ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-M04A003

THE DESY OPEN SOURCE FPGA FRAMEWORK

L. Butkowski*, A. Bellandi, B. Dursun, C. Gümüs, K. Schulz,
M. Büchler, N. Omidsajedi, Deutsches Elektronen-Synchrotron DESY, Hamburg, Germany

Abstract

Modern FPGA firmware development involves integrating various intellectual properties (IP), modules written in hardware description languages (HDL), high-level synthesis (HLS), and software/hardware CPUs with embedded Linux or bare-metal applications. This process may involve multiple tools from the same or different vendors, making it complex and challenging. Additionally, scientific institutions such as DESY require long-term maintenance and reproducibility for designs that may involve multiple developers, further complicating the process. To address these challenges, we have developed an open-source FPGA firmware framework (FWK) at DESY that streamlines development, facilitates collaboration, and reduces complexity. The FWK achieves this by providing an abstraction layer, a defined structure, and guidelines to create big FPGA designs with ease. FWK also generates documentation and address maps necessary for high-level software frameworks like ChimeraTK. This paper presents an overview and the idea of the FWK.

INTRODUCTION

At the scientific institute DESY [1], Field-Programmable Gate Arrays (FPGAs) have become indispensable components within a wide array of control and diagnostic systems. The adoption of FPGAs in these applications is primarily driven by their exceptional multichannel computation power and unparalleled flexibility. However, the FPGA firmware development process at DESY presents a set of significant challenges. The institute's facilities, including EuXFEL and FLASH, demand long-term support and maintenance, spanning up to 20 years [2]. Over such extended periods, numerous features evolve or undergo modifications, toolchains are updated or replaced, and hardware experiences changes or upgrades. What further complicates the process is the involvement of multiple developers and collaborations, with projects transitioning between responsible persons.

These firmware development challenges coincide with the management of multiple projects, often handled by small teams, and run in parallel with a continuous stream of new developments and rapid prototyping efforts. To further complicate matters, the most recent developments encompass a convergence of multiple technologies, entailing the integration of code written in Hardware Description Language (HDL), High-Level Synthesis (HLS), Embedded C/C++, and Embedded Linux into a unified and coherent design.

To address these challenges of FPGA firmware development head-on, a dedicated firmware framework has been developed. This framework is an abstraction layer that carries a set of rules, scripts, functions, and procedures that are universal and reusable. The initial framework idea is presented in Figure 1. This framework has been designed to tackle a wide range of issues:

- Universal support for multiple vendor tools through an abstraction layer.
- Consistent firmware construction for a unified approach across projects.
- Reproducible builds for enhanced stability and reliability.
- Code reusability for accelerated development and reduced redundancy.
- Collaboration support for multiple developers working on the same project.
- Faster adaptation to new hardware developments.
- Streamlined application deployment on ready hardware platforms.
- Efficient configuration management for easy customization and flexibility.
- Simplified address space management and integration with ChimeraTK framework [3].
- Automatically generating the technical and user documentation.
- Integrated design verification capabilities.
- Quality assurance enhancements through automation in versioning and builds, ready for continuous integration.
- Ability to combine multiple technologies such as HDL, HLS, Embedded C and Linux into one design.

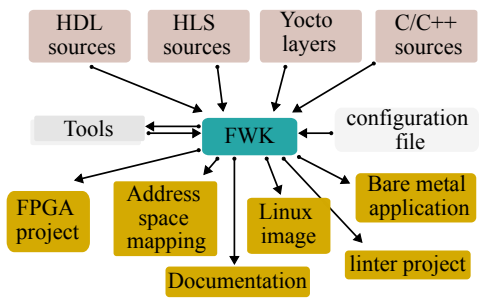

Figure 1: Framework concept.

The initial iteration of the firmware framework was created in 2013 for MTCA.4 systems at EuXFEL [4]. Initially, the framework was closely integrated with the code in a mono repository, leading to challenges related to sharing and scalability as project numbers grew. To address these issues, the decision was taken to decouple the framework from the code, restructure it, and release it as open source. This transition was motivated by the aim of supporting other institutions and facilitating collaboration through open sourcing [5].

