Review

Today’s silicon product designs are complex systems composed of multiple components (IP). Due to market demands for shorter product design cycles as well as market-segment specific design requirements, it is not practical for every product development team to independently design and validate silicon IP required for a product design. Therefore sharing RTL (Register Transfer Level) and verification IP is a necessity. In many cases, the IP blocks themselves are being developed in parallel with the product designs and sharing these dynamically changing IP blocks between multiple concurrent product design efforts is a challenge (fig.1). This paper describes an innovative methodology to meet this challenge. This methodology consists of two different working models applied to two phases respectively during the design production. The first working model applied during the early phase design work due to heavy changes including architecture and feature changes. The second model applied in the later phase of design work when the changes were more localized within IP and the requirement of turnaround time gained heavy weight.

Abstract

The Front End (FE) system data are normally collected and stored in a Register Transfer Level (RTL) database. Each team maintains its own database with a handful of IPs shared or co-developed between them.

Fig.3 Product X and Y are managed by separated teams and environments. E.g., product X is in “blue” environment and Y in “green” environment. The changed component works in “blue” environment may not apply in “green” environment after sync. It is true in vice versa. Furthermore, the product trees X and Y are dynamically changing constantly. A CI tool developed at Intel was adopted to cut the tasks. A group of CI pipelines which: 

- Provide a more-robust environment of rapid control (new or changed code) to the “golden” master repository (Product X or Y in the above example).

- Allow frequent integration and testing through a series tools and regressions such that no errors can arise without developers noticing them and correcting them immediately. Therefore, the code changes from developers and migrated into the master repositories are guaranteed in a good quality.

1st phase of IP sharing methodology (Early stages of the project development)

The port operation is based on components. It seems quite simple at the first glance. However, Product X and Y must execute the exact same environment works: both collaborate, build flow and test back-end. The changes ported from Product X may not work in Product Y environment. It is not allowed to have any change to “alter” Product master repository because hundreds design engineers are working under it at the same time. Especially, in the early phase of design work, the changes in both products are very heavily involving architecture and feature changes. In other words, the big changes from one side break the other side design work most of the time.

Problem - Difficult to implement due to environment difference, dynamic change in each product and design-quality maintenance

Fig.4 Using CI pipeline to allow the frequent concurrent changes to the master repository and ensure the master repository database healthiness.

Simple sharing methodology

Each design team carefully defines and partitions their respective master repository into components. Any shared (or common) component across the products must have the same directory structure so that SCCS can map the directories (port) among components to the original contents from one component to the other while sustaining the complete history as shown in Figure 4. Product X is the source side and Product Y is the receiver side in the following example.

Fig.5 Simple port of components between the repositories

- Product X and Y are different market sides.
- They have different environment.
- Both dynamically move forward as the changing pieces are scattered.

Issues - Difficult to implement due to environment difference, dynamic change in each product and design-quality maintenance

The port flow starts from Product X to Product Y. It closes the Product Y tree to the “sync base” and port the changes in from Product X. Product Y has two sub-trees (refer to Fig.5, one from Product X and the other Product Y). From Product X, the common components are “conceptually” ported to both top and bottom sub-trees of Product Y. The rest components of Product Y are ported to the bottom sub-tree of Product Y. Normally the port operation is automatically done by the scripts unless the merge conflict occurs. Once the port is completed, the script make the t宇nine to the CI pipelines which “fork” two independent t宇nine in parallel; one builds/replicates the top sub-tree (Product Y environment) and the other bottom sub-tree (Product Y environment). The t宇nine will be written back to the master repository of Product Y.

After this next stage port completed, the confidence of porting the upper part tree of Product Y to Product X is very high. The script follows the similar process: clone the tree from Product X and port in to the upper part tree of Product X and turn in to a single CI pipeline (just like the other users) in Product Y environment. The t宇nine is written back to Product X once it passes the pipeline.

Fig.6 In the case that either one t宇nine of Product Z failed in the CI pipelines, both t宇nine are rejected for next round fix.

- When the t宇nine are rejected. The “integrator” team debugs the error, makes the fixes and turn in to Product X. The next cycle is started. The process is repeated until the t宇nine pass the CI pipelines in Product Z.

Other tools are involved to make the IP sharing flow working smoothly

Along with the distributed revision control system SCCS and continuous integration CI tools, we also adopted other tool-defined tools serving the purpose of performance measuring (PM) and job scheduler management (JM).

PM is a performance unified measurement application. It is a system for measuring RTL (Register Transfer Level) environments. Although the performance is a unified mirror serving different projects, it provides a way for tracking, analyzing, debugging and profiling performance issues for RTL model simulations/iterations, compilations, and pipelines. PM tool integrates thousands of compute servers and workstations and associates them with queues of jobs, priority schemes and pipelines. PM box increases throughput of large computing-intensive pits and increases utilization of computing resources. It is a “smarter” PM which manages and runs the flows. Through the GUI, users can keep track of their turned progress, biases the big flow for debugging, the flow and even collaborate with other users. The PM tool is a distributed computing environment, which will execute the user’s jobs in the pipeline run, user can terminate his/her own runs in the pipeline immediately and start the debugging. The tool improves design engineer productivity and reduces the load on computing basic resources.

Summary

The methodology provided a convenient vehicle to share the FE system data (RTL, codes) between the different products with different environments, especially the shared data is “seamless” within huge flow at Intel.

Our IP sharing is actually “IP co-development” in a more precise way, because we need merge capability and the codes change history on top of the contents. Helped by other internal tools like CI, GC, PM etc., we achieved multiple goals:

- Robust and efficient sharing of IP blocks
- Ability to apply bug fixes in either product
- Assurance that the posted data is healthy
- Concurrent port process (asynchronous) along with the other users’ t宇nine
- Simple debugging and tracking of errors

You may wish to see what Product Z is required in the flow. Can we replace Product Y by Product Z? The key is to keep in mind that Product Y is in the ‘real’ pipe out database (OB) of the project. Product Z is just a ‘buffer’ zone for integrating the common components from Product X to Product Y. There are two sub-trees (ports) in Product Z: it is hard to detect any “cross reference” between the two sub-trees. We must ensure that Product Y’s tree is not compromised by changes from Product X. Effectively we are guaranteeing isolation of the two trees, especially for the tape out product Y. Therefore, keeping Product Y with a single tape-out branch is necessary.