Proc. of the Second Intl. Conf. on Advances in Electronics and Electrical Engineering — AEEE 2013

ISBN: 978-981-07-5939-1

Subversion(r): Empirical Design Methodology from the Perspective of Integrated Circuit Design

Radoslav Prahov, Holger Schmidt and Achim Graupner
Zentrum Mikroelektronik Dresden AG
Dresden, Germany

***Abstract*—An aspect of primary significance in integrated circuit (IC) design is configuration management of design data, i.e., the task of keeping a project comprising a multiplicity of revisions well organized. Apache's Subversion(r) is a software tool that can facilitate this task. It manages revisions of documentation, source code, and a wide variety of files, and it automates storing and retrieving revisions. Unfortunately, Subversion(r) provides insufficient support for IC projects consisting of large numbers of managed items. We address this problem by introducing, discussing, and demonstrating several approaches that improve the performance of Subversion(r) when handling a vast amount of files and directories. Our approaches are division of the working copy into smaller pieces with a decent granularity, conversion of the working copy into a single tarball file, and implementation of a referred central working copy. Each method is incorporated into the configuration management flow through a lifecycle of IC development, which offers the opportunity to compare and validate each technique.**

***Keywords*—configuration management, Subversion, design methodology, performance.**

I. Introduction

With the ongoing requirements for accomplishing higher productivity and quality while ensuring effective control before, throughout, and beyond the integrated circuit (IC) development process, configuration management (CM) of design data has become an important aspect of modern IC projects. Its role is to assist designers by controlling, tracking and coordinating every single change that occurs in the file system during a project lifecycle by gathering evolutionary revisions [1]. This allows users to perform unlimited updates to the project information, but at the same time, they can be assured that each user has the latest version. Users with an old version have the ability to either update their working copy to the most recent version or continue employing their old one and propagate their changes afterwards. Conversely, users with a head version (the version presently designated as most current) can check out a previous version; for example, for comparison if anything regresses. Changes during IC projects could have a diverse character. Beyond basic file modifications, they may also involve adding, removing, or updating directories; modifying the hierarchy; altering group permission and file and directory ownership; and renaming files and directories [2],[3].

Although automated support for CMs has existed for over thirty years, its prominence in the framework of IC design has sharply increased during the last decade [4]. Early automated support tools suffered from inadequate functionality and applicability. In contrast, modern tools offer advanced utilities and features [5]-[8]. Despite the evolution from simple tools to comprehensive environments, automated support for the CM is still confronted with challenges due to the advent of new innovations and technologies [9]. One such challenge is the ever-increasing complexity of IC projects [10]-[12]. For instance, because of the more complex verification flows with every new IC generation and process node, IC projects tend to comprise ever-growing design data.

As the most prominent CM tool, Apache's Subversion® (SVN) must respond to the trend toward more complex IC designs with higher file counts. However, it has a deficit when it comes to dealing with great numbers of managed items, as its efficiency proportionally degrades with a growing quantity of files and directories [13]-[15]. In addition to breaking a tool's environment, this leads to unacceptably long operation times. Even in projects with average complexity, this is a severe issue. In the majority of cases, the increased operation time drags down productivity because the longer waiting time cannot be used effectively. Reducing it not only accelerates the IC design process, but also lowers the stress level for the project's IC developers.

The present study was designed to evaluate different approaches that could adapt SVN to handling a vast quantity of files and directories more efficiently. The approaches were incorporated into the CM and were employed during the lifetime of an IC design project. The effect on SVN is compared, and the improvement is defined in this study.

The remainder of this paper is organized as follows. In section II, a concise evolutionary history and features of SVN are introduced, related work is presented, and challenges of migrating from one CM tool to another are identified. This is followed by section III, where the problem of controlling a multiplicity of files and directories is addressed, different design methodology techniques that allow SVN performance improvement are presented, and their implications for the integrated circuit workflow architecture and revision control system are discussed. Section IV explains the industrial project into which the design methodologies were

Proc. of the Second Intl. Conf. on Advances in Electronics and Electrical Engineering — AEEE 2013

ISBN: 978-981-07-5939-1

incorporated and presents an evaluation and discussion. Section V concludes the paper.

