FPGA DEVELOPMENT APPROACH FOR ACCELERATOR SYSTEMS WITH HIGH INTEGRATION COMPLEXITY

J. Dedic, K. Zagar, COBIK, Solkan, Slovenia*
A. Söderqvist, N. Claesson, R. Tavcar, Cosylab, Ljubljana, Slovenia
J. N. Rodrigues, Lund University, Lund, Sweden

Abstract

This paper presents a Field Programmable Gate Array (FPGA) development workflow for custom hardware as part of an accelerator control systems. The workflow is narrowed down to three equally important parts: requirements analysis, implementation and testing. Each part is presented with milestones and guidelines, which addresses subjects such as knowledge alignment and development standards. These are presented and have been proven to increase efficiency and quality of control systems.

INTRODUCTION

The increasing demand for high power and high availability particle accelerators has put high performance requirements on the control system (CS). With higher speeds and bigger accelerators the CS grows bigger, up to thousands of distributed nodes, and the requirements on synchronization and security are raised.

Field Programmable Gate Arrays (FPGAs) have become ubiquitous in CSs for accelerators. An essential part of the CS is the Timing System (TS), which coordinates the operation of the devices around an accelerator. The TS can be seen as the backbone and a service that is delivered to the CS.

A complete system overview, which means extensive iteration between all stakeholders, is required before the development of a TS can be started. Development standards need to be in place and continuous integration is also needed.

This paper is divided into four parts. First a generic workflow is described, later an example of knowledge alignment is given, followed by some shortly depicted production projects. In the end a conclusion is also given. In total, this is the necessary knowledge needed for efficient FPGA development for CSs.

WORKFLOW

The FPGA development workflow is divided in to three parts. Each part is approximated to one third of the total effort.

Requirement analysis

Before starting implementation of a timing system preparations to learn the interplay between all the machines are required. Key steps involved are:

1. Assemble all the nodes around the accelerator that require real-time timing information.
2. Understand all the nodes' requirements in accordance with the physical machinery.
3. Define clean interfaces and functions for all the nodes.

Defining interfaces and functions needs extensive iteration between all stakeholders involved, including physicists, electronics and software developers. Unfortunately, the implementation process often starts early, with insufficient knowledge from all three steps. This leads to a system where one part is incompatible with another. However, due to the amount of invested resources all parties tend to stick with it. It is therefore crucial that all requirements are collected and an agreement between all stakeholders is reached before implementation starts. Preparations for implementation also need to begin during the requirement analysis.

Knowledge and investigation of current technologies are necessary. Off-the-shelf products and standard components that minimizes development effort are preferred. They shorten development time as well as guarantee availability for spare parts, making upgrades and repairs easier.

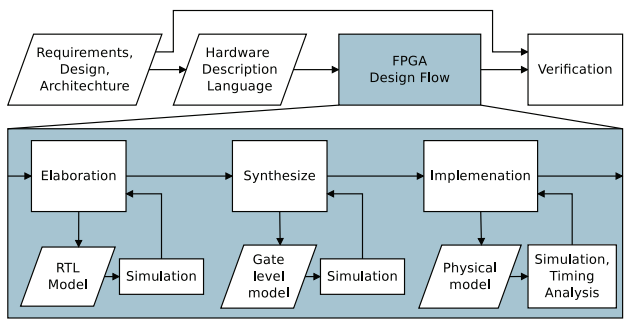

Figure 1: Automated build and verification flow for fast adaptation to changes in requirements or design.

Implementation

The requirements need to be analyzed before Register Transfer Logic (RTL) coding starts. Basic and common functionalities need to be distinguished in order to partition the design. A proper partitioning leads to a generic and scalable solution. This means that it fits the need of all possible variations of the distributed nodes, minimizing customization effort, a prerequisite to enable small design changes without breaking the overall structure.