* lukasz.butkowski@desy.de

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-MO4A003

GENERAL STRUCTURE

In the firmware framework, there are four major components:

- Framework module
- Source modules
- Main project
- Vendor tools

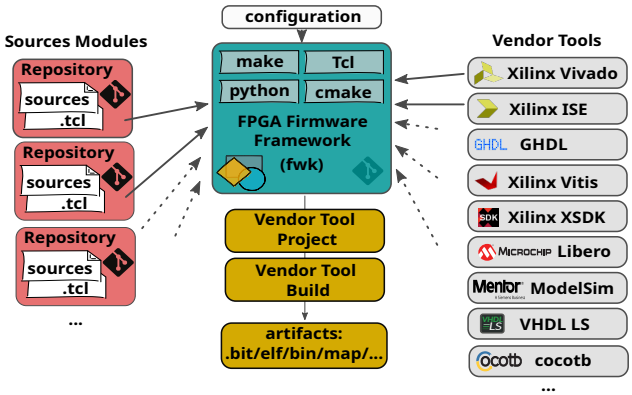

Figure 2: FPGA firmware framework major components.

The framework is a set of scripts, functions and procedures which combine all the input files needed to produce a build. It is added to the project and becomes its integral part.

Source modules are blocks of functionality that can be decoupled and abstracted from the rest of the project. Each module can have its own address space and can be shared among various projects. A module can contain code written in HDL, HLS, C/C++, or Yocto layers. Modules have a defined folder structure which is described in **Folder Structure** section.

The main project is where the modules come together to produce an output. The project can realize a complete built of FPGA bit file, run a simulation environment or build a Linux image. It also contains necessary scripts for project creation, build or simulation along with the documentation. The project has a defined structure which is described in **Folder Structure** section.

Vendor tools are executables which are used to generate artifacts from the project sources. They are not integral part of the framework. Tools should be installed system wide or in a virtual environment and accessible by the FWK scripts.

All the components are presented in Figure 2.

For the framework, it is requires that each of the source module and the project provide a configuration file with a defined interface and variables set.

The framework utilizes the following technologies:

- GNU make as the entry point.
- Tcl as the main framework scripting language.
- Tcl as the source module configuration files.
- Python as the auxiliary supporting tools.
- Cmake supporting makfiles generation.

Configuration Files

The main project is configured over environment variables set in configuration file. Over these environment variables you may set values such as the project name, project configuration name or default vendor tool. Each project can have multiple configuration files. By selecting configuration file you can change the project creation or build behavior.

Tcl Scripts

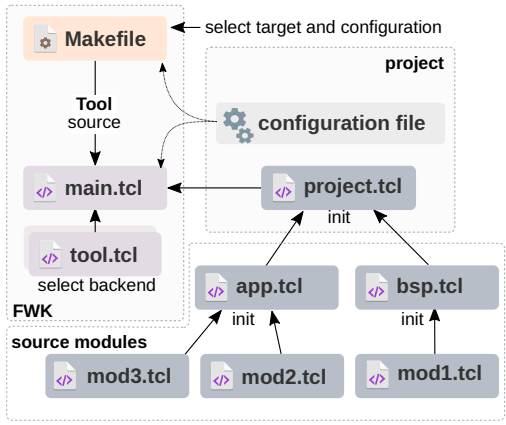

Figure 3: FWK Tcl scripts flow.

For the framework, the Tcl scripting language was chosen. Tcl is a high-level, general-purpose, interpreted, dynamic programming language intended to be embedded into applications. Most of the vendor HDL tools can be driven by Tcl commands with a built-in Tcl interpreter. This makes it possible to integrate the framework into the vendor tool, and all the FWK commands can be run from the tool's command line Tcl console.

