methodics™

Product Lifecycle Management in a Semiconductor Design Environment - Problems and Challenges

Table of Contents

1. Product Lifecycle Management – An Overview

Product Lifecycle Management (PLM) was first introduced by automobile and defense companies in the mid 1980's with the goal of reducing development cost and speeding up the product development process. This is done by integrating the different phases of a product's lifecycle into a single system and tracking dependencies and state in a connected manner. Engineering design tools, documentation, supply chain tracking, requirements management, Bill of Materials (BOM), and other product lifecycle elements are all maintained in a single system in an attempt to keep the product development coordinated and efficient.

2. PLM for Semiconductor Design – Adoption Challenges

Although PLM tools have seen some success in industries such as defense, automobile, aerospace and others with large design teams and well established methodologies, the adoption rate in the semiconductor space has been slow. This can be somewhat attributed to the following:

2.1 PLM solutions work best in a static design environment

PLM's have been traditionally deployed on projects with well defined and unchanging design processes. Unfortunately, change is inevitable in a semiconductor design flow.. Each new process technology brings a new set of physical/electrical challenges that require adjustments and possibly new tooling. The doubling of component counts every 2 years (Moore's Law) and the ever-growing scale of Systems on Chip (SoCs) is very disruptive to design methodologies and by association, the host PLM based design environment.

2.2 Total Cost of Ownership (TCO)

A major barrier to PLM adoption is the amount of **customization and integration** required to integrate the PLM tool into an IC design environment. Multiple person-year consulting engagements are common, and these deployments generate large amounts of custom code that must be tested and maintained. Any changes in tooling and methodology can be extremely disruptive to this custom code code which will result in long iterations and possibly compromises.

Another issue with large-scale custom integrations as part of a deployment is that this software is often written by use-model experts rather than professional software developers. This can result in a poorly documented, difficult to test and **unmaintainable** code-base. The best solution is when there are minimal changes to the core platform but, unfortunately, this is usually not possible with a typical PLM system *(CIM Data - "Improving the Enterprise's Return on its PLM Investment, 2010").*

A common complaint with PLM based design methodologies is the **large downtime** experienced with system upgrades. This is due to the large number of "moving parts" in the system and the resulting dependencies between these. As part of an upgrade, all of these dependencies must be tested for breakage. Custom integrations can be particularly troublesome from a system integrity perspective and, when not written correctly, can be very difficult to test.

Another issue with PLM systems is the **high cost of maintenance**. This will typically require multiple full time resources to administer the system, fix bugs with the custom integrations, and plan and maintain consistency across the platform to accommodate new requirements and changes to the design flow.

All of the above will result in a higher than expected **Total Cost of Ownership** (TCO) where the ongoing costs of maintaining the system will usually dwarf the initial licensing outlay.

2.3 PLM Systems are Project-Centric, not IP-Centric

The set of problems outlined above are difficult to solve and endemic in the way PLM systems are architected today, but arguably more important is that today's PLM systems are not **"IP-Centric"**. In today's SoCs, we are not building a single product from the ground up. Rather, the emphasis is on attempting to reuse existing IP from previous projects or leverage off-the-shelf IP from IP vendors. This IP-centric design flow that enables a significant level of IP reuse has been identified as a crucial tool for the semiconductor industry to maintain profitability in today's business environment.

To manage an IP-centric project we need to manage the lifecycle of these component IPs and consolidate the metrics of each IP as part of the parent project IP. We call this **"IP Lifecycle Management" or "IPLM"** and is the basis for the Methodics product **ProjectIC™**.

3. What is IP Lifecycle Management?

IPLM is the management of a collection of IP's, each with their own release schedule and independent set of issues and dependencies, and consolidating these into a single top level parent IP. During this consolidation, the metrics associated with each of the IP's in the hierarchy must be reconciled at the top level IP.

There are a number of basic tenets in an IPLM system:

3.1 The IPLM tool must understand IP as a base object

In an IP-centric design flow, we will be building our designs, creating user workspaces, managing quality, etc. at the IP level rather than at the file or project level. Fig.1 summarizes all the elements necessary for tracking and managing at the IP level.

