TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

MIKKO TEUHO

DESIGN LEVEL PARAMETERIZATION IN KACTUS2

Master of Science thesis

Examiner:Prof. Timo D. Hämäläinen
Examiner and topic approved by the Faculty Council of the Faculty of Computing and Electrical Engineering on September 9th 2015

ABSTRACT

Mikko Teuho: Design level parameterization in Kactus2
Tampere University of technology
Master of Science Thesis, 56 pages
November 2015
Master's Degree Programme in Information Technology
Major: Pervasive Systems
Examiner: Professor Timo D. Hämäläinen

Keywords: Kactus2, electronic design automation, embedded systems, system-on-chip, IP-XACT

Embedded systems are growing larger and more complex. Even now, current system designs can contain hundreds of Intellectual Property (IP) components. To keep up with productivity, the reusability of the IP components must be improved. This is the scope of IEEE standard IP-XACT.

This thesis is based on Kactus2, an open source IP-XACT tool developed at Tampere University of Technology. Kactus2 provides a graphical user interface for System-on-Chip and embedded system IP packing, design capture and VHDL/Verilog code generation.

This thesis describes the development and implementation of version 2.8 of Kactus2. The requirements and solutions are presented in detail for each of the new features and improvements implemented in version 2.8. Alternative solutions are presented, and the selected alternatives are justified. Possible future implementations are also given.

In version 2.8, the parameter usage of IP components is improved through the use of universally unique IDs (UUID), which requires many changes e.g. to the IP-XACT Component editors. New features include parameter importing, design level configuration through parameters and a parameter propagation mechanism. Remapping IP-XACT Memory Maps through the use of Remap States and memory Remap Elements has also been added. To facilitate the storing of hierarchical IP components, a new save action has been added to the Kactus2 toolbar.

Version 2.8 of Kactus2 was released according to its schedule. The development of the version is considered a success, as it improves the design level parameterization in Kactus2 while incorporating additional new features. Within a month of its release, Kactus2 version 2.8 has been downloaded over 200 times, and its benefits over the previous version are confirmed by industrial System-on-Chip developers.

TIIVISTELMÄ

Mikko Teuho: Suunnittelutason parametrisointi Kactus2:ssa.
Tampereen teknillinen yliopisto
Diplomityö, 56 sivua
Marraskuu 2015
Tietotekniikan diplomi-insinöörin tutkinto-ohjelma
Pääaine: Ohjelmistotuotanto
Tarkastaja: professori Timo D. Hämäläinen

Avainsanat: Kactus2, elektroniikan suunnitteluautomaatio, sulautetut järjestelmät, system-on-chip, IP-XACT

Sulautetut järjestelmät kehittyvät koko ajan suuremmiksi ja monimutkaisemmiksi. Nykyiset järjestelmät voivat sisältää satoja IP-lohkoja (Intellectual Property). Tuottavuuden ylläpitämiseksi tarvitaan parannuksia IP komponenttien uudelleenkäyttöön. Tähän on tarkoitettu IEEE:n standardi IP-XACT.

Tämä työ perustuu TTY:n kehittämään Kactus2 IP-XACT-työkaluun. Kactus2 tarjoaa graafisen käyttöliittymän järjestelmämikropiirien(SoC) IP-lohkojen paketointiin, integrointiin, konfigurointiin ja VHDL/Verilog-koodin tuottamiseen.

Tässä työssä kuvataan Kactus2:n version 2.8 kehitys ja toteutus. Versioon kuuluvat vaatimukset ja näiden ratkaisut kuvataan yksityiskohtaisesti jokaisen ominaisuuden osalta. Toteutetut ominaisuudet selostetaan ja mahdolliset vaihtoehtoiset ratkaisut esitellään. Lisäksi annetaan esimerkkejä mahdollisista parannuksista Kactus2:n ominaisuuksiin tulevaisuudessa.

Uusien ominaisuuksien joukossa Kactus2:n versiossa 2.8 on parametrien uudelleenkäytön parantaminen. Tähän kuuluvat parametrien tuominen tiedostoista, sekä suunnittelutason komponenttien muokkaaminen. Myös muistikarttojen uudelleenkartoitus on toteutettu standardin mukaisilla uudelleenkartoitustiloilla. Hierarkkisten komponenttien tallentaminen on helpotettu lisäämällä mahdollisuus tallentaa koko hierarkkinen järjestelmä yhdellä toiminnolla.

Versio 2.8 julkaistiin aikataulunsa mukaisesti ja sen kehitystä pidetään erittäin onnistuneena. Tämä versio parantaa suunnittelutason parametrisointia Kactus2:ssa, sekä lisää muita uusia ominaisuuksia. Kactus2:n versiota 2.8 on ladattu yli 200 kertaa kuukauden sisällä julkaisusta, ja sen hyödyt verrattuna vanhaan versioon on varmistettu teollisuuden ammattisuunnittelijoiden palautteella.

PREFACE

No thesis can be written alone. There are some exceptions to this, but since this is not the case here, I would like to express my gratitude to the people who have helped make this happen.

I would like to thank the entire department of Pervasive Computing in Tampere University of Technology for the friendly working atmosphere. Esko Pekkarinen for helping me understand Kactus2. Panu Sjövall for help with the generators of Kactus2. Timo D. Hämäläinen for offering me the opportunity to work on this project.

And the folks at home: my sister for graduating before me and showing how it can be done, my brothers for helping me carry on despite everything, and my parents for encouraging me onwards.

Tampere, 13.11.2015

Mikko Teuho

CONTENTS

1. INTRODUCTION 1
2. KACTUS2 3
   - 2.1 Design principles 3
     - 2.1.1 Tools related to Kactus2 4
   - 2.2 User interface 7
     - 2.2.1 Tools of Kactus2 10
   - 2.3 Component editor 11
     - 2.3.1 Memory maps editor 12
     - 2.3.2 Views editor 13
   - 2.4 Design editor 14
   - 2.5 Generators 16
     - 2.5.1 Kactus2 generators 16
     - 2.5.2 Plugin generators 17
     - 2.5.3 Source analyzers 18
     - 2.5.4 Import Plugins 19
   - 2.6 Summary 20
3. DEVELOPMENT ENVIRONMENT 21
   - 3.1 Microsoft Visual Studio 2010 21
     - 3.1.1 Visual Assist 22
   - 3.2 Qt 5.2.0 22
     - 3.2.1 Model / View / Delegate 22
     - 3.2.2 Signals and Slots 23
     - 3.2.3 Creating a tree model with Qt 23
   - 3.3 Redmine 24
   - 3.4 Source forge 25
4. REQUIREMENTS 26
   - 4.1 Feature #9: Memory map editor should show the edited item 26
   - 4.2 Bug #121: Show errors is not visible for all library items 26
   - 4.3 Feature #171: parameter usage 27
     - 4.3.1 Feature #177: Design level configuration 27
   - 4.4 Feature #180: Signal based configuration 28
     - 4.4.1 Feature #192: RemapStates and memoryRemap 28
     - 4.4.2 Feature # 193: Tabs for memory remaps 30
   - 4.5 Feature #206: Scrollable ribbon 30
   - 4.6 Feature #214: Support parameter IDs in Verilog generation 31
   - 4.7 Feature #216: Support parameter IDs in HTML document generation 31
   - 4.8 Feature #217: Support parameter IDs in generated header files 31
   - 4.9 Feature #221: Add a vendor extension for Kactus2 version 32
   - 4.10 Feature #222: Locked column in parameter editor 32

4.11 Feature #236: Group configurable element values 32
4.12 Feature #237: Product level view configuration tool 33
5. IMPLEMENTATION 34
5.1 Memory map editor should show the edited item 34
5.2 Errors are not visible for all library items 35
5.3 Parameter usage 36
5.3.1 Design level configuration 38
5.4 Memory remap and remap states 39
5.5 Scrollable ribbon 41
5.6 Parameter ID support 42
5.6.1 Verilog Generator 43
5.6.2 HTML Document Generator 43
5.6.3 Memory map Header Generator 44
5.7 Locked column in parameter editor 44
5.8 Product level view configuration tool 46
6. ANALYSIS 49
7. KACTUS2 VERSION 2.8 VERIFICATION 52
7.1 Expressions in Kactus2 52
7.2 Unresolved issues 53
8. CONCLUSIONS 56
REFERENCES 57

LIST OF FIGURES

Figure 2.1 The user interface of Kactus2 7
Figure 2.2 The different versions of the IP-XACT library tree. The left figure displays a VLNV tree, while the right displays a hierarchical tree. 8
Figure 2.3 Basic view of the Kactus2 toolbar. 10
Figure 2.4 Component editor. 12
Figure 2.5 Design editor. 15
Figure 2.6 Source analyzers in Kactus2 fileSets editor. 19
Figure 3.1 The tree model structure in Qt [33] 24
Figure 4.1 Configurable elements editor. 28
Figure 4.2 IP-XACT memory remap schema [4]. 29
Figure 4.3 IP-XACT remap state schema [4]. 29
Figure 4.4 Toolbar of Kactus2 version 2.6. 30
Figure 5.1 Memory map editor 34
Figure 5.2 Component with errors in the Kactus2 library view. 35
Figure 5.3 The expression selection.[39] 37
Figure 5.4 The improved configurable element editor. 38
Figure 5.5 The combined editor for memory maps and memory remaps. 39
Figure 5.6 Remap state editor. 40
Figure 5.7 Scrollable toolbar. 41
Figure 5.8 Class diagram of the ParameterFinders used in ExpressionFormatter. 42
Figure 5.9 Table view with a locked column 45
Figure 5.10 Alternative locked column. 45
Figure 5.11 Another alternative locked column. 46
Figure 5.12 View Configuration Tool 47
Figure 6.1 Kactus2 Gantt chart for version 2.7.x [38] 50
Figure 6.2 Kactus2 Gantt chart for version 2.8 [38]. 51
Figure 7.1 The parameter reference tree. 53
Figure 7.2 A suggestion for memory remap visualization. 54

LIST OF TABLES

Table 2.1 Source analyzers of Kactus2. .... *18*
Table 2.2 Import plugins of Kactus2.. *19*
Table 6.1 Estimated time spent for each issue. *49*

LIST OF SYMBOLS AND ABBREVIATIONS

AUB	Address Unit Bits
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
COM	Communication
ESL	Electronic System Level
FPGA	Field-Programmable Gate Array
HDL	Hardware Description Language
HTML	Hypertext Markup Language
HW	Hardware
IP	Intellectual Property
IP-XACT	A standardized XML-format for defining IP
MCAPI	Multicore Communications API
RTL	Register-transfer level
SoC	System-on-a-Chip
SW	Software
VHDL	Very High Speed Integrated Circuit HDL
VLNV	Vendor Library Name Version
XML	Extensible Markup Language

1. INTRODUCTION

This thesis is based on Kactus2, an open source tool for embedded system and system-on-a-chip (SoC) design using IP-XACT metadata. Kactus2 is being developed by Tampere University of Technology.

An embedded system describes a system consisting of a hardware (HW) platform and the software within it. The platform is constructed using reusable intellectual property (IP) blocks. A SoC is a circuit consisting of multiple IP components in a single chip [1]. The software component is executed on one or more programmable cores of the system. A HW platform may contain different implementations, and the implementation may be ported on several platforms.

Digital systems are growing larger and more complicated, with current SoC designs containing hundreds of IP blocks [2]. This results in huge stacks of layers and application programming interfaces (APIs) with a multitude of programs and libraries. Kactus2 was developed to make designing SoCs and embedded systems easier.

Kactus2 is a graphical user interface tool designed to help with the management and integration of reusable IP blocks built with the IP-XACT metadata, as well as to help with the system design flow through an easy to use interface, memory visualization and a hidden IP-XACT structure. The developed version 2.8 of Kactus2 contains features belonging to the design level parameterization of component instances.

IP-XACT is an extensible markup language (XML)-format used for unified specification of electronic design automation, semiconductor, electronic design intellectual property and system design communities [3] [4]. Through the use of IP-XACT metadata, IP components can be packaged and reused regardless of the tool, implementation or vendor used for creating the IP-component. IP-XACT was developed for both IP exchange and SoC configuration [5].

The purpose of this work is to develop version 2.8 of Kactus2 and extend its capabilities as a tool for SoC and embedded system design by developing design level parameterization. Solutions to the encountered issues are explained together with any discovered optional solutions.

This thesis is organized in a way to allow consistent flow of information from one area of interest to the next. The first chapter gives a brief introduction to this thesis. Chapter two presents the Kactus2 application through explaining the different editors, tools and generators available within Kactus2. Additionally, this chapter gives a comparison of other tools that have been developed regarding SoC and embedded system design. Chapter three features information regarding the development environment of Kactus2, and the various tools in managing the development process. Chapter four gathers requirements of version 2.8 of Kactus2. The covered issues are extracted from the Redmine issue control system of the Kactus2 project, and have a feature number associated with each feature to help with the identification of the different issues. Chapter five describes the implementation of the features given in chapter 4. The implementation includes alternative solutions and information regarding any difficulties faced during development. Chapter six recounts the implemented features as well as evaluates the amount of work done. Chapter seven gives an evaluation of version 2.8. Chapter eight contains the conclusions made from the development, and ends with an analysis of the future development of Kactus2.