Tcl scripts have been chosen as well as the configuration file for the source modules. It has advantages over static configuration files such as YAML or JSON. Unlike these, Tcl gives the possibility to use conditional statements, loops, or run a script. This allows to configure the source modules dependently on the configuration settings or the vendor tools used.

The firmware Tcl file structure is organized in detail. In this system, every source module possesses its dedicated Tcl script, a design that promotes modularity and clarity. The script's structure follows the hooks concept, ensuring that each module's functionality seamlessly integrates into the framework. To maintain consistency and interoperability, all Tcl scripts mandate the presence of standard procedures.

Within this framework, each project and source module must have a dedicated Tcl file containing the following essential procedures:

- **init** : Executed at the project's outset to initialize the project's structure, hierarchy, and configuration variables. It also sets dependencies on source modules.
- **setSources** : Invoked to configure variables with source file information.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-M04A003

- **setAddress** : Responsible for defining address space information, including name, address, and the definition file.
- **doOnCreate** : Triggered upon project creation by the vendor tool backend, facilitating the setting of project properties and sources.
- **doOnBuild** : Executed just before the project build, also by the vendor tool backend, offering the opportunity to configure build-specific properties or regenerate files.
- **setSim** : Active during simulation, this procedure sets simulation environment variables.

The framework abstracts all vendor tool commands and procedures for project management, including source additions and build processes, providing a unified set of Tcl procedures. However, it retains the flexibility to conditionally execute vendor tool-specific procedures.

The processing of module Tcl files commences with the iterative execution of the 'init' procedures. These procedures, belonging to each module, are organized within separate nested namespaces, forming a dependency tree among the modules. With this tree structure in place, all other procedures are executed recursively throughout the tree, with execution proceeding from the last child to the top parent. As presented in Figure 3, the execution initiates with the Makefile, along with specified targets and configuration files. Based on this information, the appropriate vendor tool backend is selected, and the project Tcl file is activated. The example Tcl namespaces created are presented in Figure 4.

```
::fwfwk::src::app::mod2::
::fwfwk::src::app::mod3::
::fwfwk::src::bsp::mod1::
::fwfwk::src::
```

Figure 4: Tcl namespaces created for example in Figure 3.

For vendor tools lacking a Tcl interpreter, such as GHDL, a standard Tcl shell is employed to generate the project. The project's structure is abstracted from the developer and typically takes the form of Makefiles, often generated with the assistance of CMake.

Source Code Management

The framework utilizes a source code management (SCM) tool to manage modules and their versions. Each of the main project modules, including sources, framework, and the project itself, is stored in a separate repository. Currently, Git is used as the SCM for the framework, which is placed under a Git repository. The main project serves as a super project, and all modules are treated as Git submodules. We rely on Git to manage versions and track changes. While this approach offers the advantage of using a standardized method, it does limit us to only one SCM type, which could be considered for change in the future.

In principle, all the framework components could work without any SCM, but we would lose all the benefits, such as tracking changes, enabling reproducible builds, and automating version tracking. The framework generates version files based on the information from the SCM and places this information in a way that is accessible in the final build.

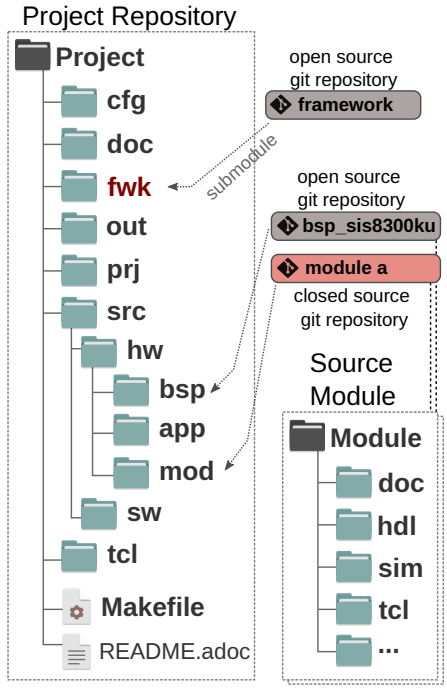

Figure 5: Firmware framework project structure.

