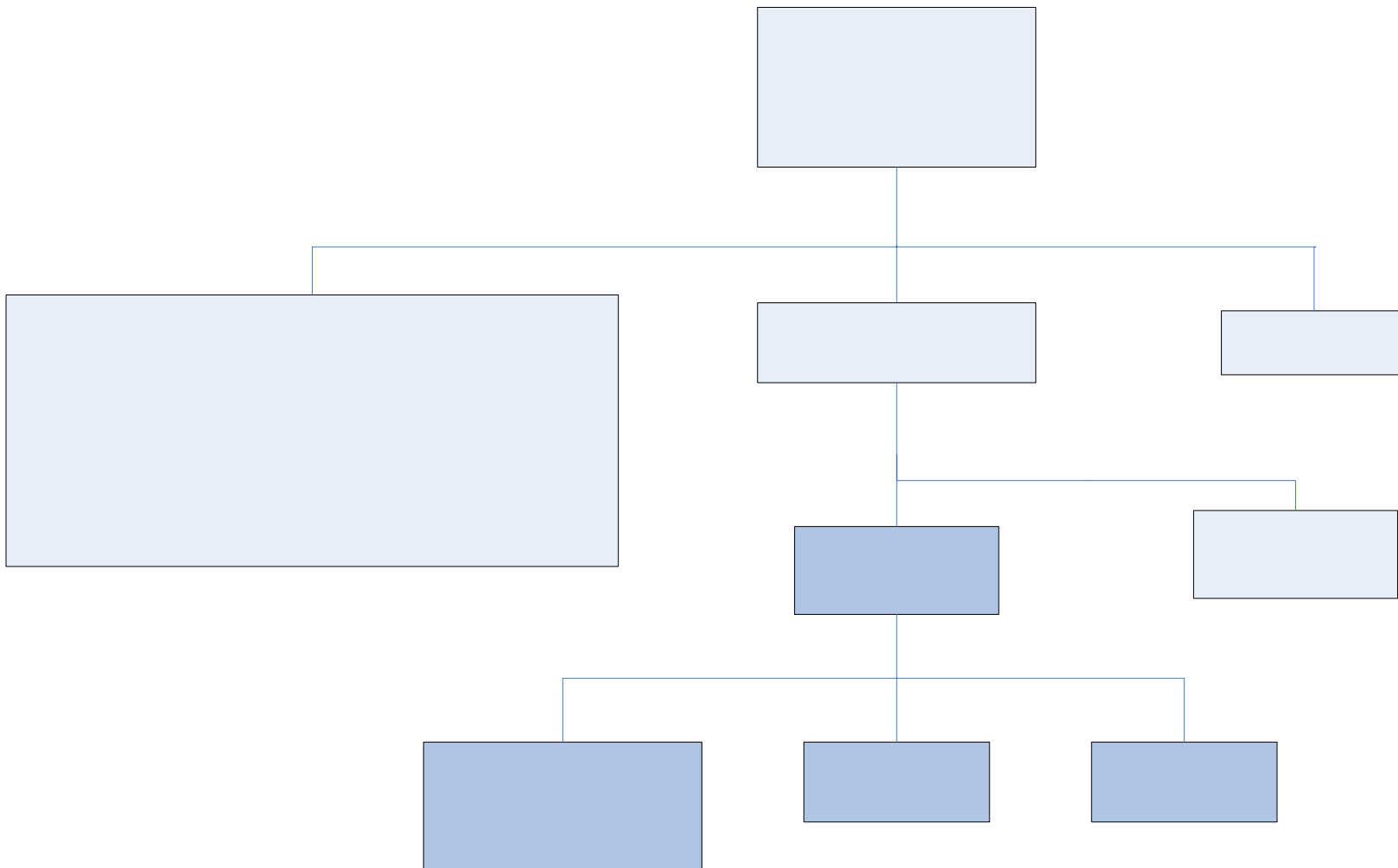
# Server Architecture



## Project Configuration

- A design project is partitioned into cds\_master, doc, and work\_libs directories
- Important for every designer to have the same design environment – same tool/library version
- Path to tool/library version hard-coded in the setup files to allow independent control of environment for each project