2. KACTUS2

This chapter introduces Kactus2, an open source tool for SoC and embedded system design [6] [7]. Kactus2 guides the user in creating IP-components and designs through a graphical user interface, using the IP-XACT standard for easier reuse of IP components. While the development of Kactus2 is still ongoing, this work concentrates on the development of version 2.8.

2.1 Design principles

IP components or cores are reusable logical building blocks for SoCs [1]. Cores come in three forms: hard, firm and soft cores. Hard cores are encrypted simulation models with a provided high performance mode, such as microprocessors or phase-locked loops. Firm cores are provided with a logic synthesis and technology mapping, but without layout information. Soft cores require the user to implement the synthesis and layout information.

SoC is an integrated circuit consisting of reusable IP-components of a board in a single chip [1], typically applied in embedded systems. In SoCs, a large design can be split into multiple smaller portions. This creates subsystems of the design, where each subsystem has fewer components to test. Thus the testing and verification of the instantiated chip are made more manageable [8].

Platform-based SoCs have been set up as the engines for embedded systems. However, current SoC designs may incorporate hundreds of IP blocks. In an ideal case, a high-enough description of the required application functionality and available execution platforms exists for the design. Using this kind of model-based design, the implementations could be automated for the SOC, and both the applications and platforms could be made reusable. However, according to Kamppi et al. [2], this introduces the following obstacles to the model-based design:

- There are too many paradigms and domain-specific models within the design.
- Modelling the design is a separate side-task not fitting to the whole flow.
- The amount of legacy code base and the different versions and variants are not adequately accounted.
- Automation for one-way use from top to bottom works only in ideal settings using limited use cases.
- The high abstraction makes the creation and comparison of visual models difficult.

Kactus2 is based on the IEEE 1685/ IP-XACT standard, which provides a framework for enabling both integration of hardware components and maintaining the relevant design information of a product [5]. The main concepts for the design of Kactus2 are the utilization of standards, a small learning curve for new users and the effortless incorporation of legacy IP components [2]. To accommodate the IP components, an IP-XACT library is supported. The library can be imported from other vendors, checked for integrity and exported from Kactus2.

Kactus2 combines a component editor, a component integration designer and source code generators [8]. Through extending the IP-XACT standard, Kactus2 supports software, software to hardware mappings and application communication abstraction using e.g. multicore communications API (MCAPI). With the standard IP-XACT metadata and Kactus2 extensions, porting applications between hardware (HW) and software (SW) or between platforms becomes simpler. [2]

With the help of the library, Kactus2 can create system designs for mapping HW and SW components. The designs can further be configured to increase the reusability of the IPs. Design synthesis and simulation scripts can be created in generators contained within Kactus2. The generators create code templates, including Very High Speed Integrated Circuit Hardware Description Language (VHDL) entities, ports and C headers for new IP-components. Additionally, Kactus2 can integrate additional, user created generators as plugins.

2.1.1 Tools related to Kactus2

System design tools can be placed in two different categories [8]: system composing tools for higher abstraction level models, and tools that manage IP-blocks and incorporate them into larger entities. Kactus2 belongs to the second group. Some of these tools allow only file management of the associated system.

Mentor Graphics offers the Hardware Description Language (HDL) Designer as a tool for Verilog and VHDL design environment, supporting IP-XACT standard [9]. The HDL Designer contains graphical user interfaces for instantiating and connecting IP-blocks, along with tools for design analyzation, creation and management. For example, Lockheed Martin Space Systems utilizes the HDL Designer in solving verification challenges in space-based Field-Programmable Gate Arrays (FPGAs) [10].

M-Files provides a tool for information storage and management [11]. It takes care of the documents, making sure everything is up to date and in the correct place. Through a check-in and check-out system, the editor of the files can be tracked. M-Files supports locking and sharing of documents. However, other tools are needed to edit the documents contained within the M-Files.

Scineric Workspace, developed by The Council for Scientific and Industrial Research in South Africa (CSIR), is a tool for FPGA design management [12]. Scineric Workspace is a file manager tool for importing and exporting many design formats using a customized SVN and Git version control. The tool contains build management capabilities for Xilinx, Altera and Plunify environments. Scineric Workspace manages the designs as components using IP-XACT standard.

Abdi et al. suggest Embedded System Environment (ESE), a model-based design method toolset for multi-processor embedded systems [13]. ESE generates Transaction Level Models (TLM) from the visualized platform and the code of an application. The TLMs can be used for SW development or for estimating system performance, thus iteratively refining the system definition. ESE provides an automatic synthesis from the TLM, the required software and hardware. With these, a cycle-accurate model is made. This Cycle Accurate Model can be given to an FPGA or an Application-Specific Integrated Circuit (ASIC) design tool.

Magillem provides a commercial IP-XACT based toolset for hardware design, embedded software and documentation [14]. These tools provide register managing for SoC solutions with an integrated design environment for platform assembly and configuration in implementing IP-based systems.

Socrates Design Environment developed by ARM provides a commercial IP-XACT design environment for IP configuration and integration [15]. Socrates Design Environment configures and integrates IP of IEEE1685-2009 standard with ARM IP to create a SoC. IP definition, IP packaging, configurability, connectivity and registers are taken into account.

Xilinx has developed the commercial Vivado Design Suite as a SoC-strength, IP- and system centric development environment for system-level integration and implementation [16]. This Design Suite provides acceleration to design, integration, verification and implementation processes of systems. The Design Suite achieves these through the use of software-programmable ARM processing systems, programmable Analog Mixed Signal subsystems and an expanding range of IP cores. Additionally, Vivado Design Suite uses place-and-route algorithms to increase the efficiency of integrated circuit design interconnections. Xilinx uses IP-XACT as the IP format.

Nivalis Solutions provides IP-XACT based tools for managing IP design using a cloud service [17]. Nivalis’ IP Packager is a web application for IP content built on an extended IP-XACT standard. The Design Assembly combines HW assembly and SW addressing operations and creates a schematic view of a design provided by the assembled IPs.

EDAUtils provides a free IP-XACT design environment for constructing and integrating IP-components [18]. EDAUtils provides a graphical user interface for building IP-XACT components from ports and parameters of a register-transfer level (RTL). The tool allows importing component definitions from VHDL and Verilog source files, offers IP-XACT validators and an equivalence checker for two component XML-files.

Nikolov et al. has introduced Daedalus, a system design flow for designing multiprocessor SoC (MP-SoC) based embedded systems [19]. Daedalus uses an integrated and highly automated environment for system design. Daedalus thus provides an open source integrated toolset for design space exploration, system-level synthesis, application mapping and prototyping of MP-SoCs.

Balarin et al. has designed Metropolis [20], a design environment for integrated electronic systems. The goal for Metropolis is to create a unified framework for system-level design chain. Metropolis provides an environment for complex electronic-system design supporting simulation, analysis and synthesis.

SystemCoDesigner, developed by Haubelt et al. [21], is an electronic system-level tool for design space exploration and prototyping of SystemC models on an FPGA basis. SystemCoDesigner helps in the early stages of design by closing the gap between electronic system level (ESL) and RTL.

Eker et al. have presented the Ptolemy approach [22]. The Ptolemy II introduces a model structure and semantic framework to support a hierarchical heterogeneity modelling environment using domain polymorphism. The Ptolemy II software environment introduces support for combining a variety of computation models hierarchically.

Synopsys has developed CoreBuilder [23], a tool for creating IP-XACT metadata and incorporating the reuse of IP-components. CoreBuilder captures the detailed information of an IP core and the design. The core is packaged into a coreKit with its associated design files e.g. verification and documentation files.

Microsoft Excel [24] can be used as an improvised tool for listing parameters, registers or other IP component elements. These can be used in to form referencing expressions. The expression editor of Kactus2 has incorporated similar features existing in Excel, such as defining complex equations, or selecting a reference suitable to the currently written input. Additionally, the idea of locked columns in a table editor of Kactus2 is originally a feature in Excel. The implementation of this feature can be found in section 5.7.

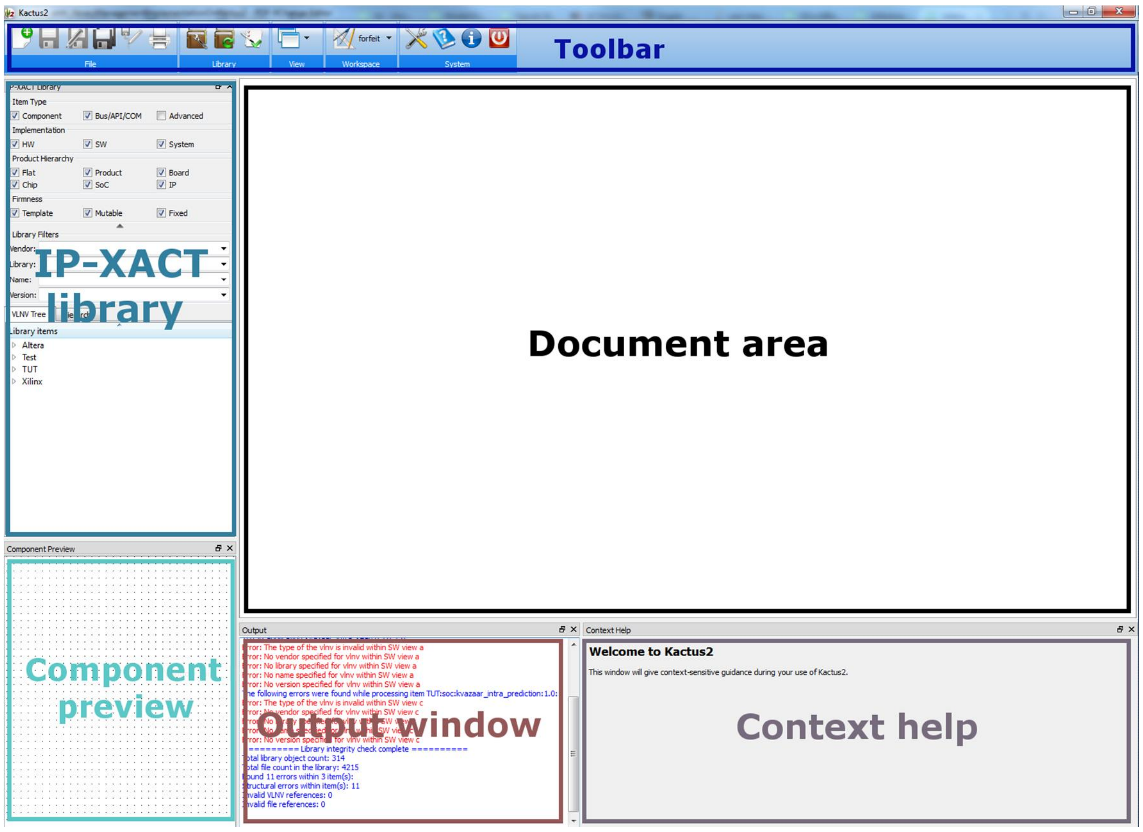
IP-XACT based tools usually extend the basic IP-XACT standard through vendor extensions. As an example, Xilinx Vivado has approximately 100-200 extensions. Extending the IP-XACT has a potential to result in vendor locking or unable to incorporate IP components created on another system. The IP-XACT standard is still valid in the corresponding applications, but the extensions in them are not usable by other tools.

Compared to the tools listed here, Kactus2 offers an IP-XACT based open source solution, combining both design management and a component editor. One or the other is missing in some of the related tools. Kactus2 provides access to the management and creation of the IP-components that is lacking in all tools focused on file management.

2.2 User interface

The main user interface of Kactus2 is presented in Figure 2.1. The interface consists of multiple views, with each view showing information regarding the currently active libraries and documents. The views can be repositioned, resized and closed to allow the user to customize the interface. Closed views can be reopened through the view tools in the toolbar. This gives freedom and control to the user, increasing the user friendliness of the application [25].

Using the default settings, the toolbar of Kactus2 is located in the top section of the interface, while the IP-XACT library is situated in the left side, along with the component preview window. The output and context help windows are located at the bottom section of the interface. The area reserved for the currently open document is located at the center of the interface. The document area contains all the views and editors related to the open document.



```
rror: The type of the vlnv is invalid within SW view a
rror: No vendor specified for vlnv within SW view a
rror: No library specified for vlnv within SW view a
rror: No name specified for vlnv within SW view a
rror: No version specified for vlnv within SW view a
he following errors were found while processing item TUT:soc:kvazaar_intra_prediction:1.0:
rror: The type of the vlnv is invalid within SW view c
rror: No vendor specified for vlnv within SW view c
rror: No library specified for vlnv within SW view c
rror: No name specified for vlnv within SW view c
rror: No version specified for vlnv within SW view c
========== Library integrity check complete ==========
otal library object count: 314
otal file count in the library: 4215
ound 11 errors within 3 item(s):
tructural errors within item(s): 11
nvalid VLNV references: 0
nvalid file references: 0
```

Welcome to Kactus2

This window will give context-sensitive guidance during your use of Kactus2.

***Figure 2.1** The user interface of Kactus2.*

The contents of the document area depend on the selected library element. The open document shows a component editor for components. This editor allows the user to manage the data contained within the selected IP-XACT component. The type of the component determines the actual component editor used. This is further described in section 2.3. If the selected library element is an HW, SW or system design, the view displays the design editor. Design editors are used to manage component designs and hierarchies. This editor is further explained in section 2.4.

