Contribution ID: **61** Type: **Oral**

A collaborative HDL management tool for ATLAS L1Calo upgrades

Thursday, 20 September 2018 14:25 (25 minutes)

Coordinating firmware development among many international collaborators is becoming a very widespread problem in particle physics. Guaranteeing firmware synthesis with P&R reproducibility and assuring traceability of binary files is paramount. Our HDL managing tool tackles these issues by exploiting advanced Git features and being deeply integrated with HDL IDE, with particular attention to Intellectual Properties handling. In LHC Run3, the ATLAS L1Calo Trigger system will be upgraded with new feature extraction and readout modules. Our tool, handling firmware development for these modules, was developed in Python and integrated with CERN Gitlab (using Web-Hooks, Gitlab API) and Xilinx Vivado.

Summary

We devised a set of Tcl scripts and a suitable methodology to allow a fruitful use of Git as a firmware repository and guarantee synthesis reproducibility and binary file traceability. One repository hosts many projects sharing a significant fraction of code. For example, many FPGAs on one board, as in the case of eFEX (1 Control FPGAs, 4 processing FPGAs) or FTM (1 control FPGA, 2 DSS FPGA). Tcl scripts, able to recreate the projects are committed to the repository, as recommended by Xilinx [1]. This permits the build to be Vivado-version independent and ensures that all the modifications done to the project (synthesis/implementation strategies, new files, settings) are propagated to the repository, allowing reproducibility. In order to make the system more user friendly, all the source files used in each project are listed in special list files, together with properties (such as VHDL 2008 compatibility) that are read out by the project Tcl file and imported into the project as different libraries, helping readability. Being written in Tcl and thanks to its versatility, this system is extensible to other FPGA IDEs, e.g. Altera Quartus.
To guarantee binary file traceability, we link it permanently to a specific git commit. Thus, the git-commit hash (SHA) is embedded into the binary file via VHDL generic and stored into firmware registers. This is done by means of a pre-synthesis script which interacts with the git repository. Both the project creation script and the pre/post synthesis scripts are written in Tcl (compatible with Xilnx and Altera) and make use of a utility library designed for this purpose, including functions to handle git, parse tags, read list files, etc.
An automatic synthesis and implementation system, called Automatic Workflow Engine (AWE), has been devised to build official binary files and verify merge requests (MR) before accepting them, to avoid polluting the official branch undermining the starting point for other developers. AWE is written in Python and runs on CERN Openstack virtual machines. It interacts with Gitlab through web hooks and reacts to MR events. It can parse MR parameters, allowing the specification of directives through special keywords in the MR title/description on Gitlab website. When a MR is opened/updated, AWE sets up the repository and launches a complete firmware workflow producing binary files. If any hindrance is encountered in the process, AWE reports it by writing notes on Gitlab website through the Gitlab API.
If the workflow is successful, AWE approves the MR. When the MR is merged (by human intervention), AWE tags the new official version and fills release note with timing/utilisation reports.
To ensure binary file traceability, AWE preserves git SHA after the MR is merged to the official branch.
A semantic versioning system x.y.z is used and embedded into git tags. AWE is designed to automatically increase the version when a new MR is opened which happens before the workflow is started in order to embed the version number into the binary file.

[1] Xilinx, XAPP1165 (v1.0) August 5, 2013

Primary author: MASIK, Jiri (University of Manchester (GB))

Presenter: GONNELLA, Francesco (University of Birmingham (GB))

Session Classification: Programmable Logic, Design Tools and Methods

Track Classification: Programmable Logic, Design Tools and Methods