II. Background

Since the advent of Concurrent Versions System (CVS), initially as a set of shell scripts coded by Dick Grune (1986) and later converted to a C program by Brian Berliner (1989) [16], CVS has become one of the most prominent version control tools, widely employed in various projects. Even now, over two decades later, it is the second most widespread tool in terms of market share (13%) [17]. During this extensive period of development and usage, its benefits were identified; however, substantial problems emerged. Hence in 2000, a group including former CVS developers launched SVN, a version control tool explicitly intended to be a successor of CVS with a similar design and improved functionality [18]. Its first official release arrived in 2004. At present, SVN is the most widespread version control tool, with a market share of 51% [17].

A. *SVN*

SVN's repository core is the storage backend where all versioned data are stored. Each time a client successfully commits certain changes, the SVN repository creates a new snapshot of the versioned file-system tree, called a revision or version. An increasing, unique number globally identifies each version. The snapshot contains the revision directory structure, file meta-data, and file contents, which might be delta compressed to save space. The delta compression keeps only the differences between successive versions of files. To retrieve a specific file revision, SVN composes a sequence of deltas up to the last full version. Because searching through all file revisions is time-consuming, full versions called skip-deltas are inserted between deltas.

On the client side, for each file in the working copy, a pristine copy, the revision number (on which the local file is based), and the timestamp of its last update are stored. The pristine copy allows using several commands without any repository interaction: checking file status, comparing files with their unmodified version (svn diff) and restoring contents (svn revert). Committing changes from the client's working copy to the repository does not trigger a synchronization of other locally unmodified files. Thus, after committing a subset of the working copy, it is left in a mixed-revision state; therefore, the base revision number must be tracked for every file and directory. To reproduce all upstream changes, the svn update command pulls all changes, optionally only up to a specified revision. Local modifications are automatically reintegrated, and the usual conflict-resolution workflow is applied.

SVN supports branching, merging, and tagging using an additional directory layer in the repository hierarchy; typically, main development happens in the trunk, while development of branches and tagged versions reside in corresponding named directories. Since version 1.5 (2008), merge information is automatically stored in the path meta-data (svn:merginfo). This simplifies merging between branches and the trunk, as parental relations are not naturally represented in the repository tree structure. However, compared to most common distributed version control systems, several merging issues still remain ([18], cf. Chapter 4).

B. *Problem Statement*

Despite SVN's dominant market share and improved functionality, SVN often suffers from a performance deficiency when it handles a multiplicity of managed items. This is not actually recent news, nor an exclusive trait of SVN, since the signs were first observed in ancestral CVS. In 1989, prolonged times were recorded while CVS managed the Prisma™ project by Prisma, Inc., which comprised over 17,000 files [19].

Subsequently the effect of escalated IC project complexity upon the behavior of SVN was assessed in [13]-[15]. How the system can be adapted to a multitude of files with a different origin was presented in [14] by investigation of several typical user cases. SVN performance limitations and suggestions for how they can be overcome were also discussed. When investigating sources of bottlenecks in SVN, the most significant finding was the relationship between the number of managed items and the execution time of SVN operations. Furthermore, the relationship is quadratic for commit commands and linear for add/checkout commands.

Taking into account all findings and results of the studies mentioned above, the optimum effectiveness of SVN can be achieved by keeping project data compact and locating the repository and working copy in a RAM disk on a sufficiently powerful machine with an adequately spacious cache area. However, with the ongoing growth of IC project complexity, the reduction of design data is hardly applicable. Even though hosting the repository and working copies in a RAM disk has proven to be the most efficient configuration [14], in our view, it remains a theoretical technique with limited possibility of application because in a considerable proportion of cases, the required allocation of space is substantial. Furthermore, implementing mandatory security measures, such as backup and regular snapshots, is more complex. Another setback is that job distribution techniques, such as using a load-shared facility (LSF), which is widespread, cannot be employed.