The IP-XACT library is located on the left side of the user interface. The contents of the library are searched from the user set active library locations. The given paths and their subdirectories are analyzed for documents containing IP-XACT related tags in files with an XML suffix. Kactus2 then analyzes the dependencies between the collected components [7]. The IP-components are organized in the library according to their vendor, library, name and version (VLNV), a unique version identifier group used to define a reference of or from an IP-XACT object [3].

The IP-XACT library has two different view styles. The VLNV tree view is displayed on the left side of Figure 2.2. This tree is constructed using the VLNVs of the IP-XACT objects found in the library paths, forming a four level deep tree structure.

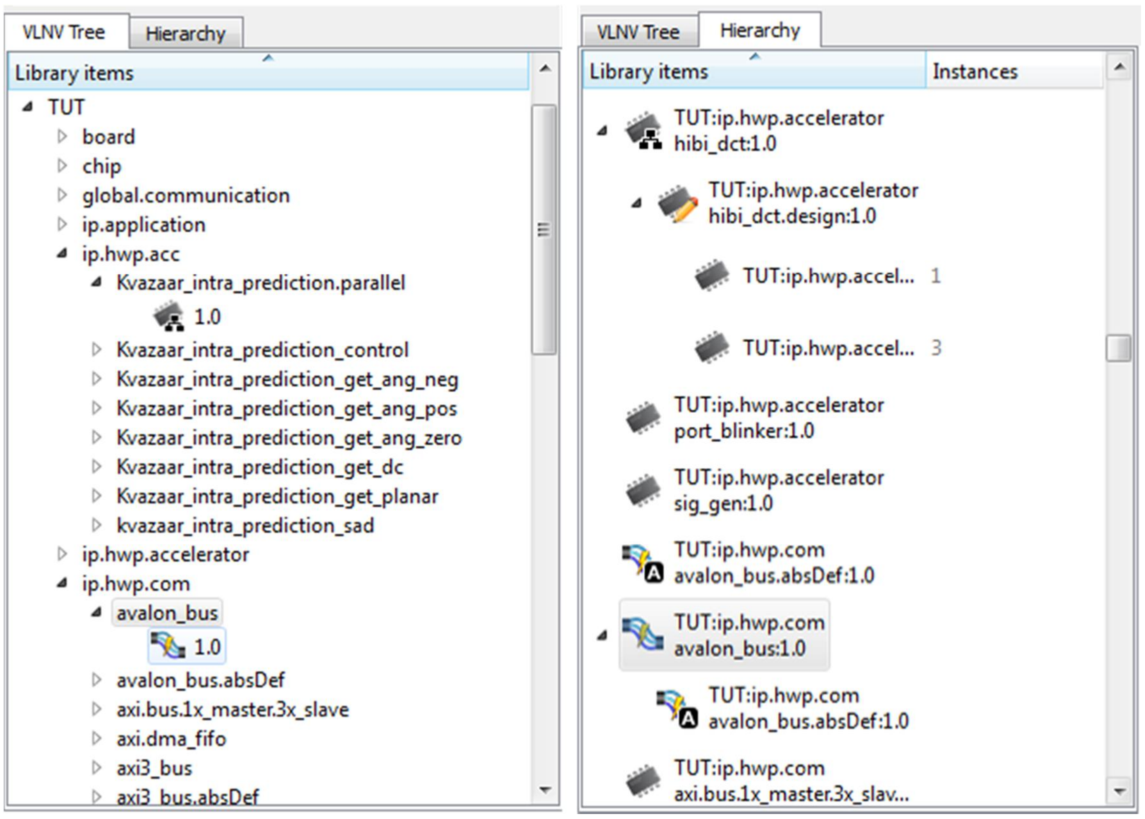

VLNV Tree
Hierarchy
Library items
TUT
board
chip
global.communication
ip.application
ip.hwp.acc
Kvazaar_intra_prediction.parallel
1.0
Kvazaar_intra_prediction_control
Kvazaar_intra_prediction_get_ang_neg
Kvazaar_intra_prediction_get_ang_pos
Kvazaar_intra_prediction_get_ang_zero
Kvazaar_intra_prediction_get_dc
Kvazaar_intra_prediction_get_planar
kvazaar_intra_prediction_sad
ip.hwp.accelerator
ip.hwp.com
avalon_bus
1.0
avalon_bus.absDef
axi.bus.1x_master.3x_slave
axi.dma_fifo
axi3_bus
axi3_bus.absDef
VLNV Tree
Hierarchy
Library items
Instances
TUT:ip.hwp.accelerator hibi_dct:1.0
TUT:ip.hwp.accelerator hibi_dct.design:1.0
TUT:ip.hwp.accel... 1
TUT:ip.hwp.accel... 3
TUT:ip.hwp.accelerator port_blinker:1.0
TUT:ip.hwp.accelerator sig_gen:1.0
TUT:ip.hwp.com avalon_bus.absDef:1.0
TUT:ip.hwp.com avalon_bus:1.0
TUT:ip.hwp.com avalon_bus.absDef:1.0
TUT:ip.hwp.com axi.bus.1x_master.3x_slav...

Figure 2.2 *The different versions of the IP-XACT library tree. The left figure displays a VLNV tree, while the right displays a hierarchical tree.*

The other view, displayed on the right side in the Figure 2.2 is the hierarchical tree view. This view displays dependencies between the different objects [26]. Each hierarchical component contains the designs referenced by its hierarchical views and the component instances contained within the design. Additionally, the hierarchical tree informs the amount of instances made from the same components. An example of this is displayed in the instances belonging to the design *TUT:ip.hwp.accelerator hibi_dct.design:1.0* in Figure 2.2.

The library can be filtered according to type of item, implementation type of component, product hierarchy and firmness. In addition, the library can be searched for a specific set of VLNV values. The components containing errors are displayed in red and the user can examine the errors through a context menu associated with the library.

The items in the library are shown as a tree model. A tree model helps to identify the relative positions of the components. In the hierarchical view, the tree model helps to describe the components that are connected to other components, such as bus abstraction definitions being a part of bus definitions and component instances belonging to a component design.

A component preview window shows the preview image of the selected component. This image is the basic, unaltered form that is used when a component is instantiated in a design. The preview image displays the name of the component, an icon displaying whether it is a hierarchical component or not and all the interfaces contained within it, distinguishing the slave and master interfaces.

The output window prints text related to the user operations. This helps in identifying possible errors within the currently selected component or design. Information on the running tasks, such as generator procedures and library integrity checking is also shown here. Notification messages are displayed in blue while warning messages are displayed in red.

To fully customize the user interface of Kactus2, the concept of workspaces was added. These allow the user to create task specific configurations to the user interface, which can be stored in the Kactus2 settings file. The workspaces include information of the size and position of the main window of Kactus2, the currently open views and the configurations given to Kactus2. The user can freely switch between the different workspaces through the Kactus2 workspace tools. The user may also delete obsolete workspaces, with the exception of the default and the currently active workspaces.

Kactus2 aims to be as user friendly as possible. Context sensitive help files are used to guide the user in operating the currently active document or editor. Awkward IP-XACT terms are hidden and information is displayed with tooltips and highlights to ensure that important data is always available. In a component editor, each editor has its own help file. This file is shown when an editor is selected. In a design editor, selecting a design item, such as a component instance or an interconnection, opens a help file related to it.

2.2.1 Tools of Kactus2

The available tools in Kactus2 are located in the top part of the user interface. Figure 2.3 displays the available toolbar. The toolbar is divided into tool specific category groups. The tools are context sensitive, being disabled and hidden when the user cannot use them. If there is not enough space to display the whole toolbar, two buttons are enabled to scroll the toolbar left and right. The scrolling can also be done by using the mouse wheel. For the implementation of the toolbar, see section 5.5. Figure 2.3 does not display the generator, diagram, configuration or protection tool groups.

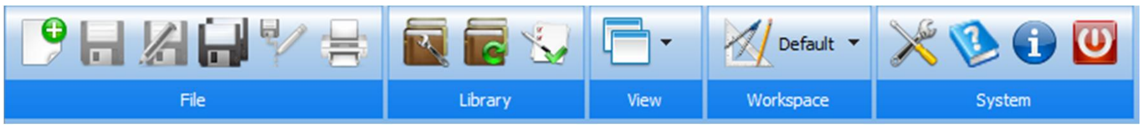


Figure 2.3 *Basic view of the Kactus2 toolbar.*

The file group contains actions related to the creation and saving of VLNV items. New VLNV items can be created as components, designs, systems, buses, communication (COM) definitions or API definitions. In addition to regular saving features, Kactus2 provides an option for hierarchically saving a design. This allows the saving of the open document and the underlying hierarchy into a location determined by the user.

The library group includes tools related to the IP-XACT library. The group offers tools for defining the library folders used in Kactus2, as well as checking the integrity of the libraries. The paths are set through selecting folders in the library configuration tool. The selected folders and the contained sub-folders are then searched for xml-files containing IP-XACT components. The library paths can also be re-searched for xml-files containing any components.

The generator tools are located in the generation group. The generator group is not visible in Figure 2.3 because the user cannot generate any files when no component has been selected. The contents of this selected component or design determine what generators are currently usable. Kactus2 contains two types of generators: built-in Kactus2 generators and separate plugin generators. Kactus2 generators are useful for every user, while plugin generators provide additional generators, which can be disabled and enabled depending on the requirements of the user.

The diagram tools are part of the design editor. For more information on design editor, see section 2.4. These tools are used for defining and expanding the design diagram. New draft components and interfaces can be created for the top component and interconnections between interfaces can be formed.

The configuration tools group provides tools for managing the currently selected component or design. In Kactus2 version 2.8, the view configuration tool is the only available configuration tool. It allows the configuration of the views used by the generators through selecting a different set of active views from the currently selected ones. The generators automatically use a contained view configuration instead of the currently selected one.

The workspace tool allows creation of new workspaces, changing the current workspace and deleting a workspace. Kactus2 contains two workspaces, default and design. Workspaces are used to define a working environment. When a workspace is changed or Kactus2 is shut down, the state of the active workspace will be stored in the settings file of Kactus2.

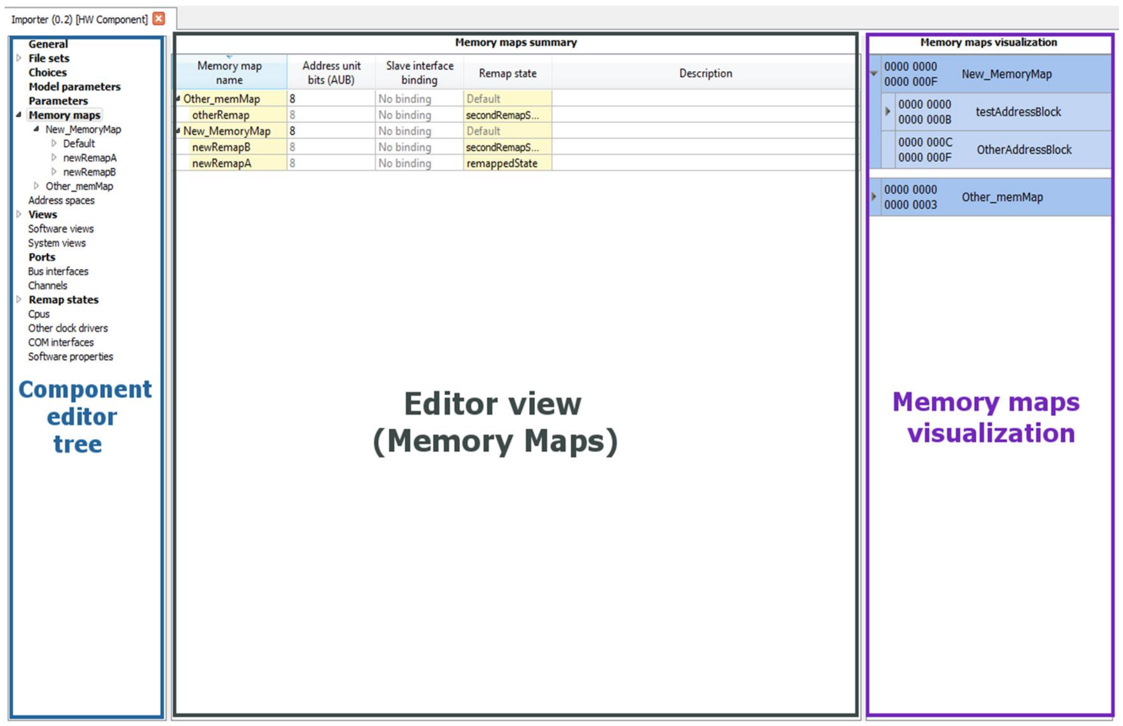
The actions of the protection group are related to the status of Kactus2, defining whether the user can make modifications to the currently open document or not. The editor locking is used to prevent user errors which, according to Nielsen, contribute to the usability of the application [25]. The protection status of the currently open document is visible in the protection tool’s icon. A red, closed lock depicts a locked document while a green open lock depicts a document available for modifying.

The system group contains tools for the Kactus2 as a system. The actions within this group allow the changing of the overall settings of Kactus2 and accessing the Kactus2 help files. An exit button is included for a safe shutdown of the application.