Requirements, even though they have been iterated multiple times between affected stakeholders, have a tendency

* {joze.dedic, klemen.zagar}@cosylab.com

ISBN 978-3-95450-122-9

to change, due to the high complexity. For that reason being able to make design changes quickly during the implementation is of high importance. Therefore it is important to set up a build flow, seen in figure (1), for individual components, groups of components and the complete system when RTL coding starts. The build flow is ideally automated and helps integration and verification.

Elaborating and synthesizing individual components saves time and can be done continuously to enable quick adaptation to specific requirements.

While integration of the complete system enforces interface matching, detects compatibility issues and enables physical estimations. Synthesizing the complete system gives an amount of used FPGA primitives and area. Implementation matches the netlist from synthesis with physical constraints constructing a physical model. This model enables timing and power estimations.

Setting up the build flow requires a version control system. It can give access to the latest working and tested version of a component and enables integration between different components as early as possible. This together with the fundamental idea of security and version tracking makes it necessary to use.

Compatibility between developers is acquired through standardizing coding styles and introducing naming conventions. Cosylab uses the same as CERN [1] which enhances developers readability of code developed elsewhere. Using a common coding style enables building of a component library of reliable and reusable components. Software tools have been developed in-house for automatic check of adherence to the agreed conventions. These agreements minimize trivial mistakes and make it easier to focus on the important mistakes during peer code review.

Testing

A structured approach is needed for effective and comprehensive testing. The unit under test should go through following test steps:

1. Simple testbench is constructed to verify the core functionality, few test cases for quick testing.
2. Construct test pattern generator with predictable test patterns, verifiable at output, to extend test cases.
3. Expand testing hierarchically upwards repeating the test steps on bigger part of the system.

The testbench is responsible for generating input and verifies functionality either by using a given golden output or generating a golden output with a model implementation in software. A schematic view over testbenches can be seen in Figure 2 and 3.

The testbench creates various kinds of input, both expected and unexpected, and validate functionality for a predefined amount of test cases. This enables automatic extensive testing and validation after design changes. Design changes emerge because of functionality and interface changes.

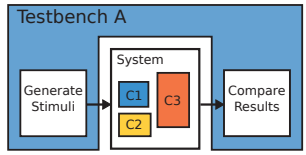

Figure 2: Testbench for complete system.

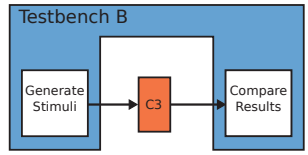

Figure 3: Testbench for single component.

It is necessary to develop a behavioural software model for larger components, or groups of components, in the system to verify the hardware implementations functionality. The generated input is applied to both the implemented hardware module and the behavioural software model and the design is validated by comparing their outputs.

FPGA ACADEMY

A fundamental difference exists between writing source code for hardware and writing it for software. Everything in hardware executes in parallel instead of in sequence, which means that knowledge about software programming usually complicates instead of assists.

The FPGA Academy was invented to overcome this knowledge gap. It is a mean for bringing new developers quickly to production quality on HDL design. A set of exercises is provided to ensure that all developers have the same essential know-how. All necessary steps in the design flow for FPGAs are undergone, including proper partitioning and code readability. The source code is also reviewed by a senior developer and corrected accordingly.

Early design choices have great impact on important results such as the area usage and the highest possible frequency. Designing a production system is therefore not a suitable first assignment. The best choices can only be made with a lot of practical experience which, for example, is gained through working with FPGAs.

Knowing all the tools for FPGA development is also difficult and merely study them is not enough. To fully understand what they do and how to use them they have to be utilized first to some extent.

Since the FPGA Academy forces the developers to use all the tools and to understand how their code performs it is a great way for patching developers missing knowledge.

PROJECTS

Cosylab has been involved in multiple big science projects, such as particle accelerators and sensor arrays. Many of them require different approaches but usually they have significant similarities. A solid understanding of how to realize projects of varying magnitude has been acquired thanks to several successful collaborations.

The following are three examples of recent projects where FPGA development has been involved. At *Spallation Neutron Source* (SNS) [2] an upgrade was supplied, at *MedAustron* [3] and *European Spallation Source* (ESS) the complete architecture was designed and implemented.