Regardless of the flaws discussed above, a substitution of the CM tool is often not an alternative and is hardly applicable because of the CM's tight integration into the design workflow. For instance, in the project for Apache Software Foundation's OpenOffice™ (for which the repository consisted of over 66,000 files), it was reported that various tools and wrapper scripts, such as issue tracking, authentication, and some tools specific to the project (EIS, LION, etc.), had to be entirely redesigned [20]. Furthermore, programs that were supplementary to the IC development process (such as tools facilitating the design and SoC project management and the GUI support tools) are only compatible with specific CM tools, usually SVN and Perforce™ (trademark of Perforce Software, Inc.). Hence, in this work, we address the SVN bottleneck and propose three approaches that are capable of mitigating the SVN dependence on the amount of managed items.

Proc. of the Second Intl. Conf. on Advances in Electronics and Electrical Engineering — AEEE 2013

ISBN: 978-981-07-5939-1

III. Design Methodology

In the following section, three different techniques are presented as each of them allows reducing the quantity of files and directories that SVN handles per operation.

A. *Division of the Working Copy into Smaller Pieces with a Decent Granularity (DM1)*

The first proposal is based on division of the working copy into smaller pieces with a decent granularity, organized in a block-based hierarchy. In such a structure, each block can be processed individually in parallel. In addition to parallelizing the operations, this allows reducing the number of files to be manipulated (submitted/updated) at once.

Putting this approach into practice can be achieved by different methods. One method would be to divide the project into as many various unit types as possible. A unit type constitutes a heterogeneous, separate design part; for example, a directory of a circuit block or sub-block. Different unit types are generally limited by the character of the project data. Therefore, a detailed verification under the project hierarchy and design data might be needed for such architecture.

We chose to divide the data as shown in Fig. 1 and Fig. 2. Since IC designs possess a decent granularity by nature, the analog library depicted is suitable for dividing into its heterogeneous subdirectories (bandgap, oscillator, and vref). However, the effort of reorganizing existing projects should be taken into consideration. Even so, this is one of the focal advantages of the approach since the method can be employed directly out-of-the-box for a substantial portion of IC designs.

B. *Conversion of the Working Copy into a Single Tarball File (DM2)*

The basic principle of the approach introduced in [13] and [14] is depicted in Fig. 3. The quantity of managed files and directories is decreased by combining them into a tarball archive, which accelerates SVN. Initially, this principle was only proposed for binary files, for which the number of files can be efficaciously reduced by two mutually complementary means [13]. In the first case, the whole directory structure is converted into one single monolithic block. For that purpose, data that are to be imported into the repository are transformed into one file via a simple tar operation and then the tar file is uploaded to the repository. When the data must be accessed (updated), the tar file is untarred and they are again available. Since the revision control system is facilitated and does not need to recursively deal with the initial directory structure, but rather with just a single block, an acceleration of about 15 times was reported [13].

The second method takes the first case a step further by compressing the single block. The process is essentially the same, except that the tar file is compressed before being uploaded to the repository. This could be done in various ways; albeit, a simple and effective one is the standard UNIXgzip command. Conversely, when the uploaded file has to be accessed, it must also be decompressed. Due to the compressed character of the block, an additional speed boost and shrinkage of the space that is consumed on the server are observed.

Since in [14] both principles were proven to be efficient, not only for binary files, but also for a wide range of file formats, the latter approach was implemented in the IC project as shown in Fig. 4. The tar/compress and untar/decompress steps were entirely automated due to their routine and particularly error prone character.

```
+- analog_library
|  +- bandgap
|  +- bandgap_resistors
|  +- bandgap_amplifier
|  +- tb_bandgap
|  +- oscillator_20kHz
|  +- oscillator_trimunit
|  +- oscillator_schmitttrigger
|  +- tb_oscillator
|  +- vref_18
|  +- …
```

Figure 1. Classical structure of analog library

```
+- bandgap_library
|  +- bandgap
|  +- bandgap_resistors
|  +- bandgap_amplifier
|  +- tb_bandgap
+- oscillator_20kHz_library
|  +- oscillator_20kHz
|  +- oscillator_trimunit
|  +- oscillator_schmitttrigger
|  +- tb_oscillator
+- vref_18_library
|  +- vref_18
+- +- …
```

Figure 2. Organization of analog library into smaller pieces with decent granularity. Below level 1, each subfolder could be manipulated in isolation.

