CONFIGURATION MANAGEMENT FOR SOFTWARE AND FIRMWARE AT PSI ACCELERATORS

T. Pal, T. Korhonen, Paul Scherrer Institute, Villigen, Switzerland

Abstract

The increased demand for usage of FPGAs in control systems has fueled a renewed interest in configuration management strategies for software and firmware. A pragmatic approach is described to progressively produce a toolkit for the management of firmware source code over time in an environment where the platforms and vendor toolsets change rapidly, with poor compatibility between versions. We take advantage of recent developments in virtualization technology, in conjunction with command line scripts as a 'means to an end' for our purpose.

INTRODUCTION

Requirements for the maintenance of firmware and software for embedded systems at existing scientific facilities as well as those planned for the future have been discussed in several articles [1-3]. The FPGA applications are built with vendor-supplied toolsets that can have very different behaviour and setup options between versions. The version cycle is also very rapid and migration to a newer toolset version can require a substantial amount of rework. In critical applications this could even mean that all the applications need to be verified and validated again just because the toolset has been changed, which can mean a significant increase in workload. In addition the lifecycle of applications in an accelerator environment greatly exceeds the lifecycle of typical consumer products. While a common consensus emerges with respect to the necessity of having a proactive approach, there does not exist, as expected, a unique solution. Instead, customized approaches are described, within the software configuration management paradigm, tailored to site-specific requirements. In the following, we present our strategy.

Our goal is to produce a toolkit of procedures to manage firmware source code over time in an environment similar to that described above. A generic approach of hardware and software entities is described for control applications at the PSI accelerators. Our emphasis is to reproduce legacy implementations and perform comparisons for project based applications within a heterogeneous environment of vendor software development platforms, e.g. Xilinx [4], MentorGraphics [5], and operating system versions. Additionally, we require the ability to run several such projects concurrently, with some degree of automation. We have tested the feasibility of our approach by using cost-effective, ESX virtual machine technology from VMWare [6] to record snapshots of the various project environments. The steering of the procedures, themselves, is done using Expect calls [7] in Perl scripts.

FPGA TOOLS

At the present time we are employing the design methodology provided by tools from Xilinx. In the future we plan to add development tools from MentorGraphics to our suite. In this section we briefly summarize these tools in order to motivate the scenarios to implement for the steering process in terms of the design flow for developing custom embedded processor systems, i.e. to 'build' a project.

The Integrated Software Environment (ISE) is the 'corner-stone' for designing FPGA logic from Xilinx. It consists of software development tools and utilities (e.g. constraints entry, timing analysis, logic placement, routing and device programming) grouped together so as to simplify some of the inherent complexity in the design process flow.

The Embedded Development Kit (EDK) is a set of tools as well as the Intellectual Property that enables one to design a complete embedded processor system for implementation in a (Xilinx) FPGA device. It requires the ISE to be installed. One particular tool we mention here from the EDK set is the Xilinx Platform Studio (XPS) which is the development environment for designing the hardware part of the embedded processor system.

In fact these are GUI environments, and one of our core requirements is to 'capture' the sequence of occurrences from the button-clicks in terms of the files being created, manipulated and processed, and to be able to execute the same sequence from their primitives in command line scripts. Figure 1 shows a simplified flow for an embedded design process in the context of the tools described above.

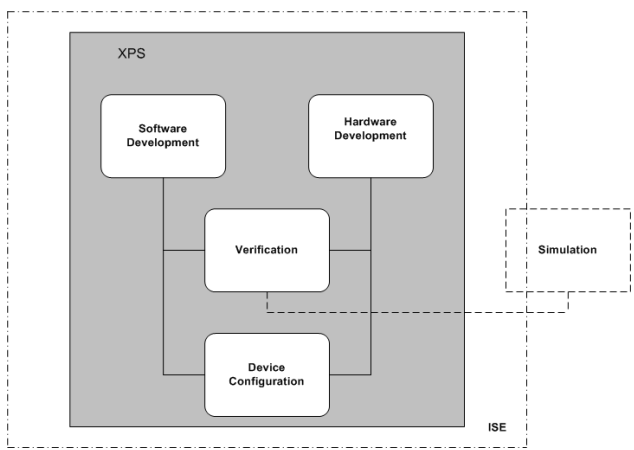

Figure 1: Synoptic view of the Xilinx embedded process.

The recommended design flow by Xilinx is to begin a new project with the ISE, and then to add the embedded processor source to this project. The XPS is used

principally to develop the embedded processor hardware system, where the configuration of the microprocessor, peripherals and interconnects of the components as well as their respective property assignments takes place. Verification of the correct functionality of the hardware platform is accomplished by means of a Hardware Description Language (HDL) simulator in the ISE, and also partially with the XPS.

VIRTUALIZATION

A central requirement for the build environment is the hardware and software infrastructure where snapshots of the system configuration can be saved and retrieved, with the possibility to modify the software installed as a function of time, variants of the host operating system, and third party software package versions, yet allowing the possibility to rollback to a previous configuration with a minimum of complexity and low risk. VMWare's virtual machine (VM) implementation is used for this purpose.

We are using the ESXi 'hypervisor or bear-metal' architecture, which in contrast to the 'hosted' option does not run on top of another operating system, but includes its own kernel. The snapshot feature is a mechanism that allows to preserve a given state of the VMs as a function of time. A snapshot can be taken at any point in time, and to revert back to that status, also at any point in time.

A schematic representation of the infrastructure, with the VM's is illustrated in Figure 2, showing the guest virtual machines (VM_1, VM_2, VM_n) and the administration via the Remote Client (RCLI).