2.3 Component editor

The component editor allows the editing of a selected IP-XACT component. A view of the memory maps editor contained within the component editor is displayed in Figure 2.4. The editable sections of the component depend on the component editor visibility settings. When the component editor is opened, general information along with a preview image of the component is shown to the user and the context help window displays the help file related to the currently open editor.

The component editor consists of two main views, the component editor tree and the editor view. The former shows the currently visible, editable sections of the component. Some of these sections, like the Memory maps of a component contain sub-sections, with their own editors. Opening a sub-section expands the component editor tree, as is shown in Figure 2.4. Selecting any of the sections opens up the editor view. The editor view contains information of the editable section, and allows the creation and editing of the parts belonging to the selected component item.



Memory maps summary

Memory map name	Address unit bits (AUB)	Slave interface binding	Remap state	Description
Other_memMap	8	No binding	Default	
otherRemap	8	No binding	secondRemapS...	
New_MemoryMap	8	No binding	Default	
newRemapB	8	No binding	secondRemapS...	
newRemapA	8	No binding	remappedState	

Memory maps visualization

0000 0000 0000 000F	New_MemoryMap
0000 0000 0000 000B	testAddressBlock
0000 000C 0000 000F	OtherAddressBlock
0000 0000 0000 0003	Other_memMap

Figure 2.4 *Component editor.*

In addition to the two main views, some editors make use of a third view to display data, as shown in Figure 2.4. The third view is situated on the right side of the component editor interface. In version 2.8 of Kactus2, only the General and Memory maps editor use the third view, with the general editor showing a preview image of the edited component, while the memory maps editor shows a visualization of the contained memory maps and their associated address blocks, with registers of each address block and fields of every register.

Kactus2 extends the IP-XACT standard to accommodate elements from the newer 2014 IP-XACT standard and prepare Kactus2 for the transition from the older 2009 version to the 2014 version. These include the addition of isPresent expression fields for registers and fields, and the addition of module parameters to the views of a component [4].

2.3.1 Memory maps editor

The memory maps editor allows the management of Memory Maps and Memory Remaps. Memory maps are part of the IP-XACT standard and contain information regarding the memory locations within a component, address blocks contained within each memory map, registers located within each address block and fields of all the registers. As shown in Figure 2.4, in addition to the name and description, the editable fields of a memory map are Address Unit Bits (AUB), slave interface binding and remap state.

AUB defines the amount of data bits for address increments in a memory map. By default, the memory maps are byte addressable, meaning the AUB is 8 bits. Usually the AUB is given as a multiple of 8, e.g. 8, 16, 32. Changing the AUB of a memory map affects the base address and range of the containing address blocks and the offset of the registers within the address blocks.

Slave interface binding describes the slave interface that has been bound to this memory map and used for accessing it. The memory maps editor does not allow the selection of this binding, but the interface can be bound in the bus interface editor of a slave bus interface.

In addition to editing memory maps, the memory maps editor allows the managing of memory remaps. The memory remaps are conditional re-mappings, and are constructed as child items under their containing memory map, as seen in Figure 2.4. Memory remaps and the associated remap states were implemented in version 2.8 and their implementation is further explained in section 5.4.

The memory maps editor does not allow editing of AUB for memory remaps, because they use their parent memory maps AUB for the address increments. Instead, the memory remaps can change their assigned remap state, which describes the state when the memory remap becomes active. The remap states are managed in the component remap state editor. The remap state for a memory map is labeled as default, describing the memory map as the default remapping, when none of the assigned memory remaps are active. The default remap state of a memory map cannot be changed.

Each of the memory maps and memory remaps contains an editor for address blocks. Register editors are located within each of the address blocks and a field editor is located within each register. These editors can be accessed through the component editor tree by expanding the memory maps and their contents. Besides offering an editor for editing the contained items, the editors offer the possibility to edit the currently selected memory map item.

2.3.2 Views editor

This editor focuses on the IP-XACT views that are separated into hierarchical and non-hierarchical, flat views. A hierarchical view contains references to a hardware design configuration and design. A component utilizing an active hierarchical view is considered a hierarchical component. A hierarchical view references a top-level implementation view.

Views contain parameters and more specific module parameters. These are designed to help define the view, further increasing the re-usability of a component when implemented as a component instance. The module parameters are a Kactus2 extension to the IP-XACT standard.

One component can contain multiple views, which is useful in creating reusable component instances for a design, when differently behaving instances are required of the same component. Any of the views can be selected as the currently active view for the instance. For example, two instances of the same component are used in a design, with one of them using a hierarchical view, and the other a flat view. The hierarchical component instance can then contain child components of its own.

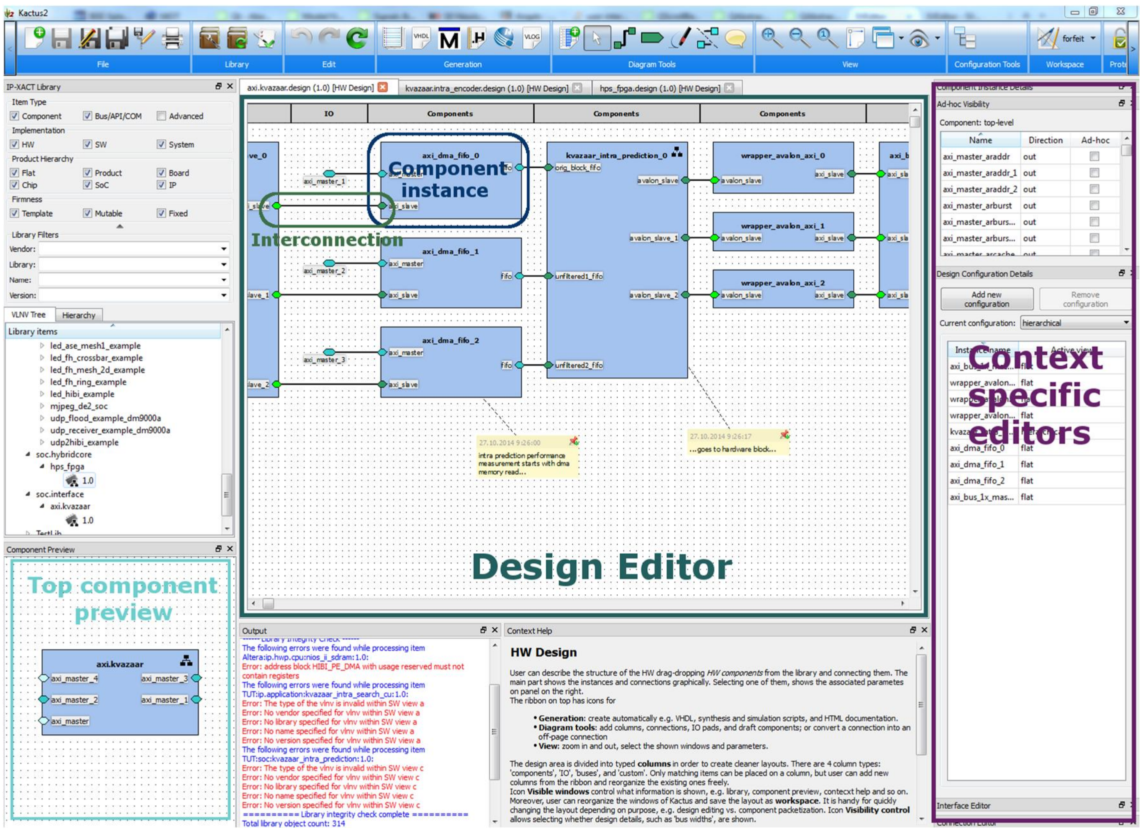
The views of a component are extended by using software and system views. These contain information regarding a software or system used in the component. The software views are associated with components containing CPUs and define application-independent software. As with software views, system views are also associated with components containing CPUs, with a difference in having a hierarchical reference to a system design that defines the mapping of software to hardware. There can be multiple system or SW views for a component, with a multiprocessor SoC being an example of such a component.

2.4 Design editor

The design editor is displayed in Figure 2.5. It shows a hierarchical view of the hardware or software components instances located within a design [2]. The main design view, show in the center of Figure 2.5, displays the area available for the design. In addition a list of context specific editors is shown. These editors consist of a component instance details editor, an ad-hoc-visibility editor, a design configuration details editor, an interface editor and a connection editor.

The structure of the design view is constructed from columns determining what component instances can be placed in each column, e.g. bus components to one column and hardware components to other. These columns can be filled with component instances created from the available components. Each component instance contains the bus interfaces contained within the component from which the instance is constructed from.

The context sensitive editors are not allowed to make changes to the component referenced by the component instance. The modifications are made to the component instances, not the underlying component. This is to ensure that a component referenced by two different component instances is structurally the same.



```
------ Library integrity check ------
The following errors were found while processing item
Altera:ip.hwp.cpu:nios_ii_sdram:1.0:
Error: address block HIBI_PE_DMA with usage reserved must not
contain registers
The following errors were found while processing item
TUT:ip.application:kvazaar_intra_search_cu:1.0:
Error: The type of the vlnv is invalid within SW view a
Error: No vendor specified for vlnv within SW view a
Error: No library specified for vlnv within SW view a
Error: No name specified for vlnv within SW view a
Error: No version specified for vlnv within SW view a
The following errors were found while processing item
TUT:soc:kvazaar_intra_prediction:1.0:
Error: The type of the vlnv is invalid within SW view c
Error: No vendor specified for vlnv within SW view c
Error: No library specified for vlnv within SW view c
Error: No name specified for vlnv within SW view c
Error: No version specified for vlnv within SW view c
========== Library integrity check complete ==========
Total library object count: 314
```

HW Design

User can describe the structure of the HW drag-dropping *HW components* from the library and connecting them. The main part shows the instances and connections graphically. Selecting one of them, shows the associated parametes on panel on the right.
The ribbon on top has icons for

- **Generation:** create automatically e.g. VHDL, synthesis and simulation scripts, and HTML documentation.
- **Diagram tools:** add columns, connections, IO pads, and draft components; or convert a connection into an off-page connection
- **View:** zoom in and out, select the shown windows and parameters.

The design area is divided into typed **columns** in order to create cleaner layouts. There are 4 column types: 'components', 'IO', 'buses', and 'custom'. Only matching items can be placed on a column, but user can add new columns from the ribbon and reorganize the existing ones freely.
Icon **Visible windows** control what information is shown, e.g. library, component preview, contecxt help and so on. Moreover, user can reorganize the windows of Kactus and save the layout as **workspace**. It is handy for quickly changing the layout depending on purpose, e.g. design editing vs. component packetization. Icon **Visibility control** allows selecting whether design details, such as 'bus widths', are shown.

Figure 2.5 *Design editor.*

Component instance details editor describes information of the component instance and the component referenced by the instance. Of notable importance is the configurable values editor, which displays the parameters contained within the component and the currently selected active view of the selected component instance. The displayed parameters can be given new values, thus specifying a component instance. The expressions created within the configurable element editor can refer to other configurable element values, not the parameter values of the contained component.

Ad-hoc visibility editor determines the visibilities of the ad-hoc ports of the selected component instance. Usually only bus interfaces are visible in a component instance, but determining an individual port as an ad-hoc makes it visible in the design editor. The ports themselves cannot be edited in the ad-hoc visibility editor.

Design configuration details editor allows the selection, creation and deleting of design configurations. Through this editor, the active views of the component instances located within the design can be changed. The design configuration details are always visible in the design editor, regardless of the selected item.

The interface view editor contains information regarding the currently selected bus interface. The editor describes the bus type, the bus abstraction type used in the interface, along with the name, description and the interface mode currently active for the bus interface. The interface view also contains a list of port mappings, describing the logical and physical names of each port.

The connection editor becomes active when an interconnection is selected within the design. The connection view contains information of the interconnection between two component instance interfaces. It displays the bus and bus abstraction type used in the currently selected connection, the connected interfaces of the connected component instances, the name and description of the connection and a list of the connected physical ports. The list of connected ports is made using the ports mapped in the connected bus interfaces.

2.5 Generators

Generators are used in Kactus2 to create files for documentation, simulation and implementation [2]. The generators are context sensitive, with the available generators determined by the contents of the currently open document. The activated generator reads the required data from the component or design and generates the file associated with the generator function.

The generators of Kactus2 are divided into two sub groups, Kactus2 generators and plugin generators. The Kactus2 generators list common, always present generators, useful for synthesis and fast prototyping on FPGA. The plugin generators are extensions to the Kactus2 generators, performing more specific tasks. Kactus2 compatible plugin generators can be added and unnecessary generators can be removed. Each plugin generator contains information on the name, version, vendor and license of the plugin.

2.5.1 Kactus2 generators

Kactus2 generators consist of common generators utilized in synthesis and fast prototyping of FPGA. These generators are considered useful for all of the users of Kactus2. In version 2.8, the common Kactus2 generators consist of the document generator, the ModelSim generator and the VHDL generator.

The Document generator generates a Hypertext Markup Language (HTML)-type documentation from the selected component, containing information regarding the IP-XACT data of the component itself and any component that has a component instance located within an active hierarchical view of the selected component. For each such component, a table is created for model parameters, Kactus2 attributes, general parameters, memory maps, ports, bus interfaces, file sets and views.