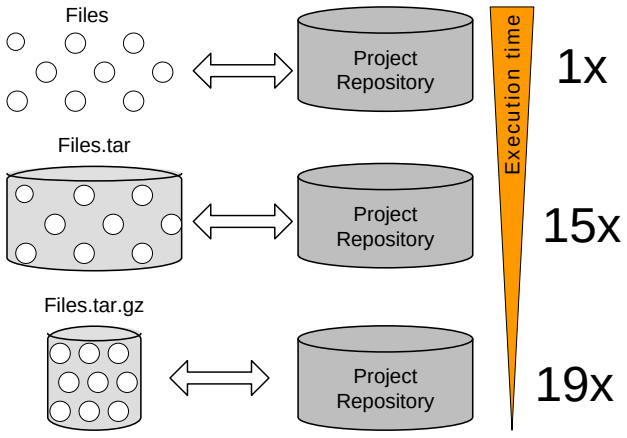

Figure 3. Three techniques for importing into the SVN repository: plain file structure, simple tar file, and compressed tar file, with respective performance.

ISBN: 978-981-07-5939-1

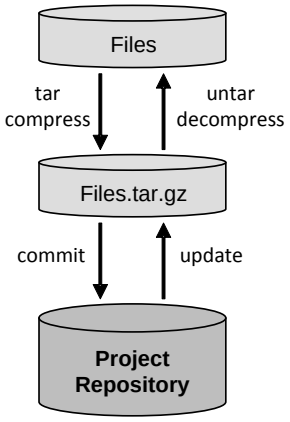

Figure 4. Details of compressed tar file approach.

The first step was carried out in the sequence:

- Directory existence verification. This step verifies that the directory that was selected to be manipulated exists. If not, the sequence is terminated.
- SVN management validation. Whether or not the specified directory is already managed by SVN is validated. If SVN managed, the directory must be erased from the SVN repository.
- Tar file existence verification. This step verifies whether a tar file with an identical name exists. If so, its content is compared against the specified directory. If they are equivalent, the sequence is discontinued. Otherwise, the step is executed and the tar file is either brought into existence or updated.

The next step, untar/decompress, is executed on the algorithm:

- Tar file update verification. Whether the tar file has already been updated in the local directory structure is verified. If not, the algorithm is terminated.
- SVN management validation. This step validates whether the directory included in the tar file has already been managed by SVN. If it is, the algorithm is discontinued.
- Workspace tar content SVN management verification. If the directory already exists in the user's workspace but it is not managed by SVN, it is automatically removed. Next, the tar file is extracted.

The application of the automated transformation of the initial data structure into a monolith block is not only to allow eliminating all trivial tasks, but also to keep this relatively error-prone phase protected.

C. *Referred Central Working Copy (DM3)*

This approach differs from the others in that it involves maintaining a central working copy for read-only access. A data unit being designed within an IC project is generally developed by a single engineer but at the same time is referenced by others. Since a significant proportion of elements in the user's working space is not modified and not directly employed (but elements still need to be referenced), the elements can be referred to the central working copy through soft links, whereas all developed elements remain in the regular working copy.

Implementation of the structure presented above is illustrated in Fig. 5. It consists of a three-level hierarchy, adding an additional level to the conventional server-client configuration traditionally employed by the revision control system.