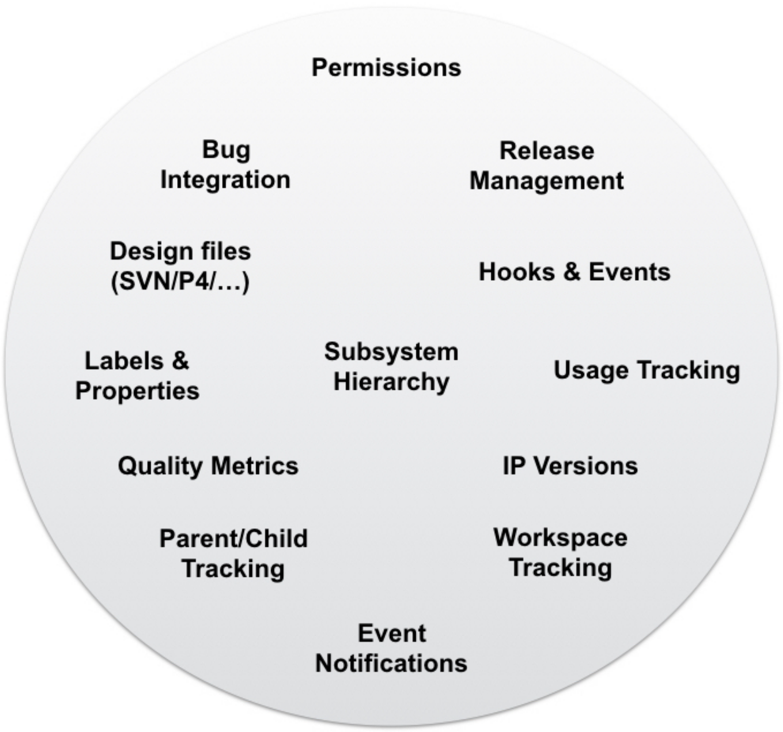


Fig. 1: The Base IP Object

Some of the important fields are:

Permissions: IP Permissions allow which users or groups can view the IP in the IP Catalog, create workspaces, and make changes to the IP (releases).

Design files / SCM Integration: Users must be able to extract the relevant file versions from an SCM system and create user workspaces. Common SCM systems in use today include Perforce, Subversion, and Git.

Usage Tracking: The system must be able track where an IP (version) is being used across all the projects in the company.

Labels & Properties: The system must support the idea of labels for IP cataloging/search and properties for extending the basic IP fields to include searchable customer specific fields.

Event Notifications: The system must include an “events and triggers” overlay to support notifications of important IP releases (filtered by quality perhaps), continuous integration after a release, etc.

Subsystem Hierarchy: The system must be fully hierarchical and support files and subsystems at each level of a design.

Workspace Tracking: The system must track user workspaces and allow audits of user activity.

Parent/Child Tracking: The system must track the parent/child relationship of every IP version and variant across all design projects.

3.2 The system must support hierarchical IP configurations

In an IPLM system, the design configuration will include a top-level project IP with hierarchical references to other IP's in the design ecosystem. In this IP-centric methodology, each IP in the hierarchy can be considered as a standalone entity that perhaps worked either in isolation or as part of a parent project.

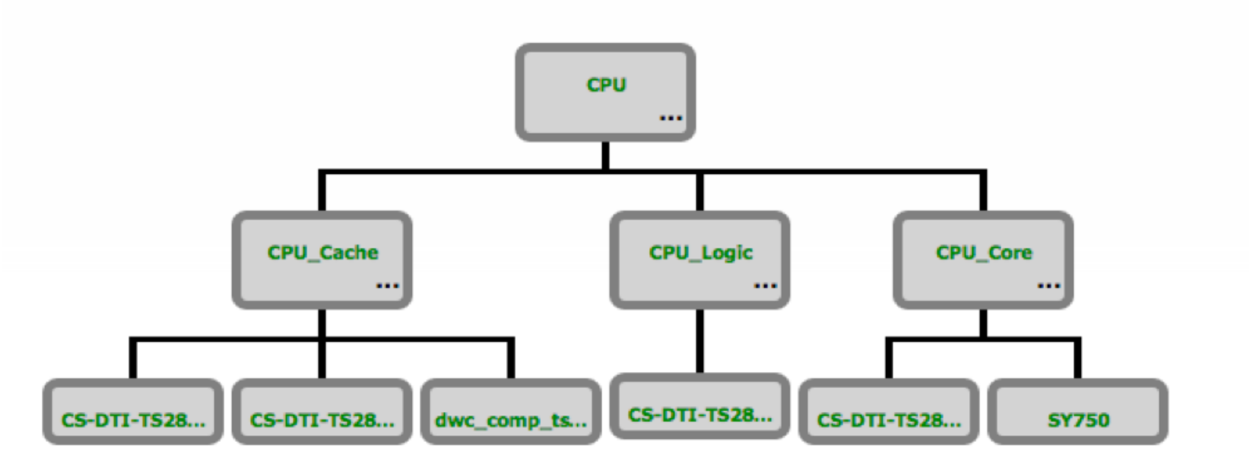


Fig. 2: Example hierarchy in a CPU subsystem.

Characteristics of hierarchical IPs include:

Abstraction: It is quite easy to integrate self-contained functional blocks into your design rather than a laundry list of disconnected individual IPs that are handled outside of the platform. This is much quicker and more intuitive.

Compatibility: Does version 10 of the USB controller really work with version 7 of the PHY? Should I use the latest version of both these blocks and hope for the best? These kinds of questions can be eliminated by simply choosing USB subsystems that have proven configurations with versions that are verified to work with each other.

