Benefits of IP-Centric Design

Not Just For Design Reuse

In SoC's, IP has become an important part of the design process and has spawned many new IP content providers, tools companies and general methodology changes to accommodate.

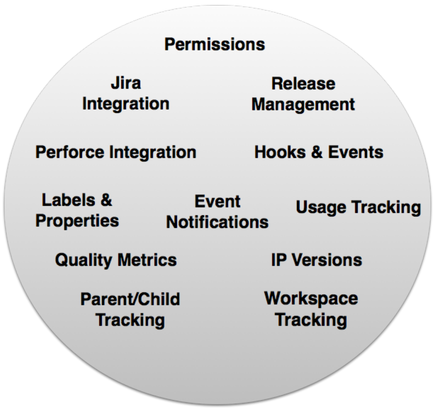
One basic assumption with an IP-based design flow is that the underlying building blocks are reusable, and that the efficiencies we get from reusing commonly found blocks makes the extra effort required to publish the IP in a reusable form worthwhile. However, in some situations reuse is difficult. - for example custom layout in todays processes is not easily reused when the process shrinks and often must be completely re-implemented.

Circuits can also behave in a non-linear manner between processes and must be designed differently. For some companies this lack of reuse undermines the value of maintaining IP blocks and drives the focus to project specific blocks and derivatives in future designs.

There is significant value in an IP-Centric design methodology, even in the absence of reusable IP blocks. This paper discusses that concept

What Makes An "IP"

An IP contains many different forms of data, the key component of an IP-centric design methodology is that these many forms of data need to be kept together and treated as an atomic unit.



Design Dependencies

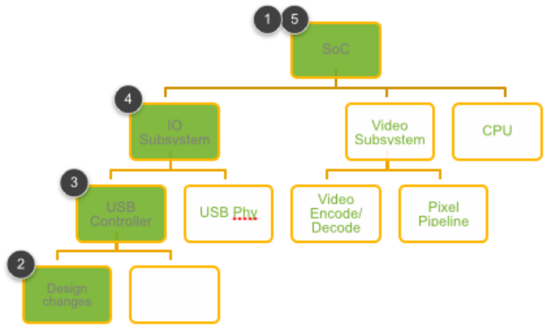
When designing with IP's an important consideration is to track subsystem dependencies. IP blocks are often designed by independent teams, and a system is needed to manage dependencies and track which versions of IP's work well together.

In the example below we see that an SoC and its hierarchical resources have been defined, including the versions of each IP in the system. In this example we show how a change in one of the lower level subsystems of the SoC impacts the release of the top level SoC.

This "propagated" release is one example of the dependency tracking we need in an IP-centric design system.

Another benefit of this dependency aware approach is that users can create workspaces at any point in the hierarchy and perform editing/testing at the subsystem level. In an IP-centric design methodology hierarchy is malleable (assuming you have the necessary test structures to validate the design)

Hierarchical release example: Attempt to release SoC after designer has made changes within USB Controller



1. *Attempt to release SoC*
2. *Changes in lower level IP are identified*
3. *Lowest level IP containing changes (USB Controller) released*
4. *Release propagated to IO Subsystem, and IO Subsystem released*
5. *SoC released using newly released subsystem*

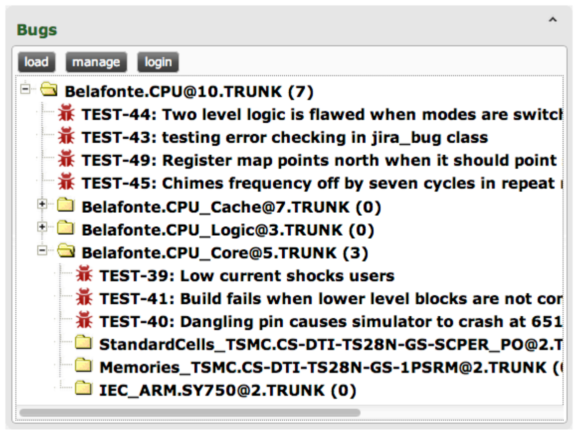
methodics™

IP-Centric Issue Tracking

One area that has been a problem SoC's is tracking bugs across the various components. Its not always clear at what level a bug should be assigned since designers often don't have visibility beyond the interface to a particular block, and as blocks are combined within an SoC level is a challenge to track bugs across hierarchical subsystems.

The bigger issue is that the tools traditionally used to track these bugs are project-centric rather than IP-centric and all of the interesting reporting and tracking is done in the context of the project. Without a way to align IP-centric hierarchy of the design with the top level SoC project users are forced to resort to manually maintained spreadsheets with all the human error concerns that go with that approach.

An example of an IP-centric hierarchical approach to defect tracking used in ProjectIC is shown in the image below.



```
Belafonte.CPU@10.TRUNK (7)
    TEST-44: Two level logic is flawed when modes are switcl
    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)
```

Parent/Child tracking

An extension of the IP-centric tracking is the need to "discover" bugs in parent/child design data,

When a copy is made of an existing block and used as a starting point for a new design, what happens if we later find bugs in the source design? Can we notify the design consumers and save them the trouble of finding the bugs independently.

The same is true of the downstream child block. If we find a bug in a copy can we notify the parent block designers?

To handle this issue we need a way to collect these parent/child bugs and consolidate them in a view of the bugs that affect the current design. The design owners can then decide if they care about these issues and can "accept" them as dependencies.

Workspace management

Another area that benefits from an IP-centric approach is creating and updating user workspaces. Maintaining each functional block as a standalone entity in your IP management system allows control of the versions of each block that is used in a workspace. Releases of blocks in a design can be managed independently and easily communicated to the team members, including new configurations of the design.

Another requirement is the ability to diff at the IP level, not just the constituent files in the data management system. This included the resources (subsystems) used in the 2 versions of a hierarchical block and other important metadata such regression suite failures, routability etc.

IP version tracking and reporting

Tracking how an IP is being across designs is useful in many ways. IP owners can easily reach to the IP consumers when looking for feedback, when communicating important bugs, managing behavior, etc.

Another way these relationships are leveraged is to track the context in which an IP block is being used. What components it is interfacing with, what versions of components have been proven to work correctly together, and other key contextual information?

This kind of analysis identifies subsystems and their versions that have seen successful deployments in parent SoC's and reduces risk for future integrators.

Summary

We've discussed how an IP-centric approach to design large SoC's brings a number of techniques to help with managing versions, releases and tracking quality. We've also seen that an IP-centric design methodology will work well with blocks that traditionally haven't been packaged as IP's. Essentially any block in an SoC can benefit from these kinds of techniques without the requirement for IP publishing/packaging and the other overhead usually associated with a fully reusable IP based SoC bill of materials.

For more information on Methodics products please email contact@methodics.com.