Importing (checking in) the design data to the repository is performed with the standard method; i.e., the new structure does not affect this process at all. When the data are to be checked out, instead of being transferred directly from the server to the client, they are initially copied onto a central replication area and then the workspace is created. Each unit from the workspace could either point to the replication area as a soft link or could be represented physically. Soft links have read-only access because blocks that are in the replication area cannot be modified. If modification is required, they must be transferred to the local workspace first. Each block can be converted at any time, replacing a link with local data and vice versa. The replication area cannot be updated (for fixed revisions) in terms of replacing an old version of a block with a newer one, but it can comprise more than one revision of a certain block. Of course, all outdated block versions could be removed once they are no longer required. From the methodology description up to this point, it could be inferred that the replication area behaves as a typical second server, except that its data do not require being backed up, as they can readily be recovered at any given moment and they do not contain any modifications.

This approach has the prerequisites of block-oriented design data structure and decent granularity, as discussed previously for Fig. 2. This method smoothly allows each block to be fetched into the working copy either as a soft link or as physical data. For instance, blocks that are never employed could be referred to the central working copy, whereas all the others would remain part of the regular working space. This measure also allows parallel processing and saves disk space.

The linked central working copy technique can be implemented with different methods. One is to develop proprietary scripts. However, especially in the field of IC design, this tends to be limited by the ability of the programmers who develop and maintain them (might lack training or experience and might not be diligent). Furthermore, scripts are relatively insufficiently flexible, and even a slight environment alteration could trigger discrepancies and inconsistencies, which are difficult to fix. Therefore, we chose a tool available on the market for implementing the technique: Methodics' BuildIC™. It is an SoC assembly engine, part of a platform for SoC design management [21]. However, we consider it to be also beneficial for workspace management, as it has a "shared area," which has the identical functionality as the replication area.

Proc. of the Second Intl. Conf. on Advances in Electronics and Electrical Engineering — AEEE 2013

ISBN: 978-981-07-5939-1

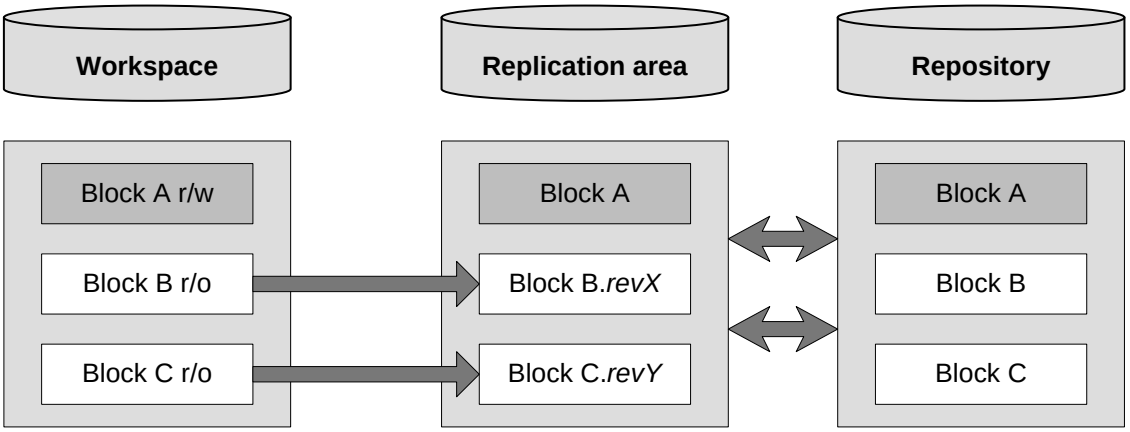

Figure 5. Schematic of the referred central working copy approach. Blocks that are not required in the user's workspace are referred to the central working copy.

IV. User Case

The design methodologies that were described in the previous section were incorporated into the CM workflow of our ZSPM1000 Smart Power Management (SPM) IC project. The ZSPM1000 is a configurable, true-digital single-phase pulse-width-modulation (PWM) controller for high-current, non-isolated DC/DC power supplies supporting switching frequencies up to 1 MHz. It includes a PMBus™-configurable digital power control loop that incorporates output voltage sensing, average inductor current sensing, and extensive fault monitoring and handling options. Project data comprised 48 blocks with a total of 50,000 files and a total size of 5.9GB.