ModelSim generator generates a ModelSim makefile, used to compile all the needed VHDL-files into their respective libraries, so the top component can be simulated. ModelSim is used as a simulator for both ASIC and FPGA designs [27].

The VHDL generator generates a top-level VHDL file from the selected, hierarchical IP-XACT component [7], which contains information regarding the model parameters and the ports of the target component. The generator uses the component to create a structural, top-level entity, which is included in the component's meta-data. VHDL is used in synthesis and fast prototyping on FPGA [28].

2.5.2 Plugin generators

Plugin generators are extensions to the common generators of Kactus2, creating more specific files relating to functionalities not necessarily useful for every user of Kactus2. This is why all the compatible plugin generators can be added or removed through the settings menu. Currently Kactus2 supports the following plugin generators: Altera BSP generator, Makefile generator, MCAPI Code generator, Memory map Header generator, PADS part generator, Quartus Project generator and Verilog generator.

Creating a plugin generator for Kactus2 has been made relatively simple. Each of the plugin generators has its own folder in the structure of Kactus2. Within the folders are plugin-related files for controlling generator procedures. Besides these files, each of the plugins has functions for examining information regarding the generator. Additionally, the plugins contain information regarding their structure and other, build-related data.

Verilog Generator version 1.1 generates a top-level Verilog file, a hardware description language module for the selected hardware component. This file contains information of the parameters, interfaces, ports connected to the interfaces and ports not found in any of the interfaces of the component.

Altera BSP (Board Support Package) Generator version 1.0 generates board support for HW components containing a CPU and at least one SW view. The generated package contains an implementation of software for a given board that conforms to a given operating system.

Makefile Generator version 1.0 generates makefiles on all the parsed data of a system design containing a design configuration. The generated makefiles help with the compiling and linking of a program.

MCAPI Code Generator version 1.1 generates MCAPI code templates based on the metadata of the selected hierarchical SW component containing a system design. As a message-passing API, MCAPI captures communication and synchronization elements required for closely distributed embedded systems [29].

Memory Map Header Generator version 1.0 generates C-headers from memory maps of a component. The generator uses addresses of the memory map's address blocks and registers in the memory maps of connected components to create C-headers of local, global or system memory maps. The outcome of the memory map header generator depends on the target of the generation. If there is no design associated with the selected component, a local memory map header is generated. The local memory map header contains the address blocks of a local memory map located within an address space. A register definition is calculated through the use of an address blocks base address and the offset of a register. If the generator is run for a hardware design, a global memory map header is generated. This takes into consideration the instances contained within the design, and the interconnections between the interfaces of the instances. The possible bridge and channel components between the master interface and the slave interface are taken into account when calculating the values for the memory map header file.

System designs have their own version of memory map header generation. This file is created by gathering all the C and C++ files contained within the file sets of the components referenced by the instances within the hierarchical system design. The absolute path of the C and C++ files are included in the generated file, with any duplicates removed.

PADS printed circuit board [30] part generator version 1.0 generates PADS part files for the selected hardware component containing a chip hierarchy. This generator can only be run from the component editor of Kactus2. The generated files are used in the design and layout management of circuit boards.

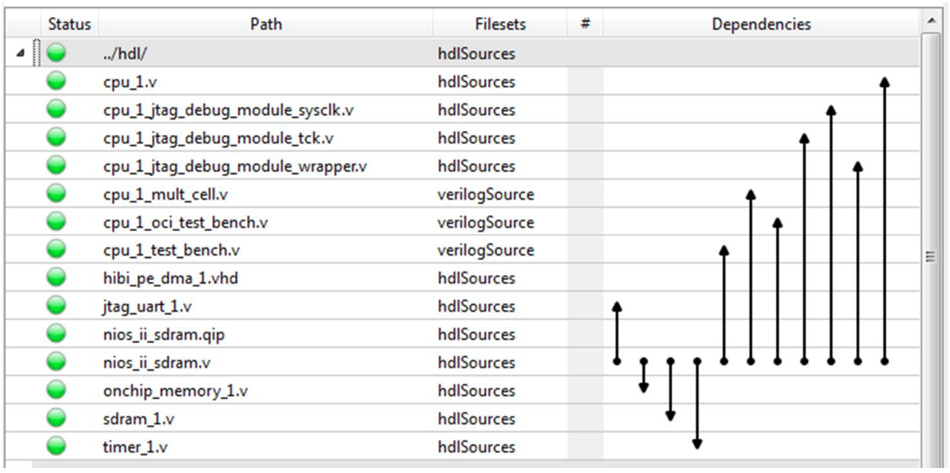
Altera Quartus Project Generator version 1.0 generates Quartus Project and Settings files from a hierarchical hardware design containing a design configuration. The Quartus Settings file takes all the included Verilog, VHDL, Quartus IP (QIP) and Synopsis Design Constraints (SDC) files from component instances located within the hierarchical design, while the created Quartus Project file contains information related to the top-entity from which the generator was run from. Both created files are usable in Quartus2 software.

2.5.3 Source analyzers

Source analyzers analyze the file dependencies of their respective files. The analyzers are useful in forming dependency chains between the analyzed files. The analyzers contained within Kactus2 are listed in Table 2.1.

Table 2.1 *Source analyzers of Kactus2.*

Analyzer	Function	Version
C / C++ Source Analyzer	Analyzes the file dependencies from C and C++ files	1.0
Verilog Source Analyzer	Analyzes the file dependencies from Verilog files	0.1
VHDL Source analyzer	Analyzes the file dependencies from VHDL files	1.0



Status	Path	Filesets	#	Dependencies
●	../hdl/	hdlSources		
●	cpu_1.v	hdlSources		
●	cpu_1_jtag_debug_module_sysclk.v	hdlSources		
●	cpu_1_jtag_debug_module_tck.v	hdlSources		
●	cpu_1_jtag_debug_module_wrapper.v	hdlSources		
●	cpu_1_mult_cell.v	verilogSource		
●	cpu_1_oci_test_bench.v	verilogSource		
●	cpu_1_test_bench.v	verilogSource		
●	hibi_pe_dma_1.vhd	hdlSources		
●	jtag_uart_1.v	hdlSources		
●	nios_ii_sdram.qip	hdlSources		
●	nios_ii_sdram.v	hdlSources		
●	onchip_memory_1.v	hdlSources		
●	sdram_1.v	hdlSources		
●	timer_1.v	hdlSources		

***Figure 2.6** Source analyzers in Kactus2 fileSets editor.*

Figure 2.6 depicts the results of a file dependency analysis within the file sets editor of Kactus2. The status column shows whether the file has been changed or not after the previous dependency analysis: a green circle represents that the file has not been changed, a yellow circle shows files with changed contents and a red circle depicts files with changed dependencies. The path column describes the relative path to the folder containing the files as well as the names of the included files. Filesets column describes the containing file set of the item within the component.

The dependencies column shows the results of the analysis. The dependencies between the files are shown with the help of arrows pointing to the files that a specified file needs. In Figure 2.6, the nios_ii_sdram.v is dependant of all the other files with the exception of the nios_ii_sdram.qip.

2.5.4 Import Plugins

The import plugins are used in Kactus2 to import various aspects of a Component from a designated file. These tools provide an easy way to incorporate the IP-XACT standard used in Kactus2 to the legacy IP-components owned by the user. The import plugins of Kactus2 are listed in Table 2.2.

***Table 2.2** Import plugins of Kactus2.*

Import plugin	Function	Version
Quartus II Pin Import	Imports ports from a Quartus pin file	1.1
Verilog importer	Parses a Verilog input, setups an RTL view and creates model parameters and ports from the input	1.2
Verilog include import	Parses a Verilog input and creates model parameters from Verilog defines	1.0
VHDL importer	Parses a VHDL input, sets up an RTL view and creates generics and ports	1.1

2.6 Summary

With the tools presented in this chapter, Kactus2 can improve the reusability of IP components. The component editor provides functionality for managing the items contained within a single component, while the design editor helps in constructing hierarchical or non-hierarchical system designs of component instances. Both of the editors can use the tools provided by Kactus2.

3. DEVELOPMENT ENVIRONMENT

Kactus2 is implemented in C++ using Microsoft Visual Studio 2010 with Qt version 5.2.0 plugin and Visual Assist. Other tools used in the development include Redmine for project management, SourceForge for source code repository. Various different tools, such as Microsoft Word, Microsoft PowerPoint and LaTex are used for documentation. Kactus2 runs on Microsoft Windows and Linux.

3.1 Microsoft Visual Studio 2010

Microsoft Visual Studio 2010 version 10.0 Service Pack 1 is an Integrated Development Environment (IDE), with a set of development tools used to compile C++ in Kactus2. Visual Studio is used as the main development environment for Kactus2.

The project visualization system of Visual Studio consists of solutions and projects [31], containing items representing the references, data connections, folders and files that are used by the software. Multiple solutions can be contained within the solution editor. In Kactus2, this system is used to contain the IP-XACT models, plugin projects and the main Kactus2 project in the same solution as different projects for ease of reference. The projects are displayed in the Solution Explorer, which provides access to the contained files.

Different editors provide the main functionality of the Visual Studio. They are context sensitive, depending on the file or document being examined [31]. The main editors used in Kactus2 are the text editor, a basic word processor of the IDE, and the code editor, providing basic functionality of a source code editor. In addition to these, the help files of Kactus2 are done using a HTML designer, which shares the basic functionality of the source code editor. The HTML designer provides additional enhancements needed to support HTML editing. Other editors are provided in the Visual Studio, but they are not used in Kactus2.

The build and debug tools of Visual Studio allow configurations to the components to build, the methods for building and the target platform of the build. The compiler of Visual Studio detects compile-time errors such as incorrect syntax checking and type mismatches. These errors are then displayed to the user in the output window of Visual Studio. The debugger of Visual Studio assists in finding and correcting the errors. In the break mode of the debugger, local variables and other relevant data can be examined. [31]

3.1.1 Visual Assist

Visual Assist is an extension to Microsoft Visual Studio developed by Whole Tomato Software [32], with a principal idea being in improving the IntelliSense and syntax highlighting of Visual Studio. Additional enhancements include code generation and refactoring tools together with configurable features for Visual Studio. The features ease the use of Visual Studio.

Visual Assist improves code generation by providing spell checking and word filling. Added together with the ability to refactor code, they make code generation faster and more precise. In general, Visual Assist helps to reduce the complexity and improve the readability of the code, without affecting its external behavior. Additionally, debugging time of native C or C++ is shortened using Visual Assist. The Visual Assist Step Filter and step over methods in argument lists help find possible errors found within the code. [32]

Visual Assist provides a searchable query to the current solution, called the Visual Assist View. This query can be used to search a specific file within the currently open solution faster and easier than navigating the solution explorer. Additionally, the Visual Assist View can display a list of all the included files of a selected document.

3.2 Qt 5.2.0

QT is a cross-platform application and UI framework used in device creation and application development. It is used in many devices and applications [33], such as in the Inflight Entertainment by Panasonic Avionics, which uses Qt with C++ and QML to enhance the in-flight experience. Zühlke uses Qt in their DLC Pro to enable a multi-touch UI utilizing the Qt API. The cross-platform support and flexibility of Qt are the key components of the Eykona Wound Measurement tool, developed by The Eykona Technologies together with the support of Qt Company [34].

In Kactus2, Qt is used as an extensive library to cover different design methods of cross-platform development for the Microsoft Windows and Linux. Kactus2's user interface is constructed using the interface libraries provided by Qt. The libraries use the system's resources to draw the user interface components. This makes Kactus2 look more native regardless of the operating system it is being run on.

3.2.1 Model / View / Delegate

Model-view-delegate system is Qts own version of model-view-controller (MVC) paradigm [33]. An MVC is a framework for separated presentation, a clear division between the perceived domain objects and the presentation objects of a user interface. The domain objects cover the model part of the system, while the presentation is made of both the view and controller models [35].

The model / view / delegate principle has been used in the editable tables of Kactus2. This principle allows the management of the relationships between data and the way it is presented to the user and allows greater customization possibilities.

The model of the system contains the dynamically managed data of the items. To locate an item and access its data within a model, the views and delegates use model indexes, with the view displaying the indexed data to the user. When an item is edited, the delegate communicates with the model directly using the model indexes, allowing the data to be represented to the user through tables or editors. [33]

3.2.2 Signals and Slots

The signal & slot system is an observer method used in Qt. The source of the observer method is replaced by the class emitting the desired signal. The observer itself is replaced by the class that connects to this signal via a slot. Both of them are sub-classes of QObject class.

The signals & slots of Qt help keep the user interface and systems of Kactus2 functional. Generally, it is desired to have the user interface components communicate with each other. This communication can be achieved through signals and their connecting slots. The signal and slot system helps in the construction of the user interface, when a Qt widget must be notified of a change in another widget [33].

A signal is emitted when an event takes place. This can be either an inbuilt or custom event, such as a keyboard button being pressed or an explicitly called emit-function. A slot is a function that is called when the signal is emitted. The selected signal has to be connected to the target slot for this slot function to be called. [33]

3.2.3 Creating a tree model with Qt

Tree models are only allowed in Qt as part of a QTreeWidget class. However, if the need arises to construct a customized tree model, the QAbstractItemModel class can be sub-classed to form a fully customizable tree model.