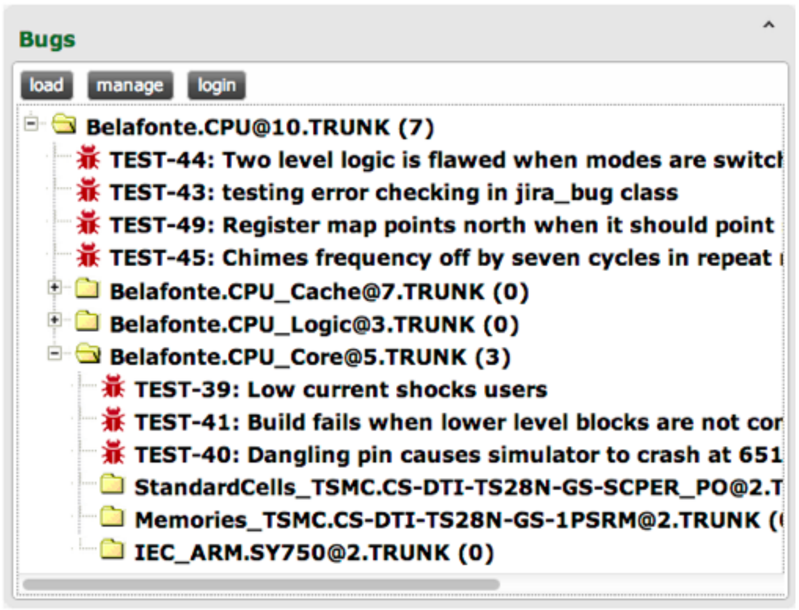
Dependency Management: Bringing in a subsystem automatically brings in all the dependencies that are needed. There is no need for cumbersome dependency discovery, since the subsystem will bring in all the needed dependencies.

Discovery: By looking at the hierarchies in which a particular IP of interest is used, the IPLM system can help with easier discovery of the various components available to the team. For example, if the USB PHY is used in a hierarchy that contains other IO interface blocks, it is very useful to be able to discover the context in which this block is frequently used. This can aid the design process immensely.

3.3 The IPLM platform must inherit metadata hierarchically

IPs will evolve according to their own lifecycle, and important metadata will be available from a number of different contexts depending on where that IP is being used. If an IP is instantiated in 3 or 4 different projects, then different bugs may get revealed, different kinds of testing will be seen, power consumption and other important metrics may be measured.

In Methodics' solution, this metadata is maintained on the ProjectIC Web dashboard on a per-IP version basis, and is based on the set of versions used consolidated across the SoC project hierarchy, to provide a dashboard entry for that top level IP. Given the mandates for IP reuse in today's semiconductor industry, every standalone project should be designed in a way that lends itself to IP reuse. As the cliché goes, "Today's project is tomorrow's IP".



Belafonte.CPU@10.TRUNK (7)
- TEST-44: Two level logic is flawed when modes are switch
- TEST-43: testing error checking in jira_bug class
- TEST-49: Register map points north when it should point
- TEST-45: Chimes frequency off by seven cycles in repeat
- Belafonte.CPU_Cache@7.TRUNK (0)
- Belafonte.CPU_Logic@3.TRUNK (0)
- Belafonte.CPU_Core@5.TRUNK (3)
  - TEST-39: Low current shocks users
  - TEST-41: Build fails when lower level blocks are not cor
  - TEST-40: Dangling pin causes simulator to crash at 651
  - StandardCells_TSMC.CS-DTI-TS28N-GS-SCPER_PO@2.T
  - Memories_TSMC.CS-DTI-TS28N-GS-1PSRM@2.TRUNK (
  - IEC_ARM.SY750@2.TRUNK (0)

Fig. 3: Hierarchical "IP-Centric" Bug Tracking in ProjectIC.

As an example, in Fig. 3 we can see how bugs across each IP version in the hierarchical configuration are consolidated and rolled up into the parent IP. The total number of bugs found within the project is the cumulative bugs found through the hierarchy.

4. Reducing the TCO – Embracing the Customer Workflow

We've seen that the Total Cost of Ownership for a PLM tool can be prohibitively high in a semiconductor design environment, but how can an IPLM tool reduce that?

ProjectIC has a handful of "integration points" to the customer design environment. The goal is to reduce the burden on the user to maintain these touch points, and do this in a robust way that will survive change and, at the same time, support the concept of IP reuse.

4.1 IP Definitions (property sets and labels)

The first integration point a customer will need to supply is the IP definition, and specifically which IP metadata they choose to maintain with the IP. In ProjectIC, this is organized using "Property Sets" which allow a collection of metadata fields to be applied in one operation. While the onus is on the customer to agree on how to catalog the IP in ProjectIC, to determine the taxonomy to use, etc., once it is done, it can all be formalized into easily maintained property sets in ProjectIC. Once these are in place, small tweaks to the Property Set definition can be made to effect change across the entire IP catalog for easy maintenance.

4.2 Workflow: Workspace Management and IP Release

Once the customer's existing IP information has been imported into ProjectIC, development work can begin. ProjectIC provides some basic functionality to create/update a user workspace and to release the changes to an IP from a user workspace.