A. *DM2*

Initially, DM2 was implemented from the foundation of the project. According to the design team, this principle allowed them to solve the SVN performance limitations; however, the improved efficiency comes at a cost. The application of SVN on a file level is not possible. Standard features of revision control systems, such as file history, revert, selective checkout and locking functionalities, are not available.

Furthermore, changes between different revisions are not traceable. For example, the following fundamental questions for revision control systems cannot be answered. Which files were changed? Who made the change? What was changed in the file? What did the file contain in a particular revision? All difference comparison futures are inevitably lost. The graphical user interface support, which is one of the focal advantages of SVN, is no longer efficacious.

In our efforts to address these issues, a policy of extensive commit messages was put in force. Nonetheless, we rapidly came to realize that this hardly helps when design teams are spread over different locations. Even so, in our experience, the issue can be avoided if an intellectual property (IP) project design is employed. An IP constitutes a standard functional block that is part of the IC but was developed separately from the project either internally or sourced by a third party. During the design phase, IP blocks are immutable. As such, they are suitable for tar and untar because of the infrequency of modifications to them.

Another hurdle is the extra step of transforming the initial data into a single block. This should be considered a critical phase, due to its routine and error prone character. A possible solution is an automation of the process of handling the data with a wrapper script. Although application of the automation script mitigates the problem, it should still be considered critical. By presumption, any directory with a name equivalent to the content of the tar file is deleted because we determine that by and large, it is left from any preceding execution; however, the folder might contain important, modified, but not yet committed project data as well. Moreover, due to the development and maintenance of scripts, a further level of complexity is added to the project design flow.

B. *DM3 and DM1*

Due to the drawbacks identified in DM2, an alternative combined DM3 and DM1 CM workflow was integrated into the ZSPM1000 project. The workflow achieves an improved performance compared with DM2. This can be explained by the parallel algorithm in which operations are performed.

When the same operation is executed in a sequential manner, the replication area is only composed during the first execution and reused later. Hence, the most rapid execution time is achieved when the replication area has already been brought into existence because of the minimal time that is required for establishing soft links.

In contrast, if the replication area does not already exist and all units are required to be available in a local workspace, the longest time is needed. However, in this project experience, only a small number of blocks were required in the users' workspace for write access (locally). This reflects a dedicated designer who needs to reference all blocks during a project lifetime but modifies only a minority of them.

The improved performance is explained by the reduced quantity of items that SVN handles (during checkout) per operation and also by the parallel manner in which the commands are executed. Furthermore, the establishment of links is performed in very little time. It should be noted that in

ISBN: 978-981-07-5939-1

contrast to the tar-untar approach, all benefits of the revision control system are preserved, which is a key asset of this mixed technique.

Several inconveniences were explored during the lifetime of the project however. Because of DM1, it is not possible to use a single operation to atomically commit multiple blocks that have been modified with related refinements because each block must be managed separately (Fig. 3). It was reported by the design team that although this slightly affects the revision control tool history by adding further revision numbers for each manipulation, it does not influence the workflow. Additional feedback confirmed that the application of this technique leads to significant performance improvement and reduction of the amount of data per operation.

The replication area in DM3 does not particularly affect the IC design workflow. It is not a critical element and does not require any special maintenance measures. Even though this area might be considered to consume extra disk space, it actually saves space because the user's working copies are reduced in size. The central working copy can be updated automatically by a post-commit script.

Nonetheless, the disadvantage of the automatic updating mechanism is the alteration of files without prior notice; for example, when a designer refers to a head revision of a given block and that block has changed. As a result, the designer would automatically be referred to the new head revision. However, in addition to the discomfort of unexpected changes in the working copy structure, this can lead to breaking the environment; e.g., regressions and debug sessions, which count on stable data. Hence, we chose to set up links to the central working area for fixed revisions and to update them when required.

v. Summary and Future Work

This paper has presented three different approaches that adapt the CM tool Subversion® to dealing with a multiplicity of managed items. Since each of the proposed approaches was ingrained in an industrial IC project flow from "scratchpad" to the final product, they were compared and validated in a realistic environment.