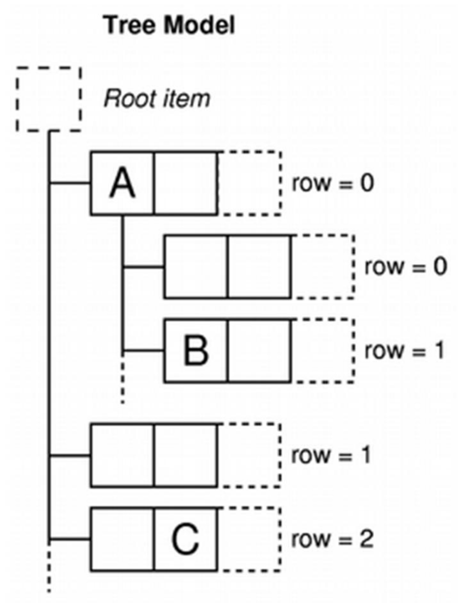


Figure 3.1 *The tree model structure in Qt [33].*

Figure 3.1 shows a tree model in Qt. The root item is the parent item of all the items within the tree. Items A and C are child items of the root item, while item B is a child of item A. In order to create a tree model from a QAbstractItemModel, the model requires implementation for the index and parent functions [33]. These two functions govern the indexing of the tree model, where the parent describes a parent index of an item. The items under the root start their own sub-trees, with indexing for their child items starting again from 0, as shown in Figure 3.1. Root items of the tree have an invalid parent, while child items have the parent index of their parent item. Root items are placed in the tree first, after which each of the referenced components parameters is checked for the correct placement within the tree. This creates the desired, expanding tree model.

3.3 Redmine

Redmine is a flexible cross-platform and cross-database project management web application, written using a Ruby on Rails framework. Redmine includes the tools required in project management, such as issue tracking, a roadmap application and monitoring tools. [36]

The Redmine installed at Tampere University of Technology covers information of the development of Kactus2 from version 2.3 onward. The Redmine contains information of the project and links to the installer and source files of Kactus2, located within its own source repository page. It also functions as a platform for reporting any issues found by the users. The issues can then be notified to the developers, who can implement them in the development flow of the application.

Important features of Redmine used in the Kactus2 project are the roadmap, the issue list and the Gantt chart. The features are used to track the development time used for the different features included within Kactus2. The roadmap visualizes the current time usage concerning the development of all the versions of Kactus2. The issue list keeps track of the features still in development, the features required and any bugs found in the usage of Kactus2. Gantt keeps track of the overall overlapping and time requirements of the different tasks.

3.4 Source forge

The code repository of Kactus2 is currently located in Source Forge [37]. Source Forge is a free web-based open source code repository. It offers a platform for development, download, review and publishing of open source software. This allows the different versions of Kactus2 for the supported operating systems to be downloadable from a single web page.

Source Forge used to contain the project management information of Kactus2. As of version 2.3, this has been moved to Redmine, due to the better project management features found in RedMine.

4. REQUIREMENTS

This section describes the issues for the development of version 2.8. The issues of versions 2.7 and 2.7.x are taken into account if the new issues depend on them. The issues are created by the Kactus2 development team as well as the user community. Redmine is used to manage and track all the suggested features and discovered issues.

Version 2.8 focused on parameter usage. In addition to the issues covered here, the version included parameter importing, parameter propagation, improving the configurable element value editor further, hierarchical saving and including model parameters to hierarchical views.

The issues are listed with either of two identifiers: a feature or a bug. A feature describes a new aspect added to Kactus2, while bug describes an existing feature that does not work in the expected way. After the identifier, there is an issue automatically generated in Redmine. The title of the issue is given after the number, describing in short what the issue is about. The details regarding the implementation of the issues are given in chapter 5.

4.1 Feature #9: Memory map editor should show the edited item

The memory map editor is constructed in a way to show information only of the contained parts of the memory map. For example, a memory map editor only displays its contained address blocks and an address block editor only displays its contained registers. However, it is required that information regarding the edited item should be made available to the user [38]. This removes the time for navigating between the different editors and gives visibility on the current status of the edited component, and according to Nielsen [25], visibility of the system status increases the usability of the application.

4.2 Bug #121: Show errors is not visible for all library items

Library items in Kactus2 are coloured red if Kactus2 finds errors in the contained IP-XACT structure. When checking the integrity of the library, the errors found and the identification of a component is printed in the output of Kactus2 to help the user find the locations of the errors.

Nevertheless, currently some errors do not change the colour of a component with errors in the contained IP-XACT structure. This makes the option to show errors in the

Kactus2 library unavailable in the context menu of the component. Known cases are [38]:

- A file in one of the component’s filesets is not found
- A hierarchical reference in a software view is found to be invalid
- A hierarchical reference in a system view is found to be invalid.

It is required that errors relating to the IP-XACT structure of a component are visible within the IP-XACT library tree of Kactus2.

4.3 Feature #171: parameter usage

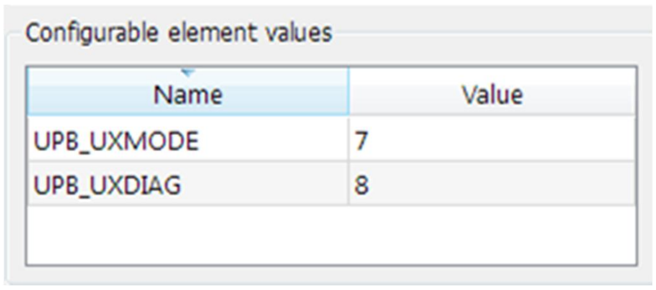
This is a general issue, consisting of subtasks of parameter usage. Parameter usage consists of expressions within parameters, formatting the contained expressions, handling references in parameters, the definition of and references to global component parameters, parameter importing from e.g. Verilog files and design level configuration.

4.3.1 Feature #177: Design level configuration

The design level configurable elements value editor is shown in Figure 4.1. The editor contains name value pairs of parameters of a component instance. New configurable element values can be created and the values can be edited.

The parameters are identified by name but they are not guaranteed to match the parameters of the component referenced by the instance. It is required that the configurable element values need to be fully synchronized with the improved parameter definitions, and the ability to create arbitrary configurable element values for the containing component instance should be removed [38].

The resolve type of the parameter determines its usability in the configurable element editor [38]. Parameters of resolve value “immediate” or those that have no resolve value assigned are not editable. However, these are still displayed in the configurable element editor to show as a possible reference when constructing expressions in the editor. Other resolve values of parameters, generated or user generated, allow the editing of the configurable element value in the configurable element editor of the component instance.



Configurable element values

Name	Value
UPB_UXMODE	7
UPB_UXDIAG	8

Figure 4.1 *Configurable elements editor.*

4.4 Feature #180: Signal based configuration

This is a general issue, consisting of dynamically configuring the module and its contents using module inputs. The configuration values affect the register maps of the module and its bit availability.

For example, an expression within the offset of a register contained within an address block of a memory map refers to a parameter. This parameter has a resolve value of "user" and can thus be modified in the configurable elements editor. The user can then configure the memory maps contained register by giving this parameter a new value in the configurable element values editor of the component instance.

4.4.1 Feature #192: RemapStates and memoryRemap

Kactus2 is missing the option to create and modify memory remaps and remap states. These are given as a description of memory map remapping in the IP-XACT. Editors for both IP-XACT items are required in the component editor of Kactus2 [38]. The placement of the new editors in the component editor tree is important. With the memory remaps being a part of the memory maps of a component, they should be placed in close presence to the memory maps. The remap states are a separate entity with only the memory maps using them.

Memory remaps are used as part of a memory map in the IP-XACT standard. The remaps describe conditional and additional memory map items for the memory map. The conditions are described by the remap states, with each memory remap containing one remap state.

The memory remap schema from the IP-XACT standard is displayed in Figure 4.2. The state refers to the remap state that activates the items located within a memory remap. The memory remap contains a list of memory map items, address blocks, memory banks and subspace maps. Name group contains information regarding the identification of a memory remap, while the isPresent defines the presence of the memory remap in the containing document.

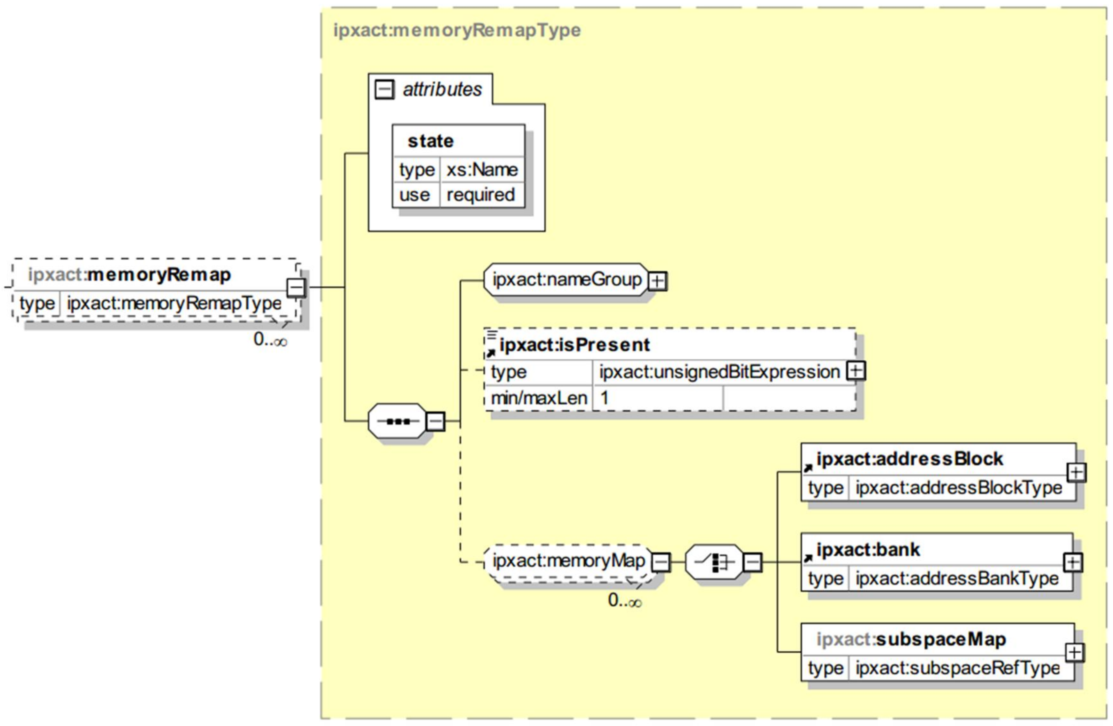


Figure 4.2 *IP-XACT memory remap schema [4].*

Figure 4.3 Displays the IP-XACT schema for the remap states. The remap state contains a unique name, contained within a universal IP-XACT element nameGroup, as well as an optional list of remap ports. The remap ports are selected from a list of ports contained within the component, as displayed by the contained attribute portRef. These form the condition that activates the remap state. When a remap state is activated, the memory remap containing the selected state becomes active.

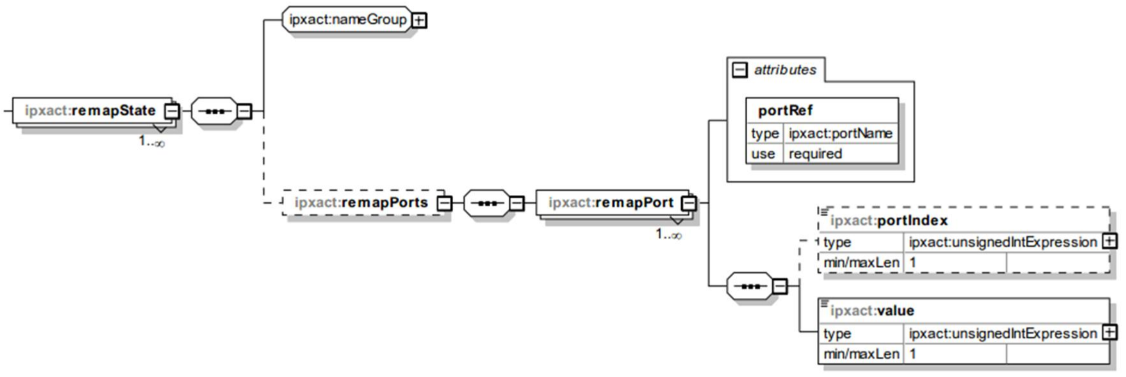


Figure 4.3 *IP-XACT remap state schema [4].*

4.4.2 Feature # 193: Tabs for memory remaps

An editor for the memory remaps is required to be placed near the memory map to which the remap belongs to. Additionally, the memory map visualization in Kactus2 does not handle the visualization of memory remaps. The memory map visualizer should draw the memory remaps for easy reference and comparison. [38]

The memory remap editor should be easily distinguished as being connected to the containing memory map. Therefore the placement of the new memory remap editor should be connected to the memory map containing the remap. This is to be done as a tab for the memory remaps inside the memory map editor. Thus the memory remaps are closely connected to a memory map, preventing confusion regarding to which memory map does a memory remap belong to.