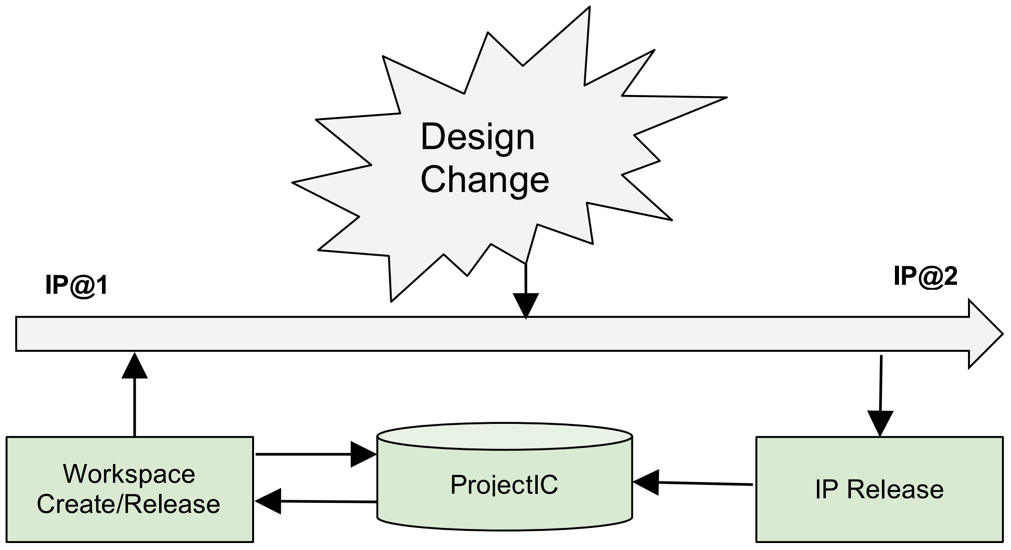
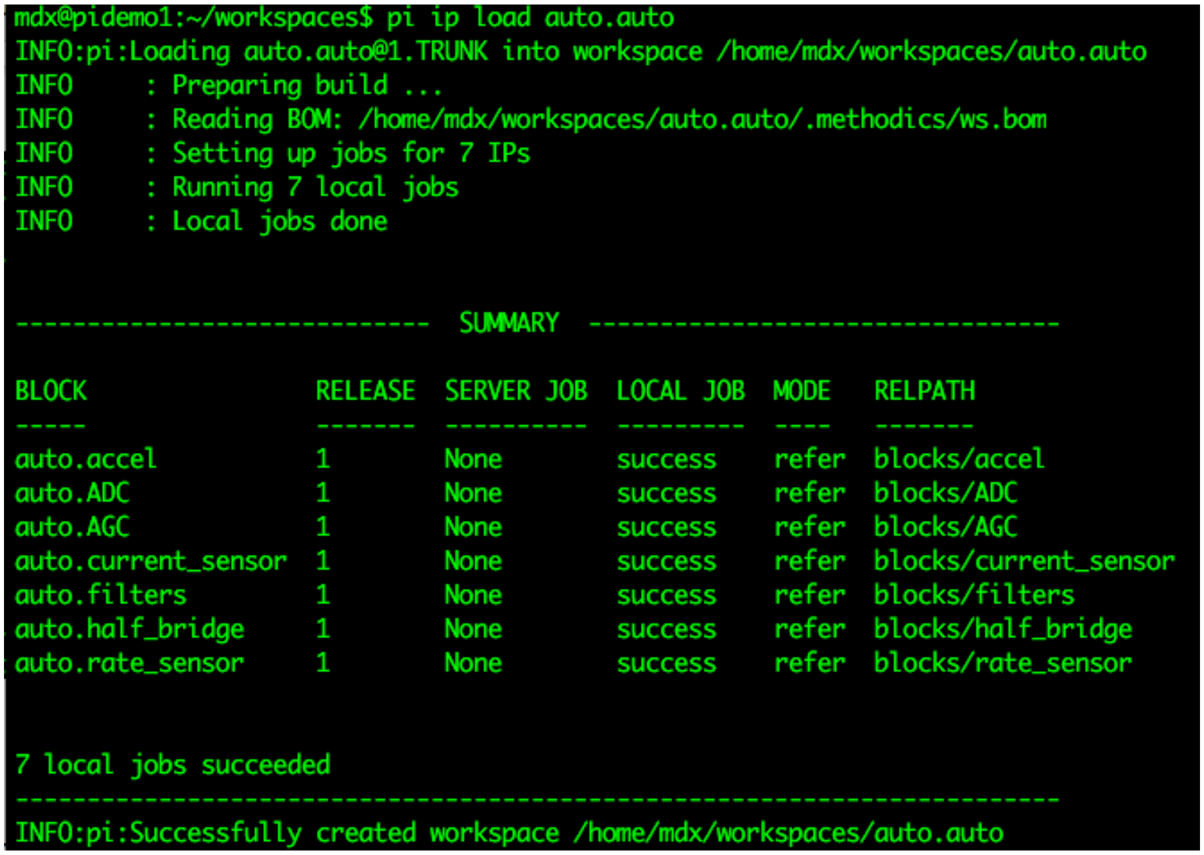


Fig. 4: "Workspace Create" and "IP Release" in ProjectIC

These workspace and release functions are required to enable the design methodology but do not add any overhead to the workflow. They do, however, give ProjectIC an opportunity to collect metadata such as simulation/verification results, place and route status, power/performance metrics, and other useful information that can be maintained with the IP release for review in the ProjectIC dashboards.