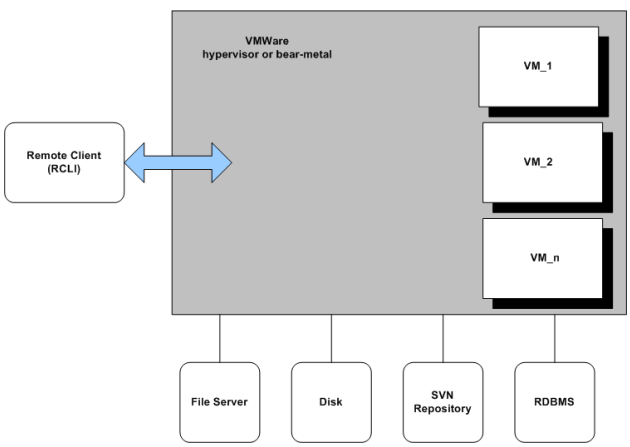

Figure 2: The topology of the virtual machines, RCLI and external infrastructure described in the text.

The File server and disk storage are `afs` directory structures. They can either be the developer's home directory or a common project-related directory. Version control for the projects is done using Subversion (SVN). Access to a Relational Database Management System (RDBMS) is also foreseen. All of the above are available on each of the VMs.

IMPLEMENTATION

While there are advantages to initiate a project and elaborate the development stages in a GUI environment, there exists a constant risk if repeatability depends on remembering to click menus and boxes in the correct sequence in the user interface. In contrast, creating a command line procedure guarantees that the same sequence of commands are issued each time, and is, arguably, better suited to the maintenance part of the project lifecycle.

As an illustrative example, Figure 3 shows the main steps involved in our implementation, as an UML activity diagram, which is generic in nature to the different projects.

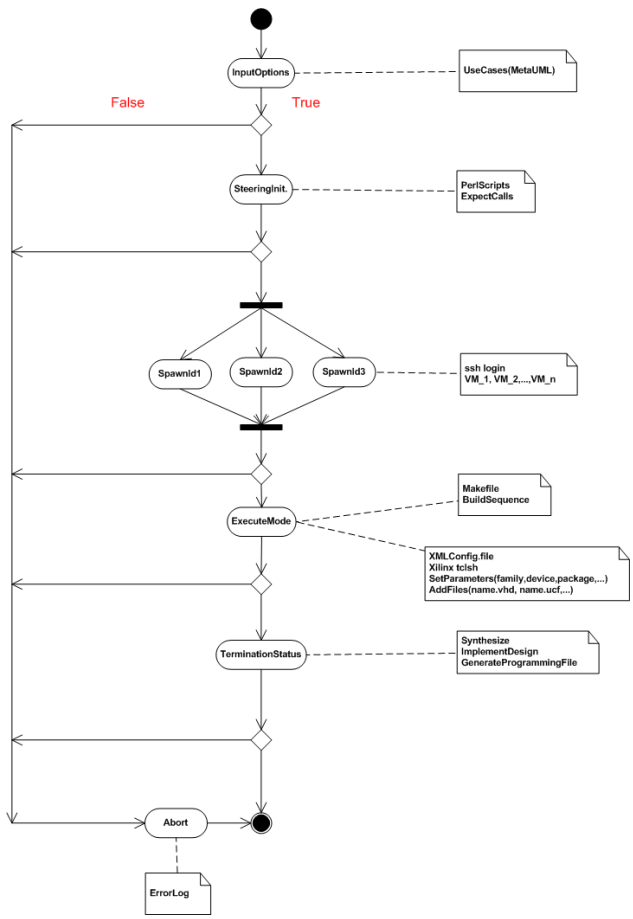

Figure 3: A generic UML activity diagram illustrating the main steps involved for the project 'build' in the virtual machine environment.

The input requirements and options are UML use cases, such as for example, the project location, Xilinx versions and directory structures in the pre-configured SVN repository. For steering of the procedure, from an end-user console, Expect calls are used in Perl scripts. Expect is ideally suited for automating procedures where periodic input is expected from the spawned process. Several

processes can be spawned in parallel, each mapping to a given VM, for building a specific project. The projects themselves, can either be 'self-contained' in terms of executing a `Makefile`, or driven by a sequence of instructions. The keyword tags for the instructions are contained in XML files, in analogy to the conventions used by Xilinx. The scripts iterate through the sequence of the tools, such as: synthesize, implement design and generate programming file. Directories, intermediate and final files are created in the work space of the spawned process as required. These tasks are readily possible thanks to the extensive and well tested Perl library modules.

CONCLUSION

A pragmatic approach has been presented and tested to manage software and firmware at the PSI accelerators, using virtualization technology in conjunction with command line scripts. This enables us to better control our resources across the application lifecycles, in an environment of rapidly changing vendor toolsets and platforms.

We believe that future requirements can be integrated with relative ease within this framework in view of its modular and flexible architecture.

ACKNOWLEDGMENTS

It is a pleasure to thank Rene Kapeller for his assistance in the installation and troubleshooting of the VMWare virtual machines infrastructure.

REFERENCES

[1] Contributions to the Topical Workshop on Electronics for Particle Physics, TWEPP-07 (2007); http://www.particle.cz/conference/twepp07.
[2] J. Carwardine et al., "The ILC Control System", ICALEPCS07, (2007); http://neutrons.ornl.gov/conf/icalepcs07.
[3] J. B. Lister et al., "The ITER CODAC conceptual design", Fusion Engineering and Design 82 (2007) 1167-1173.
[4] Xilinx, Inc.; http://www.xilinx.com.
[5] Mentor Graphics Corporation; http://www.mentor.com.
[6] VMWare, Inc.; http://www.vmware.com.
[7] D. Libes, "Exploring Expect", O'Reilly & Associates, Inc. (1995).