**Title**

HW-SW SystemC Co-Simulation SoC Validation Platform

**Design Flow Report**

Contract No. 4200022968

by

 TU Braunschweig, Germany

March 15, 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 LEON FFT processor 5

2.2.1 Setting up a platform template 5

2.2.2 Configuring the template 5

2.2.3 Generating platform simulation and software 5

2.2.4 Running/Debugging simulation 5

2.2.5 Analysis of simulation results 5

**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) (Figure 1) 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.

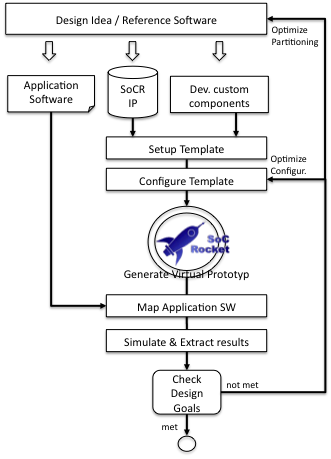


Figure - SoCRocket Design Flow

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

The following chapter explains all phases of the SoCRocket design flow on the basis of an example shipped with the library. The example demonstrates the implementation of a Fast Fourier Transformation\*. The time-domain data is delivered by an on-chip sensor. The sensor produces 1000 data frames, consisting of 64 complex samples (512 bytes), per second. The computed frequency domain data is supposed to be transmitted via a SoCWire link.



Figure - FFT data flow

*\*The example is solely intended to demonstrate the features of the SoCRocket design flow. The presented hardware and software is by no means the optimum solution for the given problem. The foremost aim is to provide a representative system, which may be used as a starting point for other activities, and at the same time keep the different design tasks as simple and understandable as possible.*

### Partitioning the system

The starting point of the design activities is C-Code for a Radix4 FFT, which can be found in the **./software/fft64** directory.

The FFT shall be implemented in software on a LEON3 processor. A respective TL simulation model can be found in the SoCRocket IP library (**./models/extern/LEON3**). To get a first idea about the performance, the code may be executed on the stand-alone processor simulator. The simulator can be compiled by executing following command:

**./waf --target=LEON3\_funcLT**

For compiling the reference software enter:

**./waf –target=fft64.sparc**

The simulator binary and the program (sparc executable) will be dumped into the build directory of the library. To start the simulation run:

**./build/models/extern/LEON3/simulatorSources/funcLT/LEON3\_funcLT -a ./build/software/fft64/fft64.sparc -f 100**

The execution statistic, which is generated at the end of the simulation, shows an instruction count of 20655. Assuming a target clock rate of 100 MHz it seems feasable to execute the code 1000 times per second.

The library also provides a SoCWire model, an AMBA bus model and various peripherals. Modeling the Sensor requires a custom component. How to design own/custom components is explained in the SoCWire User Manual. The resulting initial system is depicted in .

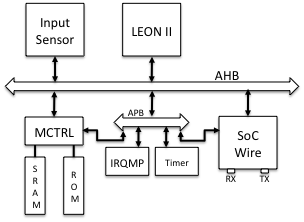


Figure - Initial System

To enable design space exploration and mapping of the hardware dependent software portions, structure and parameters of the system must be captured in a design template.

### Setting up a platform template

In the past years the LEON processor provided by ESA and Aeroflex Gaisler has evolved into a standard target for mapping space applications. Therefore, the template presented in this example has a high potential for re-use.

As previously explained a SoCRocket design template consists of a C file implementing **sc\_main** and a template descriptor for exporting the various configuration parameters. The template files for this example are located in the **./templates/singlecore** directory. In the following all of them will be explained in depth.

**C template - main.cpp**

The C template **main.cpp** implements the SystemC top-level function **sc\_main**. The method instantiates all the simulation models required by the design. The constructor parameters of the different components are represented by variables respecting following syntax:

**conf\_{name of model}\_{name of constructor parameter}**

e.g. **conf\_ahbctrl\_ioaddr – ioaddr parameter of model ahbctrl**

Components can be considered mandatory or optional. In the current example the **ahbctrl** is the central interconnect component and therefore mandatory. Hence, **ahbctrl** is instantiated directly:

120 // CREATE AHBCTRL unit

121 // ===================

122 // Always enabled.

123 // Needed for basic platform

124 AHBCtrl ahbctrl(**"ahbctrl"**,   
125 conf\_ahbctrl\_ioaddr, // The MSB address of the I/O area

126 conf\_ahbctrl\_iomask, // The I/O area address mask

127 conf\_ahbctrl\_cfgaddr, // The MSB address of the configuration area

128 conf\_ahbctrl\_cfgmask, // The address mask for the configuration area

129 conf\_ahbctrl\_rrobin, // 1 - round robin, 0 - fixed priority

130 conf\_ahbctrl\_split, // Enable support for AHB SPLIT response

131 conf\_ahbctrl\_defmast, // Default AHB master

132 conf\_ahbctrl\_ioen, // AHB I/O area enable

133 conf\_ahbctrl\_fixbrst, // Enable support for fixed-length bursts

134 conf\_ahbctrl\_fpnpen, // Enable full decoding of PnP configuration

135 conf\_ahbctrl\_mcheck, // Check for intersections in memory map

136 conf\_ahbctrl\_powmon, // Enable/disable power monitoring

137 ambaLayer

138 );

The instantiation of optional components depends on configuration parameters. In the singlecore template this accounts for, e.g., the GPTimer. Only if the **conf\_gptimer** macro is defined, the GPTimer will be instantiated and connected.

