**Title**

HW-SW SystemC Co-Simulation SoC Validation Platform

**Design Flow Report**

Contract No. 4200022968

by

 TU Braunschweig, Germany

March 14, 2012

**Document No.: IDA-SCSV-DF-001**

**Table of Contents**

1 Introduction 2

1.1 Purpose & Scope 2

1.2 Revisions 2

2 Socrocket design Flow 3

2.1 Overview 3

2.2 Setting up a platform template 3

2.3 Configuring the template 4

2.4 Generating platform simulation and software 4

2.5 Running/Debugging simulation 4

2.6 Analysis of simulation results 4

**Table of Figures**

Figure 1 - SoCRocket Design Flow 3

**Table of Tables**

**Es konnten keine Einträge für ein Abbildungsverzeichnis gefunden werden.**  
Wählen Sie im Dokument die Wörter aus, die in das Inhaltsverzeichnis eingeschlossen werden sollen, und klicken Sie dann in der Formatierungspalette unter "Formatvorlagen" auf ein Überschriftenformat. Wiederholen Sie dies für alle einzuschließenden Überschriften, und fügen Sie das Inhaltsverzeichnis dann in das Dokument ein. Sie können ein Inhaltsverzeichnis auch erstellen, indem Sie auf die Option "Manuelle Formatierung" klicken und die Einträge dann manuell eingeben.

# Introduction

## Purpose & Scope

This document is the Design Flow Report (DFR) of the SystemC Co-Simulation SoC Validation Platform (SoCRocket). It describes the natural way of assembling, building, executing and analyzing a platform simulation using the infrastructure of the library. The document walks the user through all these steps at the hand of an example design.

The DFR should be read in conjunction with the Interconnect Methodology Summary (IMS) and the Analysis Capability Report (AR). The IMS describes all the available interconnect options, and the AR the analysis API along the various parameters and performance counter for Design Space Exploration.

## Revisions

The following table will be updated during the course of the project.

|  |  |  |
| --- | --- | --- |
| **Version** | **Date** | **Description** |
| 0.1 | 21/02/12 | Initial draft version |
|  |  |  |
|  |  |  |
|  |  |  |

# Socrocket design Flow

## Overview

The SoCRocket Design Flow (DF) () provides a method for designing TL system models of embedded hardware for various applications. Primary use cases are early software development and design space exploration. Flow and tools have been particularly developed for the aerospace domain, but are by no means restricted to it.

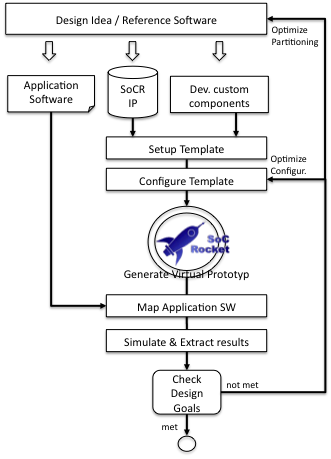


Figure - SoCRocket Design Flow

The flow typically starts from a piece of reference software implementing a design idea. In the domain of embedded computing the software is usually written in C/C++ language. Although, this is not a requirement, and any other language may be selected, it is recommended to use C/C++, because all simulation models of the SoCRocket library are written in SystemC, which is an extension to the C++ standard.

In the first step of the flow the reference software is segmented to create an initial partitioning. The partitioning represents the decision, which parts of the reference software will become software running on one of the target processors and which will become dedicated hardware. Finding the right partitioning for a system is a complex task and usually requires multiple iterations.

For assembling the initial system the SoCRocket library provides a set of simulation IPs representing core components from the Aeroflex Gaisler GRLIB hardware library. This includes a LEON instruction set simulator (ISS), AHB/APB interconnect and peripherals. The library contains SpaceWire and SoCWire models, and is constantly extended. The designer may introduce own custom components, e.g. hardware accelerators for crucial parts of the code. Guidelines for creating and integrating own simulation models can be found in the SoCRocket User Manual.

In the next step the structure of the system must be captured in a template. In the given context a template consists of a SystemC top-level file containing an implementation of sc\_main and a descriptor file (.tpa). To facilitate design space explorations the template may contain an arbitrary number of variables representing constructor parameters or design options (e.g. cache size or number of CPUs). To hook up with infrastructure and tools, these variables must be exported. This is done in the descriptor. The descriptor is an XML file specifying all parameter, their default value and range. The library provides two predefined platform templates (see ./templates dir). More information about the structure and syntax of templates is given in section 2.2.

To compile and run a platform simulation the parameters exposed by the template must be properly initialized. For very simple systems this can be done manually. More comfortable means are provided by the SoCRocket Platform Wizard. The Wizard parses the template descriptor (tpa) and displays all available parameters in a graphical user interface. The parameters are shown along their descriptions and defaults. The user can adapt the settings, store the configuration and generate a system simulation at the touch of a button. Configurations and Platform Wizard are explained in section 2.3.

Running platform simulation still requires mapping the software portions of the design. Generating the respective memory images depends on the system memory map. The Platform Wizard interprets the memory map, generates appropriate scripts for linking and integrates the newly generated platform in the build environment (section 2.4). The build system compiles the platform into an executable simulator. It is recommended to pass name and location of the memory images (ELF files) as command line parameters. How to run and debug simulation is explained in section 2.5.

During simulation the platform infrastructure records execution statistics. For this purpose every simulation model of the library provides sets of performance counters. In the default configuration the performance counters are written to the terminal after the end of the simulation. It is also possible to trace counters in log or waveform files, and to register handles for certain bounds and events. This document only provides an example for using the analysis features of the platform (section 2.6). All available options are described in the Analysis Capability Report.

The results of the analysis step are supposed to support the verification of the design goals. The designer usually aims for requirements such as throughput and latency at the lowest possible cost. Thereby costs typically are silicon area or power consumption. Correct prediction of implementation costs is not completely possible at transaction level, because target technology and synthesis strategy have a big impact on the final result. Nevertheless, the TL model is capable of delivering a strong indicator (see SoCRocket Power Modeling Report). In terms of timing accuracy the AT-mode of the provided models has proven to be very close to results from RTL simulation (see IP Verification and Performance Report). If the benchmarks are positive and further optimizations do not appear promizing the actual hardware implementation may start. Otherwise the designer can decide to modify parameters (optimize configuration), or alter the partitioning, and try again.

Next to the Approximately Timed AT-mode all models of the SoCRocket library provide a Loosely-Timed LT-mode for fast register/address accurate simulation. This mode is intended to speed-up the software mapping. It can also be used to optimize the software while hardware implementation is in progress, or to develop additional software after hardware is completed.

The remainder of this documents guides the user through all phases of the design flow at the example of a standard single-core LEON design. All discussed code can be found in the ./templates and the ./software directories of the library.

## LEON FFT processor

### Setting up a platform template

All simulation models of the SoCRocket library can be manually instantiated. Example instantiations are shown at the end of each respective IP section of the SoCRocket User Manual.

To facilitate design space exploration (DSE), the library provides a mechanism for generating platform simulations from templates.

### Configuring the template

### Generating platform simulation and software

### Running/Debugging simulation

### Analysis of simulation results