An Innovative Methodology for RTL and Verification IP Sharing Between Two Projects

Albert Xu and Joonyoung Kim

Visual and Parallel Computing Group, Many Integrated Cores (MIC), Intel Corporation

Abstract

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 all of the IP required for a product design. Therefore sharing RTL (**R**egister **T**ransfer **L**evel) 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.

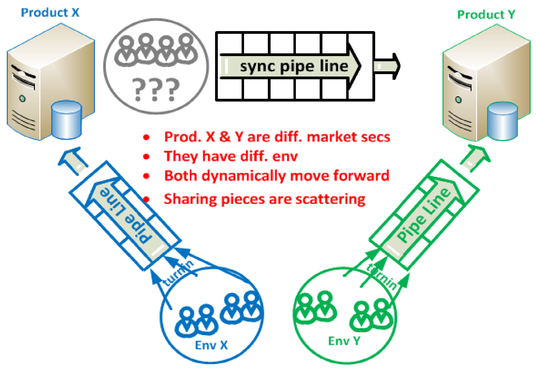

Fig.1 How to share IP between different products dynamically?

Introduction

The Front End (FE) system data are normally collected and stored in a **R**egister **T**ransfer **L**evel (RTL) database. Each team maintains its own database with a handful of IPs shared or co-developed between them.

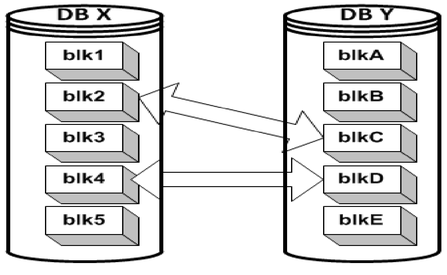

Fig.2 Databases DBX and DBY belong to two different design teams. Only IP blocks blk2/blkC and blk4/blkD are shared between the two projects. We chose a *commercially available* distributed revision control product (SCCS) which has the capability to split the entire database into components such that each component is a standalone database itself. It can share (i.e. merge and keep the history) the data at any component level across the products. The "blk" in Figure 1 is equivalent to the component in SCCS system. SCCS stands for Source Code Control System. "SCCS" is used as an "encoded" tool name throughout this paper.

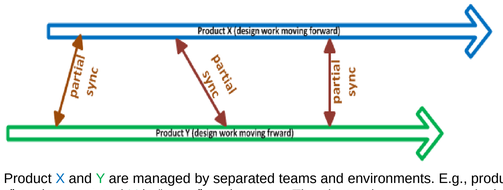

Fig. 3 Product X and Y are managed by separated teams and environments. E.g., product X is in "blue" environment and Y in "green" environment. The changed component works in "blue" environment may not work in "green" environment after sync. It is true in vice versa. Furthermore, the product lines X and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

- Provide a multi-developer environment of rapid commits (new or changed codes) to the "golden" master repository (Product X or Y in the above example);
- Allow frequently integration going through a serious builds and regresses 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;

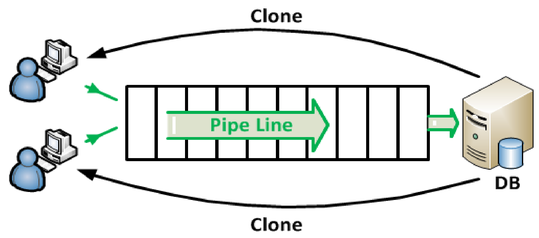

Fig 4 Using CI pipeline to allow the frequent concurrent changes to the master repository and ensure the master repository database healthiness.

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 structures so that SCCS and CI tools can do the port ("port" merges the contents from one component to the other while sustains 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.

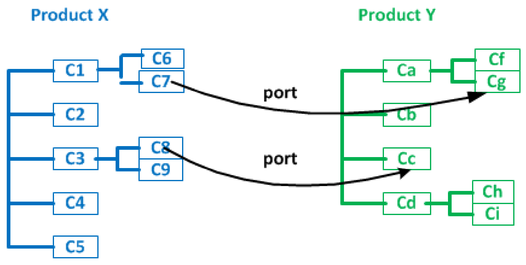

Fig. 5 Simple port of components between the repositories

Issue – Difficult to implement due to environment difference, dynamic change in src/dest products and design quality insurance

The port operation is based on components. It seems quite simple at the first glance. However, Product X and Y do not have exact the same environment (mainly tools, collaterals, build flow and test bench). The changes ported from Product X may not work in Product Y environment. It is not allowed to have any chance to "nuke" Product Y 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 broke the other side design work most of the time.