ISBN 978-3-95450-122-9

Spallation Neutron Source

Cosylab's contribution to the *Spallation Neutron Source* was a redesign of the TS and an upgraded Timing System Master. The current hardware was ageing and spare parts became scarcer. Calibrating and replacing hardware was tedious labour, thus the accelerator also had undesired downtime.

All the requirements from the onsite teams were re-evaluated to reach an understanding of how to implement them on the new hardware. Finally the software control loop algorithms were implemented on the FPGA firmware.

MedAustron

In the MedAustron project Cosylab was involved from the beginning. It is a medical and experimental facility located in Austria. Some key features of its control system are:

- Possibility to have several virtual accelerators for concurrent testing and commissioning of independent parts.
- Change beam cycle configuration in soft real-time without downtime.
- Accurate and flexible event distribution system with 100 nanosecond GPS timestamping capabilities.

To facilitate these features extensive customization of MRF firmware was required.

European Spallation Source

ESS will be the brightest and most modern neutron source in the world and will have features which put unique requirements on the TS. Some key features of ESS TS are:

- Real-time data transmission from the Timing System Master to the Timing Receivers without affecting the regular messages that control the equipment in hard real-time.
- Send heartbeat events with a frequency of 14 Hz and only initiate sequences directly subsequent the heartbeat message.
- Enable timestamping for all input and output for post-mortem analysis.

The *Timing System Prototype* is based on off-the-shelf products from MRF with considerable firmware customization.

Summary

The CS and hence the TS is often underestimated and do not get enough attention until most of the other decisions are made. This results in inappropriate requirements and more expensive solutions. By developing the CS from the start there is a stronger possibility that it works as desired.

Valuable lessons and insights have been gained from each project. Being able to identify the deliverables and deliver at deadlines and within budget is important since it reduces costs.

CONCLUSION

This approach makes sure that the Control System is designed as intended. It also ensures that the requirements are real and not conceived out of nothing. By assuring that the overall knowledge of the system is known before implementation nothing is under or over specified. This leads to less time spent on unwanted workarounds and on implementing unnecessary requirements. By defining clear requirements it is also apparent when the Control System is ready for use in production.

Following development standards in the hardware team, acquired from internal training, gives a lot of advantages. Such as increased mobility and flexibility of the developers, e.g. moving between projects and sharing code. These abilities help to utilize the whole team. As soon as a project wraps up some can start on a new, while the others join later. It also eases documentation of the system and makes it easier to add features that are requested at a later stage.

Continuously integrating the system validates the interfaces and the current implemented functionality. Significant indications about the system always emerge during the integration. All this knowledge will speed up the final integration when the complete system is set-up. By having automated tests for all hierarchical levels it is also easy to verify the system, e.g. after changes in requirements occur and have been realized.

Cosylab has been and is involved in the construction of many control systems at multiple accelerators thereof SNS, MedAustron and ESS. Following this workflow, the best practices mentioned and by emphasizing the importance of requirement analysis many projects has been brought to successful completion.

REFERENCES

[1] P. Loschmidt, N. Simanić, C. Prados, P. Alvarez and J. Serrano, "Guidelines for VHDL Coding", *http://www.ohwr.org/projects/hdl-core-lib/documents*, April 2011.

[2] D. Thompson, D. Curry and J. Dedic, "Timing system update for SNS", Proceedings of ICALEPCS2009, Kobe, *Japan*, 2009.

[3] J. Gutleber, A. Brett, M. Marchhart and J. Dedic, "The MedAustron accelerator control system", Proceedings of ICALEPCS2011, Grenoble, France, 2011.

[4] J. Dedic, M. Plesko, R. Sabjan, I. Verstovsek and K. Zagar, "Control systems for new large experiments", Proceedings of PCaPAC 2010, Saskatoon, Saskatchewan, 2010.

ISBN 978-3-95450-122-9