# Project Directory Structure



## Workarea

- For a given project a designer works in his/her workarea
- Workarea is a local directory structure that mirrors the directory hierarchy of the project and contains specific rev of files as specified by the Revision Search Order
- The workarea model used at RFMD is one in which files in the workarea are symbolic links to smart cache => better utilization of disk space

## Workarea Model

- When one checks out a file for editing, the symbolic link is deleted and a writeable physical copy of the file is placed in the workarea
- When one checks in the file, the file in the workarea is replaced by a link to the new revision of the file

## Workarea Setup Script

- An automated perl script initially sets up the workarea for a user
- The script has the option to set up multiple workarea for a single user. Usually designers involved in verification tend to utilize multiple workareas

## Customization

- `.cdsinit` and `.cdsenv` rev controlled
- User customization discouraged
- User customization like bind-key definitions allowed through `cdsUsr` file in user's workarea – invoked within `.cdsinit`
- `cds.lib` not rev controlled to allow definition of libraries from Lib Manager – master `project.lib` included within `cds.lib`

## Dummy Workarea

- Dummy workareas with “local copies” instead of “links to cache” are setup at each remote location
- Cron jobs periodically perform an update in this area at each site
- This minimizes the chance that any given user would be the first to update across the network and it provides a periodic back-up of the latest rev of the project. The server itself is backed up on a filer machine nightly

## Checking-in/Out of Cadence Cells

- **A Cadence cell is a Unix directory containing a sub-directory for each view, and possibly a cell level properties file called prop.xx.**
- **ClioSoft and Cadence generic data management (GDM) interface understand that when one works with a view, one has to check-in and check-out the view directory and its files together as a “co-managed set.”**

## Problems with prop.xx

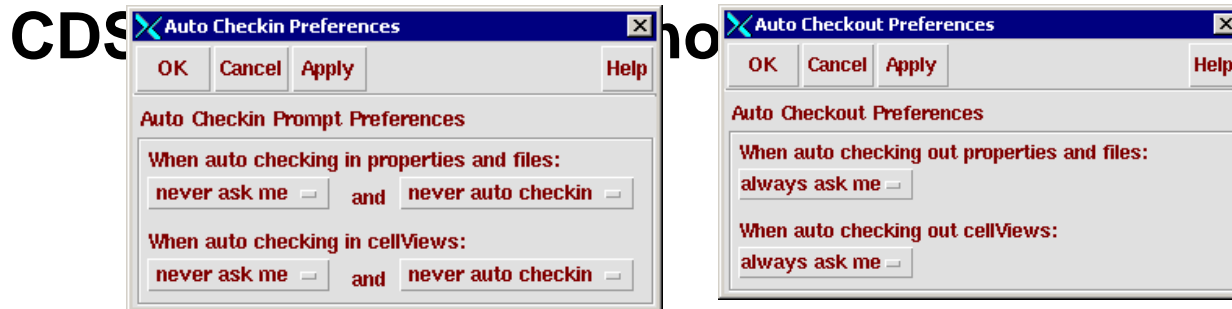
- Cell level prop.xx file does not always get checked-in whenever a symbol, schematic, or model view is checked in, even though it should and that has created problems if one checks in from the view level rather than the cell level in the Library Manager.