Folder Structure

Firmware framework provides a guideline about the folder structure of the projects and the source modules. Applying this guideline is not mandatory but the standardization of the folder structure makes easier to collaborate or maintaining multiple projects. The project and module structure is presented in Figure 5.

The project folders have the following functions:

- **cfg** - contains project configuration files
- **doc** - documentation of the project
- **fwk** - firmware framework folder
- **out** - project build output files, all artifacts go in here (.bit, .map, .xsa etc)
- **prj** - project environment, vendor tool project files, temporary files
- **src** - sources of the project
- **tcl** - project Tcl's script files

If the project consists of hardware and embedded software or Linux components, additional source modules in the 'src' folder should be grouped into 'hw', 'sw', and 'yocto' subfolders, respectively.

The source module folders have the following functions:

- **doc** - documentation of the project
- **hdl (src)** - module sources
- **sim** - test and simulation sources such as tesbenches
- **tcl** - module tcl files

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-M04A003

ADDRESS SPACE

Address space description and generation are essential aspects of FPGA design, with implications for hardware and software integration. The framework simplifies this challenge by collecting address space information, accommodating various formats, and constructing address space trees.

The FWK's role is to collate data about each module's address space, organize hierarchy, and generate output files for hardware and software interaction, as presented in Figure 6. It can adapt to standardized or proprietary formats, making it versatile for integrating legacy and new modules within a single design. FWK creates address trees in Tcl to manage diverse address space formats effectively.

The address space backend, a crucial component, translates the Tcl-based address tree into an actual address space description in a form understood by high-level software frameworks like ChimeraTK or bare-metal applications. Additionally, the address space backend generates documentation files, register-transfer level (RTL) files such as VHDL, or register models and abstractions used in design verification. Currently, there is only one supported backend called DesyRDL, described in **DesyRDL** section.

In modern designs, there is a possibility of having multiple address trees that may not be related or may share the same bus. These address trees can be accessed by various managers, including embedded CPUs, over Ethernet, using direct memory access (DMA), or PCIe. This complexity adds sophistication to the generation of address maps. To address this issue, within the firmware framework, we define what is known as an "access channel." An access channel describes how you can access the address tree and defines the address spaces that are visible from different points.

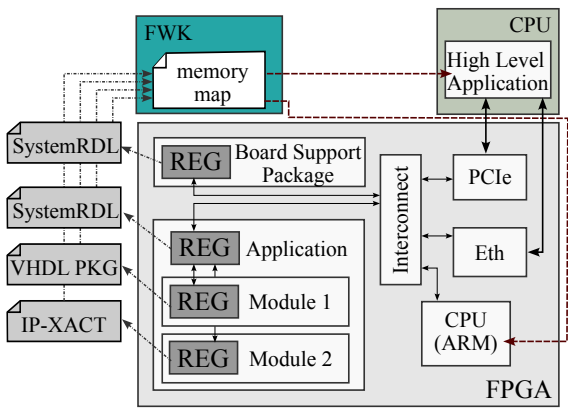

Figure 6: Collecting and compiling of address space by FWK.

Address Space Tree

The example address space tree is presented in Figure 7. Each node has a label, address, address format, and an address definition file or variable specified. FWK simplifies the creation of these trees by providing procedures. As an argument, they accepts an address definition file or another tree which allows the merging of trees and the construction of a proper hierarchy.

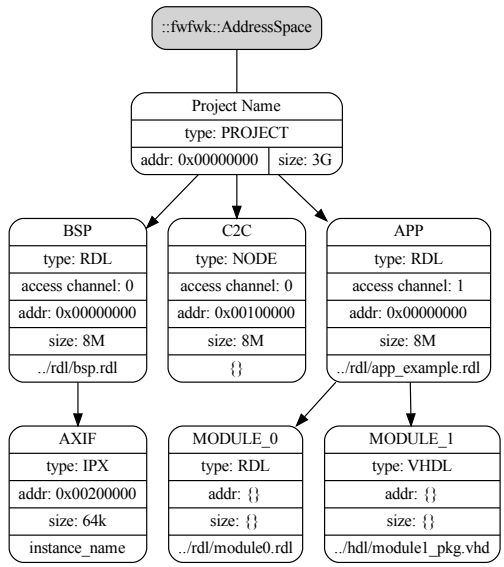

Figure 7: Example address space tree in Tcl.