4.5 Feature #206: Scrollable ribbon

Kactus2 uses a toolbar ribbon to display the different tools available. The old toolbar ribbon from version 2.6 is shown in Figure 4.4. The available tools have not changed between the versions, with the exception of the added view configuration tool and the hierarchical save tool.

In the version 2.6 toolbar, the icons are small enough to accommodate placing them on two rows. Additionally, the space reserved for each icon is wide. This is so that the name of the tool can be displayed below the icon. Overall, this leads to poor use of space in the tool bar.

The width of the toolbar is set according to the width of the Kactus2's window. When the window is too small to display the available tools, the toolbar groups shrink into smaller, selectable sections. The tools are selectable from the compressed sections. In Figure 4.4, this is shown for the edit, generation, diagram tools, view and protection tool groups. While convenient, this hides information regarding the placement of the tools, which according to Nielsen [25] hinders user friendliness.

To improve the usability of Kactus2, the tool bar ribbon is changed to show icons of all the currently usable tools. If the icons require more space than is available due to the size of the current window, the ribbon should become scrollable horizontally. [38]

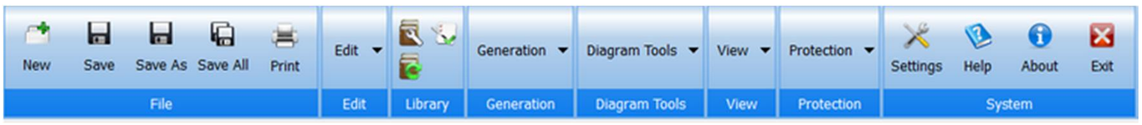


***Figure 4.4** Toolbar of Kactus2 version 2.6.*

The scrolling should be made available to use with the mouse wheel or by using the left and right buttons of the contracted ribbon. The buttons should only be visible when the ribbon is too large to show on the current screen.

4.6 Feature #214: Support parameter IDs in Verilog generation

In the Verilog files generated by the Verilog generator of Kactus2, parameter references used in the expressions of parameter values are displayed with their respective universally unique IDs (UUIDs). The UUIDs of the parameters are used as a way to tell parameters with same names apart, but they are not visible in Kactus2. Since the IDs are not human readable, they must be converted to the names of the referenced parameters. The places where the parameter references are not displayed properly are [38]:

- The parameter values for the top component
- Declarations of ports
- The parameter assignments of component instances

4.7 Feature #216: Support parameter IDs in HTML document generation

The HTML file generated by the document generator of Kactus2 contains information regarding the selected component. In this generated document, the values in fields that refer other parameters of the component are shown as the UUIDs of the referenced parameters.

The UUIDs of the parameters are not human readable and must be converted to parameter names [38]. This conversion must be made in every field where an expression is possible to be given.

4.8 Feature #217: Support parameter IDs in generated header files

The generated memory map header files contain defined registers, the register positions calculated using the connecting master and slave interfaces. However, the calculations do not account for expressions containing parameter references. Because of this, the positions containing expressions are calculated wrong, giving false definitions to the registers in the generated files.

The UUIDs displayed in the generated memory map header files must be converted into more human readable names of the referenced parameters [38]. The calculation of the register defines must also be corrected to accept parameter referencing in any part of the registers definition, such as address blocks base address and the registers offset. Additionally, the calculation of the register definitions in the containing memory map header files should display information to help in the checking of the register definitions.

4.9 Feature #221: Add a vendor extension for Kactus2 version

Information of the version of Kactus2 used to create a component is not currently available. This is needed in the future development of an IP-XACT migration tool [38]. The version information helps in determining the components that are missing some sections or extensions that are incorporated to the later versions of Kactus2. An example of the features is the addition of the UUID to new components and parameters created in Kactus2, which was indoctrinated before version 2.8. With the help of the version recognition tools, the components and their containing parameters that do not have UUIDs could be found and possibly given one.

The version number should be made visible in the component's XML-file. This is to be an extension to the IP-XACT standard of the component, and so would be best placed in the vendor extensions of a component.

4.10 Feature #222: Locked column in parameter editor

All the parameters and model parameters of a component contain multiple fields displaying the different aspects of a parameter, causing the parameter editor to become very wide. A wide editor needs a scrollbar to properly display all of its contents. However, when the name of the parameter is not displayed, the contents of the editor do not specify which parameter is currently being edited, thus resulting in confusion and unintentional mistakes in changing the values of a different parameter than the one which was intended to be changed.

The editor should be made to contain a locked column i.e. a column that is always visible to the user regardless of the position of the scrollbar. Locking the name column of the parameter editor ensures that information regarding the edited parameter is always visible to the user [38], preventing any unwanted changes to the parameters.

The locked column can also be extended to cover other editors. In addition to all parameter, model parameter, module parameter and view parameter editors contained within the component editor, the port editor should have the ability to lock the port number and the name columns.

4.11 Feature #236: Group configurable element values

In the design editor of Kactus2, the configurable element values of a component instance are placed into the configurable elements value editor in a mixed order, affecting the readability and reuse of the configurable element values. The configurable elements must be grouped according to their location within the referenced component [38]. This requires the changing of the configurable element editor into a tree model, displaying the element location as a branch.

Only the common parameters of the referenced component and the parameters of the currently selected active view of a component instance should be included in the configurable element editor. Using groups in the configurable element editor increases the cohesion between a component instance and the referenced component.

4.12 Feature #237: Product level view configuration tool

The active views of a design can be configured through the use of a component design editor. Some of the generators and plugin generators of Kactus2 use the views to generate their files. However, it is required that the generators can be given another set of views separate from the active views selected for the component instances in a design. Thus, a tool for configuring the active views of the selected design is needed [38].

In this new view configuration tool, the hierarchical composition of the design, with the component instances and their contained hierarchical components should be visible to the user. The new view configuration should be stored in the design configuration xml-file referenced by the currently selected active view. In addition to the view configuration editor, the generators that handle views should be modified to take this configuration into account when one exists.

5. IMPLEMENTATION

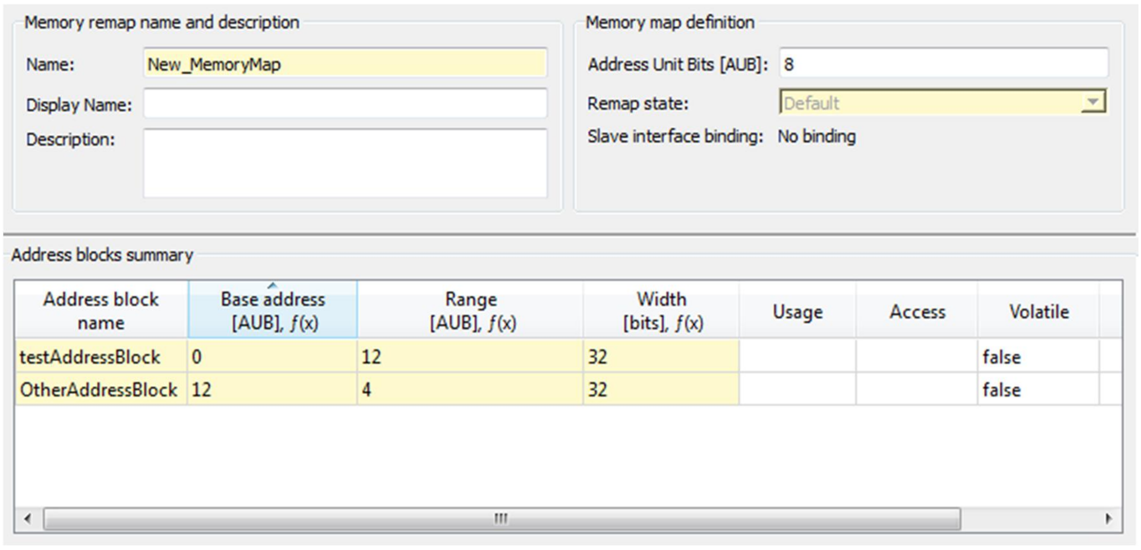
This chapter describes the implementations to fulfil the requirements given in chapter 4. Optional solutions are given where applicable.

5.1 Memory map editor should show the edited item

The memory map editor is used to edit the contents of a component’s memory maps with sub-editors for the items contained within the memory map: the address block editor, the register editor and the field editor. A memory map editor contains the editable address blocks, an address block editor contains the editable registers and the register editor contains the editable bit fields.

Figure 5.1 displays the implemented memory map editor. The name group box is placed to the top left corner of the editor to allow fast access to the identifying elements of the item being edited, as well as to create consistency between the different editors.

In addition to the name group editor, the memory map editor contains fields for editing IP-XACT memory map data. These fields are placed near the name group, on the right column of the editor as seen on Figure 5.1. This ensures that the user connects the fields to the item being edited. The rest of the available editor space is taken up by the editor for managing the editable items belonging to the memory map. This is done in each of the sub-item editors contained within the memory maps editor, i.e. the address block editor, register editor and the field editor.


Memory remap name and description

Name: New_MemoryMap
Display Name:
Description:

Memory map definition

Address Unit Bits [AUB]: 8
Remap state: Default
Slave interface binding: No binding

Address blocks summary

Address block name	Base address [AUB], ƒ(x)	Range [AUB], ƒ(x)	Width [bits], ƒ(x)	Usage	Access	Volatile
testAddressBlock	0	12	32			false
OtherAddressBlock	12	4	32			false

Figure 5.1 *Memory map editor.*

Kactus2 is constructed in such a way that all the IP-XACT related information regarding the component is found in one place, in the xml-file of the component. When an editor is opened, the data contained within it is retrieved from this file. This refreshing of the editable items helps in keeping the different editors of the same item synchronized.

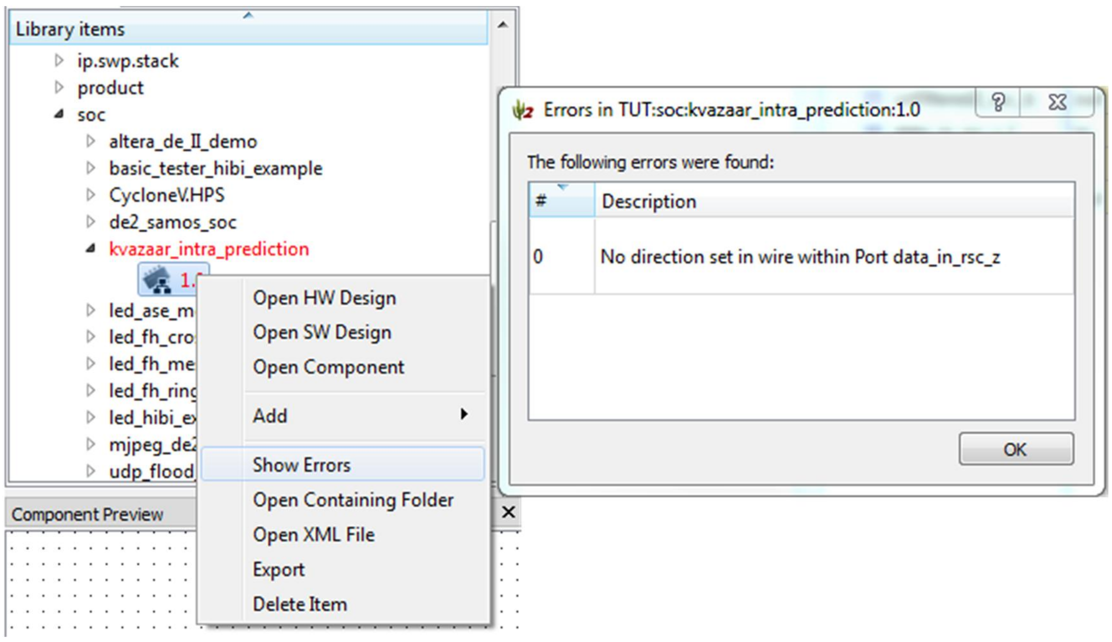
It is important to keep the consistency of the editors together to increase the usability of Kactus2. According to Nielsen [25], consistency within different parts of the application increases recognition and user friendliness. To follow this consistency, the sub editors of a memory map are changed to look like the upgraded memory map editor, with their information fields located in two columns above the editor for the contained items.

5.2 Errors are not visible for all library items

Figure 5.2 shows how a component containing errors in its IP-XACT information should be visible in the IP-XACT library of Kactus2.

IP-XACT library of Kactus2 contains a context menu for each component, through which the use can access the “Show Errors” functionality. Selecting this gives a list of all the IP-XACT-related errors found within the component, as displayed in Figure 5.2.

Kactus2 strives to confirm the IP-XACT validity of the components within the given library paths by examining all the components for errors. Any errors found in the invalid components are displayed in the output window.



The following errors were found:

#	Description
0	No direction set in wire within Port data_in_rsc_z

Figure 5.2 *Component with errors in the Kactus2 library view.*

The IP-XACT standard describes a file as an optional element of a file set. This unbounded file or directory contains a name describing the absolute or relative path to the actual file. According to the semantic consistency rules of the IP-XACT standard, this name sub-element must result in a string containing a valid uniform resource identifier (URI). The file validity in Kactus2 checks only for the emptiness of the name element. In the IP-XACT standard, checking of the existence of the file contained within the name element requires external knowledge. Since the validity checking in Kactus2 checks for the IP-XACT validity of components without external knowledge, a missing file is not displayed in the error listing of a component.