The demonstrated methods will be particularly beneficial in future IC design projects for which the revision control tool would be stretched to its breaking point with the increased quantity of project data.

Thus far we have mainly explored the design mythology from the user's perspective. Our next step in this research will be to perform a performance case study. We are interested in comparing the performance improvement of each technique.

Acknowledgment

The authors wish to thank the design team for the ZSPM1000 project at ZMD AG for their technical assistance and feedback during the lifetime of the project as well as Ms. M. Wallace and Mr. D. Aitken for proofreading the manuscript.

The work reported in this study was funded by the Seventh Framework Program of the EC under grant agreement no. 237955 (FACETS-ITN).

References

[1] 828-2012 – IEEE Standard for Configuration Management in Systems and Software Engineering, March 2012.

[2] C. Kidd, "The case for configuration management," IEE Review, vol. 47, pp. 37-41, September 2001.

[3] K. Hinsen, K. Läufer, and G. Thiruvathukal, "Essential tools: version control systems," IEEE Computer in Science & Engineering, vol. 11, pp. 84-91, November-December 2009.

[4] M. Rochkind, "The source code control system," IEEE Transactions on Software Engineering, vol. 4, pp. 364-370, December 1975.

[5] K. H. Lee, "Design and implementation of a configuration management system," Global Telecommunications Conference, vol. 3, pp.1563-1567, November-December 1993.

[6] A. Do, "The impact of configuration management during the software product's lifecycle," Digital Avionics Systems Conference, vol. 1, pp. 1.A.4-1 – 1.A.4-8, November 1999.

[7] A. Chan and S. Hung, "Software configuration management tools," Software Technology and Engineering Practice, pp. 238-250, July 1997.

[8] H. Yue, X. Liu, and S. Zhao, "Evaluate two software configuration management tools: MS Perforce and Subversion," Computational Intelligence and Software Engineering, pp. 1-6, December 2010.

[9] D. Kim and C. Youn, "Traceability enhancement technique through the integration of software configuration management and individual working environment," Secure Software Integration and Reliability Improvement, pp. 163-172, June 2010.

[10] X. Wang, W. Chen, Y. Wang, and H. You, "The design and implementation of hardware task configuration management unit on dynamically reconfigurable SoC," Embedded Software and Systems, ICESS '09, pp. 179-184, May 2009.

[11] M. Mehendale, "SoC – the road ahead," IEEE VLSI Design, January 2006.

[12] J. Burns, "Technology trends and implications on SoC design," IEEE SoC Conference, pp. 386, September 2011.

[13] D. Bell, "Performance tuning Subversion," IBM developerWorks, May 2007, accessed August 2012 http://www.ibm.com/developerworks/java/library/j-svnbins/index.html.

[14] R. Prahov, H. Schmidt, E. Müller, and A. Graupner, "Subversion(r): an Empirical Performance Case Study from a Collaborative Perspective on Integrated Circuits and Software Development," ICSESS, in press.

[15] R. Prahov, E. Müller, and A. Graupner, "Configuration Management from the Perspective of Integrated Circuit Design," IEEE 27th Convention of Electrical and Electronic Engineers in Israel, pp. 1-5, November 2012.

[16] P. Cederqvist, et al., "Version Management with CVS" (for cvs 1.12.13), accessed August 2012, http://ximbiot.com/cvs/manual/, 2005.

[17] "The open source development report," Eclipse Survey Report 2011, June 2011, accessed August 2012, http://www.eclipse.org/org/community _ survey/Eclipse_Survey_2011_Report.pdf.

[18] B. Collins-Sussman, B. Fitzpatrick, and C. Pilato, "Version Control with Subversion" (for Subversion 1.7), 2011, http://svnbook.red-bean.com/.

[19] B. Berliner, "CVS II: Parallelizing software development", USENIX Winter 1990 Technical Conference, pp. 341–352, 1990.

[20] The OpenOffice™ project, http://www.openoffice.org/.

[21] "The BuildIC™ SoC Development Platform," Methodics, accessed August 2012, http://www.methodics-da.com/products/projectic.