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

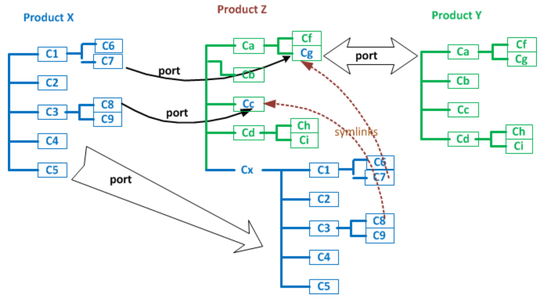

Fig. 6 "Integration" repository Z is inserted between two tape-out repositories. It was built from trees of Product X and Y. The new repository becomes a "buffer zone" for "integration" purpose. The entire tree of Product Y is ported to the upper tree of Product Z and Product X, except common components C7 and C8 in this example, is ported as Cx in Product Z. The "physical" common components are ported to the upper part of tree. The components of C7 and C8 of Cx in Product Z are the symbolic links pointing to the 'Product Y' portion of the tree. This is to make sure that all the components in the upper part tree are real entities so that Product Y doesn't get empty symlinks through "port" operation.

Automating "port" through cron

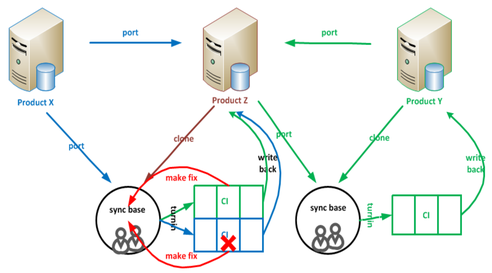

Fig. 7 The port flow starts from Product X to Z. It clones the Product Z tree to the "sync base" and ports the changes in from Product X. Product Z has two sub-trees (refer to Fig.5), one from Product X and the other Product Y. From Product X, the common components are "conceptually" ported to both top and bottom sub-trees of Product Z. The rest components of Product X are ported to the bottom sub-tree of Product Z. Normally the port operation is automatically done by the scripts unless the merge conflict occurs. Once the port is completed, the scripts make the turnins to the CI pipelines which "forks" two independent turnins in parallel; one builds/regresses the top sub-tree in Product Y environment and the other the bottom sub-tree in Product X environment. The turnin will be written back to the master repository of Product Z.

After the first stage port completed, the confidence of porting the upper part tree of Product Z to Product Y is very high. The scripts follow the similar process: clone the tree from Product Y, port from the upper tree of Product Z and turn in to a single CI pipeline (just like the other users) in Product Y environment. The turnin is written back to Product Y once it passes the pipeline.

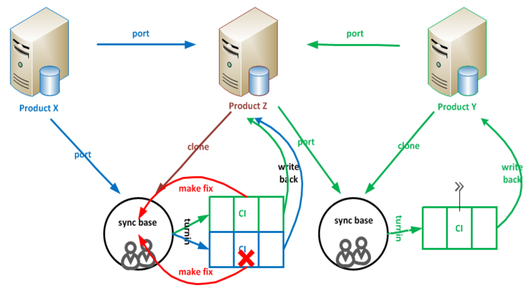

Fig. 8 In the case that either one turnins of Product Z failed in the CI pipelines, both turnins are rejected for next round fix.

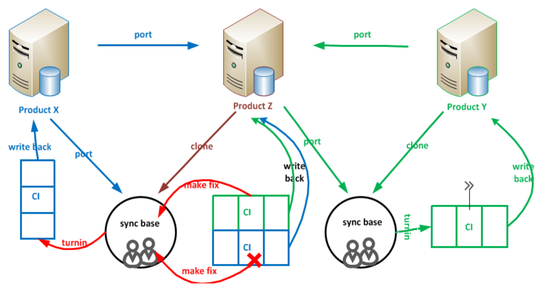

Fig. 9 When the turnins are rejected, The "integration" team debugs the error, makes the fixes and turn in to Product X. The new cycle is started. The process is repeated until the turnins pass the CI pipelines in Product Z.