## Auto Check-Out Preferences

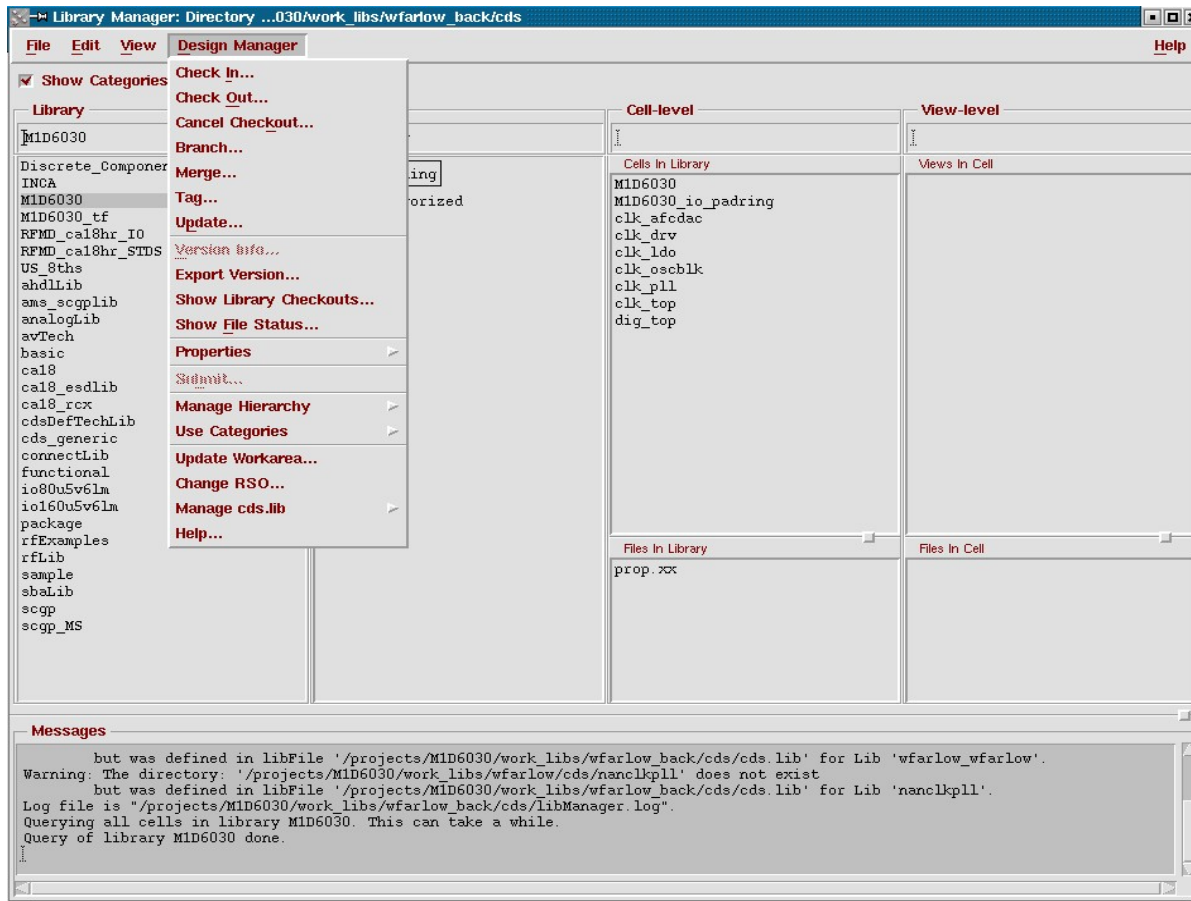
- To avoid problems with unknowingly checking out cells or properties, the following variables are defined

**CDS\_AUTO\_CKOUT=none**

**CDS\_PRIOMPT\_CKOUT=all CDS\_AUTO\_CKIN=none**



# Design Manager Menu



## Tagging

- Generally we want to keep the latest revisions of everything “clean.” However, there are cases where this is not possible:
  - One has work-in-progress that he needs to share with someone on the team, but the cell views will not work properly. In this case it is still preferable to check in rather than “go unmanaged” by copying cells into unmanaged scratch libraries.
  - Your cells work fine by themselves, but might not play well with others in the next level of hierarchy above.
- For these reasons we depend on tagging of the cells to help ensure we get “good” cell revisions in our work areas when updating.