360 #if conf\_gptimer != 0

361 // CREATE GPTimer

362 // ==============

363 GPTimer gptimer(**"gptimer"**,

364 conf\_gptimer\_ntimers,// ntimers

365 conf\_gptimer\_index, // index

366 conf\_gptimer\_addr, // paddr

367 conf\_gptimer\_mask, // pmask

368 conf\_gptimer\_pirq, // pirq

369 conf\_gptimer\_sepirq, // sepirq

370 conf\_gptimer\_sbits, // sbits

371 conf\_gptimer\_nbits, // nbits

372 conf\_gptimer\_wdog, // wdog

373 conf\_gptimer\_powmon // powmon

374 );

375 // Connecting APB Slave

376 apbctrl.apb(gptimer.bus);

377 // Connecting Interrupts

378 **for**(**int** i=0; i < 8; i++) {

379 signalkit::connect(irqmp.irq\_in, gptimer.irq, conf\_gptimer\_pirq + i);

380 }

381 // Set clock

382 gptimer.set\_clk(LOCAL\_CLOCK,SC\_NS);

383 #endif

It is also possible to generate multiple instances of components in a loop. This technique is used in an alternative architecture template, which is also delivered with the library (multicore). In the multicore template the number of CPUs in the system depends on a configuration parameter.

It has already been shown above that also the binding of sockets can be hooked to parameters. The presented method has been proven to be very flexible and straight forward. The amount of flexibility in the template can be easily tuned. Setting up a system for design space exploration requires neither complicated tools nor proprietary techniques.

**XML template descriptor – singlecore.tpa**

The XML template descriptor is used to export and describe the parameters in the C template file. The purpose is to automize the definition of parameters and the generation of hardware dependent scripts for simulation (e.g. linking, waf integration). The code abstract below shows the essential parts of the **singlecore.tpa** template descriptor:

1 <template name=**"Single-Core Platform"**>

2 <description>

3 Default platform template containing one LEON3 processor, with MMU and Cache, AHB/APB bus system, interrupt controller and a fully   
 configurable memory controller.   
 4 GPTimer, SoCWire and a custom Sensor Device   
 can be optionally enabled.

5 <br/>

6 <h2>Features:</h2>

7 <ul>

8 <li>Direct ELF file loading</li>

9 <li>GDB stubs build-in</li>

10 <li>Power Monitor</li>

11 <li>Timing Report</li>

12 <li>Simulation Statistics</li>

13 <li>Automatic boot-code and memmap alignment</li>

14 </ul>

15 </description>

16 <option name=**"Basic Configuration"** var=**"conf"**>

17 <option name=**"System Options"** var=**"sys"**>

18 <option var=**"lt\_at"** name=**"LT or AT simulation"** type=**"bool"**   
 **default**=**"true"** hint=**"True – LT abstraction, False – AT abstr."**/>

19 <option var=**"gdb"** name=**"Enable GDB support"** type=**"bool"**   
 **default**=**"true"** hint=**"Ture - Simulation models will include GDB   
 support. Just add ' gdb' to the end of the comandline to wait for GDB connection."**/>

20 <option var=**"timing"** name=**"Enable Timing Report"** type=**"bool"**   
 **default**=**"true"** hint=**"True - The simulation models will print a   
 timing report at the end of simulation."**/>

21 <option var=**"power"** name=**"Enable Global Power Report"** type=**"bool"**   
 **default**=**"true"** hint=**"True - Aggregate power data for all components. This may slow down the simulation and take up a lot of RAM and HDD space (Not recommended for default)."**/>

22 </option>

23 <option var=**"ahbctrl"** name=**"AHB Controller"**>

24 <option var=**"ioaddr"** name=**"AHB IO Area Address"** type=**"int"**   
 **default**=**"0xFFF"** range=**"0..0xFFF"**/>

25 <option var=**"iomask"** name=**"AHB IO Area Mask"** type=**"int"**   
 **default**=**"0xFFF"** range=**"0..0xFFF"**/>

…

42 <option var=**"mcheck"** name=**"Check the consistency of the memory map"**   
 type=**"bool"** **default**=**"true"**>

43 <description>

44 Check if there are any intersections between core memory areas.   
 In case of error issues a failure assertion.

45 </description>

46 </option>

47 </option>

…

62 <option var=**"mmu\_cache"** name=**"MMU Cache"**>

63 <option var=**"addr"** name=**"AHB Base Address"** type=**"int"** **default**=**"0"**

…  
 99 </option>

100 </option>

…

183 <option var=**"irqmp"** name=**"IRQ Controller"**>

…

190 </option>

191 <generator type=**"systemc"** name=**"config.h"**/>

192 <generator type=**"template"** path=**"systemc"** name=**"wscript"**   
 src=**"singlecore/scwscript.py"** />

193 <file type=**"systemc"** name=**"sc\_main.cpp"** src=**"singlecore/main.cpp"** />

194 <file type=**"systemc"** name=**"prom.S"** src=**"singlecore/prom.S"** />

195 <file type=**"systemc"** name=**"prom.ld"** src=**"singlecore/prom.ld"** />

196 <file name=**"wscript"** src=**"singlecore/wscript.py"** />

197 </template>

The name of the outermost tag (line 1 – line 197) defines the name of the template. It is displayed by the Configuration Wizard and despite has no effect.

### Configuring the template

### Generating platform simulation and software

### Running/Debugging simulation

### Analysis of simulation results