4.2.1 Workspace Create

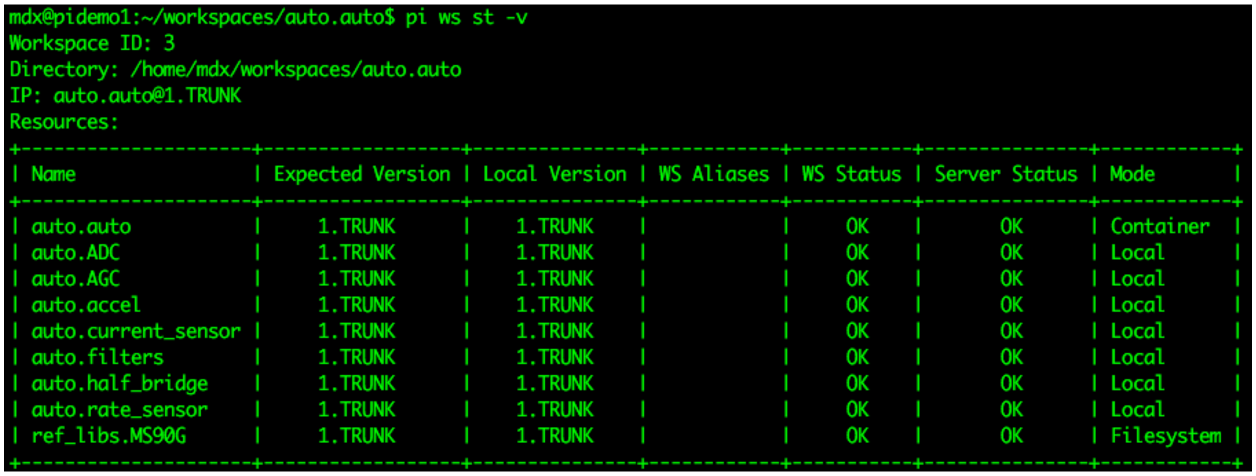
Once the customer IP data has been imported into the system, a project configuration can be created as a hierarchical IP, and we may build a user workspace leveraging the data management integration (Perforce/Git/Subversion) in ProjectIC. This is an important step because, as well as providing a key function for the user, it establishes a connection between the workspace (users design data) and the ProjectIC database. Figure 5 shows how we load the files associated with an IP (including the files associated with the subsystems) into a workspace.



```
mdx@pidemo1:~/workspaces$ pi ip load auto.auto
INFO:pi:Loading auto.auto@1.TRUNK into workspace /home/mdx/workspaces/auto.auto
INFO     : Preparing build ...
INFO     : Reading BOM: /home/mdx/workspaces/auto.auto/.methodics/ws.bom
INFO     : Setting up jobs for 7 IPs
INFO     : Running 7 local jobs
INFO     : Local jobs done


----------------------------  SUMMARY  --------------------------------

BLOCK                RELEASE  SERVER JOB  LOCAL JOB  MODE   RELPATH
-----                -------  ----------  ---------  ----   -------
auto.accel           1        None        success    refer  blocks/accel
auto.ADC             1        None        success    refer  blocks/ADC
auto.AGC             1        None        success    refer  blocks/AGC
auto.current_sensor  1        None        success    refer  blocks/current_sensor
auto.filters         1        None        success    refer  blocks/filters
auto.half_bridge     1        None        success    refer  blocks/half_bridge
auto.rate_sensor     1        None        success    refer  blocks/rate_sensor


7 local jobs succeeded
-----------------------------------------------------------------------
INFO:pi:Successfully created workspace /home/mdx/workspaces/auto.auto
```