You may ask why Product Z is required in this flow. Can we replace Product Y by Product Z at all? Keep in mind that Product Y is the real "tape out" database (DB) 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 (projects) in Product Z, it is hard to detect any "cross references" between the two sub-trees. We must ensure that Product Y's tree is not contaminated by collateral 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 tree is necessary.

2nd phase of IP sharing methodology (Later stage of the project development)

As the design work in both products entered the later phase, the changes were more localized and the turnin successful rate got higher. In order to speed up the turnaround time, the second configuration and flow was introduced. The middle "man" Product Z was eliminated.

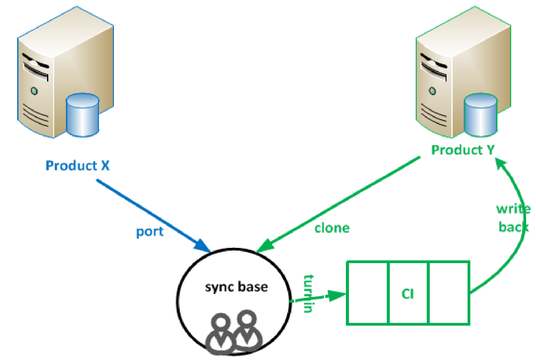

Fig. 10 The port flow starts from Product X to Product Y directly. It clones the Product Y tree to the "sync base" and ports the changes from Product X. The "sync base" has only one single tree, i.e. Product Y. If there is no merge conflict followed by successfully build/regress in the CI pipeline of Product Y environment, the turnin will write back to the master repository of Product Y. This flow is simpler and has shorter turnaround time. At the same time, the design quality is very tightly controlled.

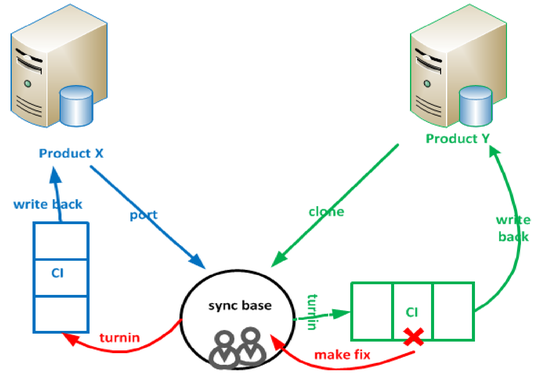

Fig. 11 In the case that the turnin is rejected, it will be debugged and fixed in Product X followed by another round auto-sync.

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 Intel-developed tools serving the purpose of performance measuring (PM) and job scheduler management (NM).

PM is a performance unified measurement application. It is a system for measuring RTL (Register Transfer Level) environment performance in a unified manner among different projects. It provides a standard tool kit for tracking, analyzing, debugging and profiling performance issues for RTL model simulations/emulations, compilations, and turnins; NM is a distributed computing/batch-processing platform. It clusters tens of thousands of compute servers and workstations and associates them with queues of jobs, priority schemes and policies. NM tool increases throughput of large computing-intensive jobs and increases utilization of computing resources. It is a "smarter NM" which manages and runs the flows. Through the GUI, users can keep track of their turnin progress, brows the log files for debugging the errors, and even collaborate with other users by looking at their turnin progress. When the first error occurs in the pipeline run, user can terminate his/her own turnin in the pipeline immediately and start the debugging. The tool improves design engineers' productivity and reduces the load on computing batch resource.

Summary

This 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 "scattered" within a huge tree. Due to the complexity and design flow reasons, each project has its own repository (or DB) independently. 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 Intel internal tools like CI, NM, PM and etc., we achieved multiple goals:

- Robust and efficient sharing of IP blocks
- Ability to apply bug fixes in either product
- Assurance that the ported data is healthy
- Concurrent port process (seamless) along with the other users' turnins
- Simple debugging and tracking of errors