## Tags

- A tag is a label applied to a specific revision of a file or directory or cell in the revision control system's "repository"
- We use tags to indicate both the "maturity" of a cell, and to note which revisions of different cells are "consistent" with each other

# Cell Level Maturity Tags

- **model\_works**  
The behavioral model has basic functionality that can be used to debug test benches and illustrate basic behavior. Apply this tag to symbol and model views (verilog, veriloga, verilogams), and also to schematic and config views that instantiate cells that are set to model views.
- **model\_verified**  
The behavioral model has all the required functionality and meets the performance requirements for parameters that are modeled. Apply this tag to symbol and model views (verilog, veriloga, verilogams), and also to schematic and config views that instantiate cells that are set to model views.
- **sch\_works**  
The schematic has basic functionality that can be used to debug test benches and illustrate basic behavior. This tag is applied to schematics and symbols used for transistor level simulations.
- **sch\_verified**  
The schematic meets all requirements, but may change after layout is near completion. This tag is applied to schematics and symbols used for transistor level simulations.
- **layout\_works**  
The layout may be used for floor planning purposes. This tag is applied to the corresponding schematic, if one was used for estimating device areas.
- **layout\_verified**  
The layout is DRC and LVS clean with the corresponding schematic. This tag is also applied to the corresponding schematic.
- **design\_verified**  
The model, schematic and layout meet all requirements and are consistent with each other.

# Maturity Tags

The following table shows which tags will be applied to which views  
 model views are: verilog, veriloga, verilogams, empty

purpose	tag	view				
		config	model	schematic	symbol	layout
model simulations	model_works	x	<b>X</b>	x	x	
	model_verified	x	<b>X</b>	x	x	
transistor simulations	sch_works	x		<b>X</b>	x	
	sch_verified	x		<b>X</b>	x	
layout design	layout_works			<b>X</b>		<b>X</b>
	layout_verified			x		<b>X</b>
final verifications and tapeout	design_verified		x	x	x	x

## Maturity Tags Usage

- Depending on the type of work one is doing, only one type of tag is used when tagging the cells, or in using tags in one's RSO as shown below:
- transistor simulations:
  - Apply tag sch\_works or sch\_verified to schematic and symbol views
  - set RSO to “sch\_works, main” or “sch\_verified, main”
- model simulations:
  - Apply tag model\_works or model\_verified to schematic, symbol, and model views used in the sim (for AMS, use the script sim/sos\_config\_tag.pl)
  - set RSO to “model\_works, main” or “model\_verified, main”
- layout work:
  - Apply tag layout\_works or layout\_verified to schematic and layout views
  - set RSO to “layout\_works, main” or “layout\_verified, main”
- tapeout and final verifications:
  - Apply tag design\_verified to schematic, symbol, layout, and model views
  - set RSO to design\_verified

## TestFixture/Conf Tags

- We have recognized the need to apply tags that correspond to different testfixtures that we use for verification, as the cell level maturity tags allow users to retrieve inconsistent sets of hierarchical cells. Consider the following example.
  - A sub-system lead runs a verification on the entire PLL and tags everything under the PLL “model\_works” on Monday.
  - On Tuesday, the charge-pump designer makes a change to the symbol ports and model, verifies them in his block-level test bench, and tags them “model\_works.”
  - On Wednesday the Tx sub-system lead updates to “model\_works” and gets a PLL schematic that incorrectly instantiates the new charge-pump symbol.
- By applying tags to all cell/view revs used in a good simulation or verification run we can avoid this problem. We are calling these “testfixture/config” tags to distinguish them from the cell maturity tags

## Do's and Don'ts

- Do not include/define another user's revision controlled library in your Library Manager path
- Update your workarea frequently
- Minimize checkouts and the length of time you have files checked out
- Read popup dialogs carefully; some are different for revision controlled libraries
- Look at your checkouts couple of times a day

## Introducing Data Management to a Mixed-Signal Design Team

- Designers need to be made aware of the benefits and they should buy-in to the methodology
- Training on Rev Control Concepts and DM Tool
- Automate setup environment with scripts
- Small pilot team to work out requirements & methodology & test it on a real test project

## Future Enhancements

- Implement atomic check-in
- Better utilize access control features
- Better utilize branching to manage different tapeout versions of the same product

## Conclusion

- Without DM and project configuration it is inefficient to execute large mixed-signal projects
- DM infrastructure, properly defined and introduced to the design team significantly improves productivity and leads to cycle time reduction and increases likelihood of 1<sup>st</sup> pass silicon success