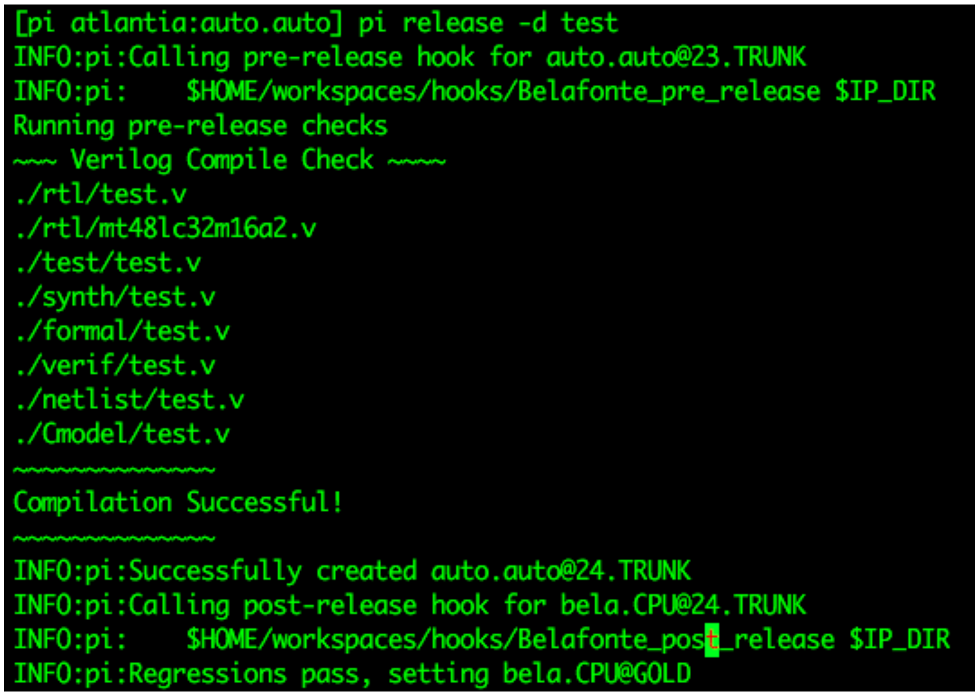
Fig. 5: Loading an IP into a workspace

Figure 6 shows the ProjectIC workspace status output. This report tells the user what changes are in the workspace compared to the published release file versions, and any local modifications not yet committed into the data management system.


```
mdx@pidemo1:~/workspaces/auto.auto$ pi ws st -v
Workspace ID: 3
Directory: /home/mdx/workspaces/auto.auto
IP: auto.auto@1.TRUNK
Resources:
+---------------------+------------------+---------------+------------+-----------+---------------+-----------+
| Name                | Expected Version | Local Version | WS Aliases | WS Status | Server Status | Mode      |
+---------------------+------------------+---------------+------------+-----------+---------------+-----------+
| auto.auto           |     1.TRUNK      |    1.TRUNK    |            |    OK     |      OK       | Container |
| auto.ADC            |     1.TRUNK      |    1.TRUNK    |            |    OK     |      OK       | Local     |
| auto.AGC            |     1.TRUNK      |    1.TRUNK    |            |    OK     |      OK       | Local     |
| auto.accel          |     1.TRUNK      |    1.TRUNK    |            |    OK     |      OK       | Local     |
| auto.current_sensor |     1.TRUNK      |    1.TRUNK    |            |    OK     |      OK       | Local     |
| auto.filters        |     1.TRUNK      |    1.TRUNK    |            |    OK     |      OK       | Local     |
| auto.half_bridge    |     1.TRUNK      |    1.TRUNK    |            |    OK     |      OK       | Local     |
| auto.rate_sensor    |     1.TRUNK      |    1.TRUNK    |            |    OK     |      OK       | Local     |
| ref_libs.MS90G      |     1.TRUNK      |    1.TRUNK    |            |    OK     |      OK       | Filesystem|
+---------------------+------------------+---------------+------------+-----------+---------------+-----------+
```
Fig. 6: Reporting the workspace status