DesyRDL

DesyRDL is an open-source tool that generates outputs for an address space defined by one or many SystemRDL™ 2.0 [6] input files. It has been written in Python with the use of the open-source systemrdl-compiler [7]. The input files are prepared by the FWK, which also converts all address space formats to SystemRDL. DesyRDL compiles them and produces artifacts such as VHDL code, documentation in the form of markup files, header files, a map file for ChimeraTK, and register models for design simulation and verification. The concept is presented in Figure 8.

DesyRDL, as an address space generation backend, is one of the key components of the firmware framework.

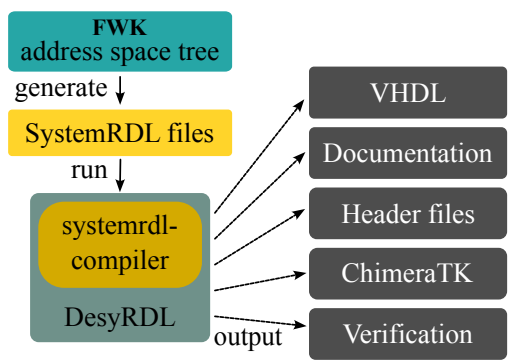

Figure 8: Generation of artifacts by DesyRDL tool.

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-M04A003

DOCUMENTATION

FWK follows the Documentation as Code approach [8]. This means that documentation is kept together with the code in the same repository. Documentation is written in text-based files with a defined syntax. The framework prefers AsciiDoc syntax for documents, as it has integrated tools that generate output based on this format. The Antora framework is used to generate web-based documentation, and Asciidoctor is used for PDF generation. The role of FWK is to gather module documentation, generate register documentation, and execute documentation backend tools, as presented in Figure 9.

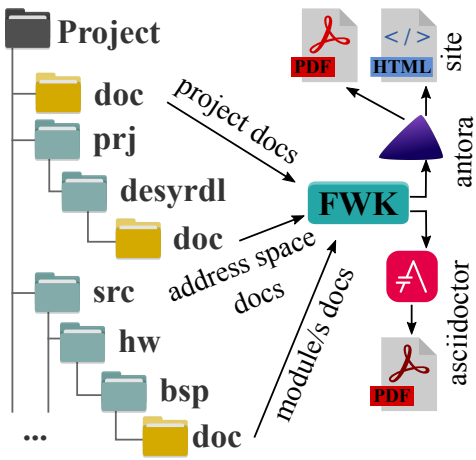

Figure 9: Documentation generation flow.

VENDOR TOOLS

At the time of this writing, FWK supports a number of vendor tools. The list is presented in Table 1.

Table 1: FWK Supported Vendor Tools

Tool Name	Type
Xilinx Vivado	Synthesis/Simulation
Xilinx ISE	Synthesis
Xilinx PlanAhead	Synthesis
Xilinx Vitis HLS	Synthesis
Xilinx Vitis	Processor support
Xilinx XSDK	Processor support
Xilinx Appguru	Processor support
Microchip Libero	Synthesis
Siemens ModelSim	Simulation
cocotb	Simulation
GDHL	Simulation/Synthesis/ Language Server
VHDL LS	Language Server
VHDL Tool	Language Server
bitbake	Yocto Embeded Linux

EXECUTION FLOW

As described in the introduction, the framework unifies the execution flow regardless of the vendor tool used. In Figure 10 there is a common flow presented that is used to regenerate, build, simulate and analyze the project.

```
# clone repo
git clone git@ ...repo
cd repo
# clone all submodules repositories
git submodules update --init --recursive
# create virtual environment with tools
make env
# create project
make cfg=example1 project
# build the project -> generate artifacts
make cfg=example1 build
# build documentation
make cfg=example1 doc
# run project simulation
make cfg=example1 sim
# open tool gui for design analysis/debug
make cfg=example1 gui
```

Figure 10: FWK common flow.

RESULTS

The framework's effectiveness has been demonstrated not only within its birthplace, the MSK group at DESY, but also in various other groups and institutes. Within the MSK group, where it originated, it empowered teams ranging from 3 to 7 members to efficiently manage approximately 40 projects, each with multiple configurations across 15 distinct hardware setups. This framework also facilitated the contributions of up to 45 developers to the codebase, leading to remarkable enhancements in collaboration, code quality, and change traceability.

The advantages of implementing this framework are extensive:

- Significant reduction in overall development time.
- Heightened system maintainability.
- Streamlined collaborative development process.
- Enhanced quality control practices.
- Simplified management of multiple projects.
- Reducted complexity through well-defined abstraction layers.

However, it is imperative to acknowledge the associated challenges:

- Initial investment of time and effort to learn the framework.
- Adherence to specific rules, potentially limiting flexibility.
- Commitment of resources for ongoing framework development and maintenance.

While the benefits of utilizing this framework outweigh its drawbacks, it is important to recognize that the initial

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-MO4A003

cost can be substantial.

Additionally, open-sourcing the code offers advantages, such as improved collaboration by minimizing the issues with code sharing, and broader feedback and review. Nevertheless, this endeavor is not without its own resource demands, including supporting external users, enhancing documentation, ensuring backward compatibility, and overall code refinement.

Balancing these considerations is essential in making an informed decision regarding the implementation of the framework.

CONCLUSIONS

In conclusion, DESY's open-source FPGA firmware framework (FWK) tackles the challenges of FPGA firmware development by providing a structured and unified approach. It offers an abstraction layer, ensures consistency, and supports multiple technologies, including HDL, HLS, Embedded C/C++, and Embedded Linux. This framework streamlines collaboration, enhances reproducibility, and simplifies address space management, making it a valuable tool for FPGA developers. However, while it significantly reduces development time and improves maintainability, it requires an initial learning curve and ongoing maintenance commitment, which should be carefully considered when implementing the FWK.

Overall, DESY's FWK represents a crucial advancement in FPGA firmware development, benefiting scientific research institutions and other projects by providing a well-defined framework for efficient and reliable FPGA design.

REFERENCES

[1] DESY, *Deutches Elektronen-Synchrotron*, [Online; accessed September-2023]. https://www.desy.de

[2] M. Altarelli, R. Brinkmann, and M. Chergui, *The European X-Ray Free-Electron Laser. Technical design report*. DESY XFEL Project Group, 2007.

[3] G. Varghese *et al.*, "Chimeratk-a software tool kit for control applications", *IPAC17, Copenhagen, Denmark*, 2017.

[4] Ł. Butkowski, T. Kozak, P. Prędki, R. Rybaniec, and B. Yang, "FPGA Firmware Framework for MTCA.4 AMC Modules", in *Proc. of International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS'15), Melbourne, Australia, 17-23 October 2015*, Melbourne, Australia, 2015, pp. 876–880. doi:doi:10.18429/JACoW-ICALEPCS2015-WEPGF074

[5] DESY, *FPGA firmware framework*, [Online; accessed September-2023]. https://gitlab.desy.de/fpgafw/fwk

[6] Aaccellera, *SystemRDL 2.0 register description language*, [Online; accessed September-2023]. https://www.accellera.org/images/downloads/standards/systemrdl/SystemRDL_2.0_Jan2018.pdf

[7] A. Mykyta, *SystemRDL 2.0 language compiler front-end*, [Online; accessed September-2023]. https://systemrdl-compiler.readthedocs.io/en/stable/

[8] E. Holscher, *Docs as code*, [Online; accessed September-2023]. https://www.writethedocs.org/guide/docs-as-code/