4.2.2 IP Release

Once a user has created a workspace, they are ready to start making changes using their standard tool flow to the point where they can make a release for consumption by other members of the organization. To enable this, ProjectIC provides a release command that records the file versions in the workspace, including hierarchical any IP versions used as resources.


```
[pi atlantia:auto.auto] pi release -d test
INFO:pi:Calling pre-release hook for auto.auto@23.TRUNK
INFO:pi:    $HOME/workspaces/hooks/Belafonte_pre_release $IP_DIR
Running pre-release checks
~~~ Verilog Compile Check ~~~~
./rtl/test.v
./rtl/mt48lc32m16a2.v
./test/test.v
./synth/test.v
./formal/test.v
./verif/test.v
./netlist/test.v
./Cmodel/test.v
~~~~~~~~~~~~~~~~
Compilation Successful!
~~~~~~~~~~~~~~~~
INFO:pi:Successfully created auto.auto@24.TRUNK
INFO:pi:Calling post-release hook for bela.CPU@24.TRUNK
INFO:pi:    $HOME/workspaces/hooks/Belafonte_post_release $IP_DIR
INFO:pi:Regressions pass, setting bela.CPU@GOLD
```
Fig. 7: Creating an IP release

In this example, we can see that the release called the pre-release and post-release hooks in ProjectIC. These hooks trigger scripts to check that the IP's basic smoke tests pass in the pre-release case (Verilog compilation checks in this example) and a full regression run in the post-release case. As a result of the post-release hook, the customer-defined ProjectIC alias “GOLD” was set on the IP, and users were notified that a new qualified release was available.

4.3 Connecting IP Creators to IP Consumers

By providing workspace management functions, we can track workspaces and the contents of those workspaces in ProjectIC. This means we can track where IP is being used across the company, and in what context. We can track commonly occurring clusters of IPs and their versions, and use that information to identify subsystems across the company beyond the formally declared subsystems in the IP catalog.

When we combine this workspace management with IP release commands, then we can see that we are starting to connect the IP creators (designers) with the IP consumers (the integration team). The IP in the system evolves with releases of recorded quality, and users can search for these in the IP catalog with a minimum of overhead.

Note that we haven’t changed the user’s design flow in any way. In fact, we’ve supplemented the user environment with important commands to the design flow that will transparently gather metadata and add it to the underlying IP objects in ProjectIC.

4.4 Collecting, consolidating and displaying metadata in ProjectIC

The last integration touch point required in ProjectIC is the collection and consolidation of metadata on the web dashboard. There are 2 main sources of metadata:

4.4.1 User workspaces as a source of metadata (the ProjectIC API)

As a user works on the design, important metadata accumulates in the workspace that we’d like to associate with the next version of the IP. For example, we may want to annotate the latest power measurements, the number of passing and failing regressions, the number of EDA licenses consumed during that design iteration, whether the design met timing constraints in a place and route job, and many others.

This metadata can be collected via user-defined scripts that write properties into the ProjectIC database. To achieve this, ProjectIC includes a comprehensive API to enable integration into the customer environment. The same API calls that we use for our clients are exposed and fully documented for customer integration.

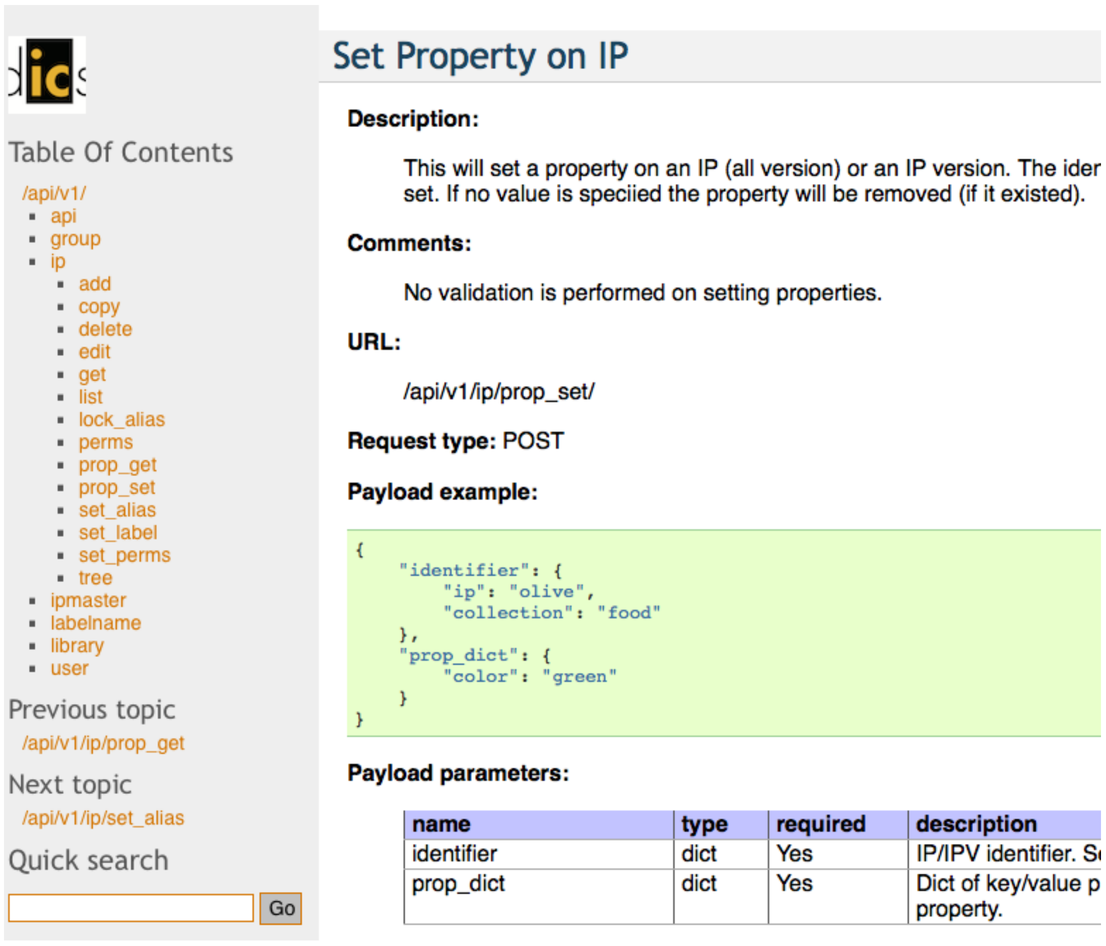


Table Of Contents

/api/v1/
- api
- group
- ip
  - add
  - copy
  - delete
  - edit
  - get
  - list
  - lock_alias
  - perms
  - prop_get
  - prop_set
  - set_alias
  - set_label
  - set_perms
  - tree
- ipmaster
- labelname
- library
- user

Previous topic

/api/v1/ip/prop_get

Next topic

/api/v1/ip/set_alias

Quick search

Go

Set Property on IP

Description:

This will set a property on an IP (all version) or an IP version. The ider
set. If no value is speciied the property will be removed (if it existed).

Comments:

No validation is performed on setting properties.

URL:

/api/v1/ip/prop_set/

Request type: POST

Payload example:

```
{
    "identifier": {
        "ip": "olive",
        "collection": "food"
    },
    "prop_dict": {
        "color": "green"
    }
}
```

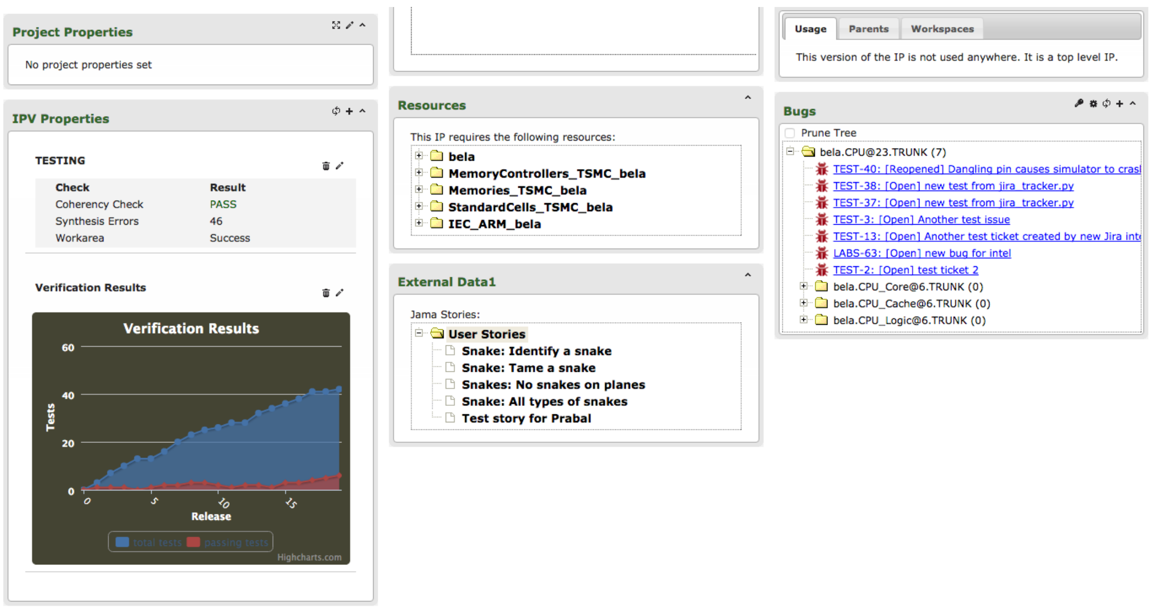
Payload parameters:

name	type	required	description
identifier	dict	Yes	IP/IPV identifier. S
prop_dict	dict	Yes	Dict of key/value p property.

Fig. 8: ProjectIC API Docs.

4.4.2 External sources of metadata

Other common sources for metadata in ProjectIC are external systems such as Requirements Management tools, regression/verification platforms, coverage tools, bug tracking interfaces, and other third-party systems. ProjectIC supports a number of dashboard widget types including tables, charts, graphs, text and others. These are populated from JSON format properties passed into ProjectIC using the API.


Project Properties

No project properties set

IPV Properties

TESTING

Check	Result
Coherency Check	PASS
Synthesis Errors	46
Workarea	Success

Verification Results

Resources

This IP requires the following resources:

- bela
- MemoryControllers_TSMC_bela
- Memories_TSMC_bela
- StandardCells_TSMC_bela
- IEC_ARM_bela

External Data1

Jama Stories:

- User Stories
  - Snake: Identify a snake
  - Snake: Tame a snake
  - Snakes: No snakes on planes
  - Snake: All types of snakes
  - Test story for Prabal

Usage | Parents | Workspaces

This version of the IP is not used anywhere. It is a top level IP.

Bugs

Prune Tree

- bela.CPU@23.TRUNK (7)
  - TEST-40: [Reopened] Dangling pin causes simulator to crasl
  - TEST-38: [Open] new test from jira_tracker.py
  - TEST-37: [Open] new test from jira_tracker.py
  - TEST-3: [Open] Another test issue
  - TEST-13: [Open] Another test ticket created by new Jira inte
  - LABS-63: [Open] new bug for intel
  - TEST-2: [Open] test ticket 2
  - bela.CPU_Core@6.TRUNK (0)
  - bela.CPU_Cache@6.TRUNK (0)
  - bela.CPU_Logic@6.TRUNK (0)

Fig. 9: Externally Sourced ProjectIC Dashboard Widgets

In the example above, we can see custom widgets displaying verification results from the user workspace, IP Requirements from the Jama requirements management system, and hierarchical bug defects collected from the Jira defect tracking system. All of these are associated with a particular IP or IP version and are updated automatically whenever a new release is made.

5. Summary

While fulfilling an important need for large scale product development in automotive, aerospace, and other large industries, we've seen that the traditional PLM model suffers from major barriers to adoption in the semiconductor space. The high TCO and generally inflexible workflow is a bitter pill to swallow in a dynamic design environment, and generally seems unsuited for IC design. We've seen that a new type of lifecycle management - IP Lifecycle Management (IPLM) - is required to enable an IP-Centric, transparent approach to managing the IP ecosystem for large SoC's. IPLM is low maintenance with relatively few insertion points into the user workflow, and supplements the existing design environment with a powerful workspace management and multi-site data management platform that are prerequisites for an SoC design flow.

The ProjectIC™ IPLM platform centrally maintains project configurations (akin to BoMs in a PLM tool) and user/group and IP permissions, and can organize requirements and documentation by IP, user workspaces, and hierarchical dependencies... i.e., it provides the most critical functions of a PLM system, but without the cost and increased maintenance overhead.

By providing key workspace management / release functionality and leveraging on these to harvest metadata and inject it into the IP Management platform, designers can finally realize the low overhead and nimble Lifecycle Management system the semiconductor industry has been waiting for.