Software views and system views are extensions to the IP-XACT model of Kactus2, and are based on the IP-XACT view. Both have mandatory fields for a hierarchical reference, which must be a VLNV reference to a system design.

According to the IP-XACT standard, a view can have a reference to a design or a design configuration, which is inside a view's instantiations. The instantiations contain components, design and design configurations for all views. All the items inside the instantiations have a reference to an IP-XACT item in the form of a VLNV [4].

According to the semantic consistency rules of the IP-XACT standard, a VLNV used to refer an IP-XACT document must match an existing IP-XACT document. The VLNV validity checking in Kactus2 only checks that the different VLNV-fields are not empty. This is why an invalid, non-empty hierarchical reference in a system or software view is not displayed in the error listing of a component.

However, if a Kactus2 determined mandatory field is given as empty, it will be displayed as an error message in the output of Kactus2. This is to tell the user that something is wrong regarding the inspected component. And although the library itself does not display any IP-XACT related errors, the component being inspected is colored red. This is to warn the user of a malfunctioning component. If the component is opened, the component editor displays all the faulty locations that are found in the component, including the missing files and the inappropriate design references in software and system views.

5.3 Parameter usage

In Kactus2, parameters are utilized in e.g. constructing structures and memory regions. Parameters can refer other parameters and thus create expressions. However, global references are not allowed, in order to ensure the reuse of each IP block. Parameters can be redefined for each instance of a component. This is explained in section 5.3.1. Parameters contain six different attributes used to define values that a parameter can be given. [39]

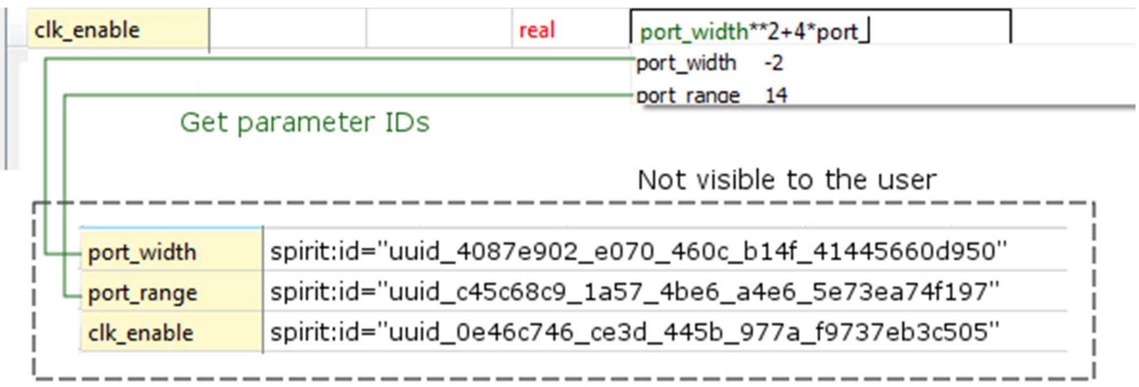
Since an IP component can contain thousands of parameters, managing them becomes an issue. Kactus2 uses an expression editor to control, validate and evaluate parameters and their contained expressions immediately upon insertion [39].

```
<spirit:parameter type="int" usageCount="2">
  <spirit:name>port_width</spirit:name>
  <spirit:value spirit:id=
    "uuid_4087e902_e070_460c_b14f_41445660d950">-2
  </spirit:value>
</spirit:parameter>
<spirit:parameter type="real">
  <spirit:name>clk_enable</spirit:name>
  <spirit:value spirit:id=
    "uuid_0e46c746_ce3d_445b_977a_f9737eb3c505">
     uuid_4087e902_e070_460c_b14f_41445660d950**2+4*
     uuid_4087e902_e070_460c_b14f_41445660d950+4
  </spirit:value>
</spirit:parameter>
<spirit:parameter type="int">
  <spirit:name>port_range</spirit:name>
  <spirit:value spirit:id=
    "uuid_c45c68c9_1a57_4be6_a4e6_5e73ea74f197">14
  </spirit:value>
</spirit:parameter>
```

Program 1. *XML showing a parameter containing an expression. [39]*

Program 1 displays the XML of the parameters of a component within Kactus2. Three parameters are given, with parameter *clk_enable* containing an expression referring parameter *port_width*. In the XML file, the references are contained as UUIDs of parameters. When the expressions are displayed to the user the UUIDs are changed to the names of the corresponding parameters.

During the construction of an expression within a parameter, the expression editor of Kactus2 offers a list of possible parameter references for an incomplete word, with each selection consisting of a name-value pair of a parameter. The UUID of the selected parameter is inserted to the created expression. This is displayed in Figure 5.3.



clk_enable			real	port_width**2+4*port

port_width	-2
port_range	14

port_width	spirit:id="uuid_4087e902_e070_460c_b14f_41445660d950"
port_range	spirit:id="uuid_c45c68c9_1a57_4be6_a4e6_5e73ea74f197"
clk_enable	spirit:id="uuid_0e46c746_ce3d_445b_977a_f9737eb3c505"

Figure 5.3 *The expression selection.[39]*

The available list of referable parameters depends on the parameter finder given to the expression editor. The parameter finder is further explained in section 5.6.

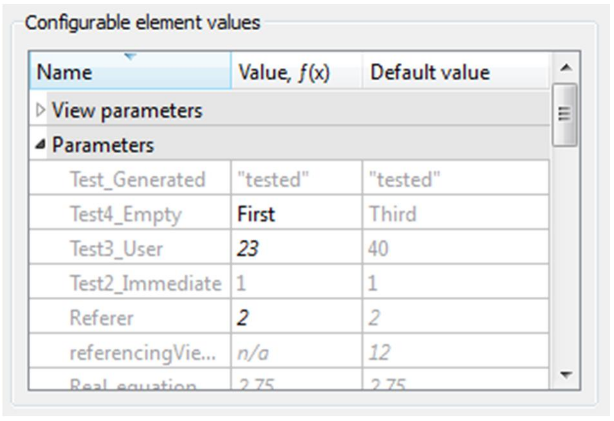
5.3.1 Design level configuration

According to the IP-XACT standard, the configurable element values determine the configuration for a component instance.

The previous editor allowed arbitrary creation and removing of configurable element values, which is against the IP-XACT standard. Thus the context menu containing modifications to the configurable elements was removed from the editor.

Figure 5.4 displays the improved configurable element editor. Regarding the requirement 4.3.1, the configurable elements value editor was changed to contain three columns: name, value and default value. The name column displays the name of the referenced component parameter or the location of the parameter within the referenced component, the value column displays the current value of the configurable element and the default value column displays the value of the parameter inside the referenced component.

By default, all of the fields located within the editor are non-editable. To allow the editing of a component parameter, the resolve value of the parameter must be set to “user” or “generated”. A resolve value of “user” defines the value to be specified by user input, while a resolve value of “generated” specifies the parameter value as set by a generator. “Immediate” or unspecified resolve value indicates that the specification of the parameter is made in the containing component.


Configurable element values

Name	Value, ƒ(x)	Default value
▷ View parameters		
◢ Parameters		
Test_Generated	"tested"	"tested"
Test4_Empty	First	Third
Test3_User	*23*	40
Test2_Immediate	1	1
Referer	*2*	*2*
referencingVie...	*n/a*	*12*
Real_equation	2.75	2.75

Figure 5.4 The improved configurable element editor.

It is worth noting that the configurable element values can be given as expressions. As the expression editor in Kactus2 uses a parameter finder to locate the value associated with the given parameter name, the configurable element editor requires two expression editors: one for the default values, to solve references made to component parameters, and the other for the configurable element values, to solve references made to the configurable element values or the parameters of the component referencing the design.

Configured values are stored as vendor extensions inside the design configuration xml-file referenced by the selected hierarchical view and stored alongside the component instance to which they belong. The storage format is the referenced ID followed by the given configurable element value. Only those configurable elements which are given a new value inside the configurable element editor are saved.

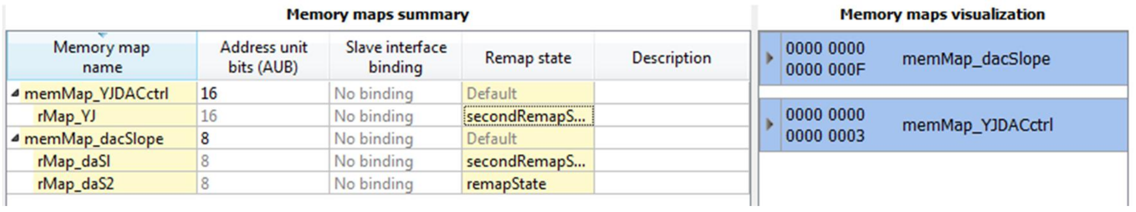
Regarding the requirement 4.11, the configurable elements editor needs to show information relating to the placement of the parameter inside the referenced component. This is to preserve the consistency of the system which, according to Nielsen [25], helps the user in identifying and relating the parameters to the referenced component. The method to show the containing component sections was to change the table model into a tree model. However, Qt does not provide a customizable tree model [33], so one had to be inherited from QAbstractItemModel class.

5.4 Memory remap and remap states

According to the IP-XACT standard, a memory map can contain memory remaps. The memory remaps are used to describe additional memory map contained items in a specific remap state.

To incorporate the new IP-XACT elements, the memory remap and the remap state, the component editor tree was modified. The memory map editor was modified to contain the memory remaps, and the remap states editor was placed as its own entity, with the added option to disable it in the settings, if the user so desires.

Figure 5.5 displays the new combined memory editor. Combing the memory map and remap editors caused a modification to the structure of the component editor tree regarding the memory maps. A new layer was added for the memory remaps between the memory map and address block layers, enhancing the connection between a memory map and its memory remaps. A default memory remap was added as a level for the original memory map, separating it from the memory maps remaps while keeping them linked together.



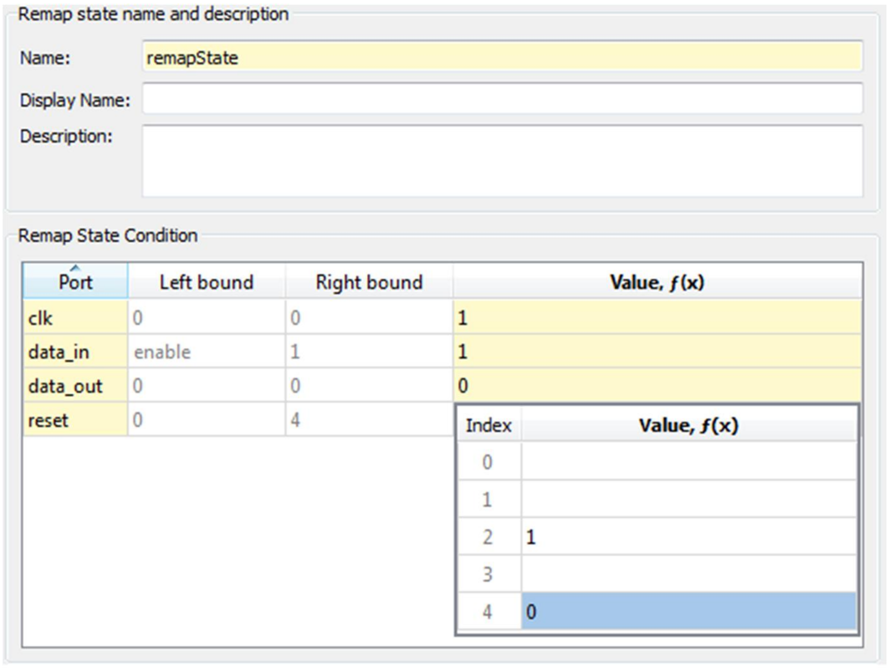
Memory maps summary

Memory map name	Address unit bits (AUB)	Slave interface binding	Remap state	Description
memMap_YJDACctrl	16	No binding	Default	
rMap_YJ	16	No binding	secondRemapS...	
memMap_dacSlope	8	No binding	Default	
rMap_daSl	8	No binding	secondRemapS...	
rMap_daS2	8	No binding	remapState	

Memory maps visualization

0000 0000	0000 000F	memMap_dacSlope
0000 0000	0000 0003	memMap_YJDACctrl

Figure 5.5 *The combined editor for memory maps and memory remaps.*



Remap state name and description

Name: remapState

Display Name:

Description:

Remap State Condition

Port	Left bound	Right bound	Value, ƒ(x)
clk	0	0	1
data_in	enable	1	1
data_out	0	0	0
reset	0	4	

Index	Value, ƒ(x)
0	
1	
2	1
3	
4	0

***Figure 5.6** Remap state editor.*

Figure 5.6 displays the developed remap state editor. A remap port has a mandatory value to determine when the selected port activates the remap state. For a vectored port, this value is given for all the desired indexes of the port. To determine the vectored status of a port, the remap state editor calculates the width of the port using the left and right bounds taken from the xml-file of the containing component. If this width is greater than one, the port is vectored and can be given values for each index of the vector. In Kactus2, giving values for each of the indexes in the port is done by constructing an array editor for the value. This array editor can be seen in Figure 5.6, opened for the remap port *reset*.

The requirement given in section 4.4.2 suggested a separate tab editor within a memory map. However, this creates unnecessary navigation within the memory map editor, which is against the desired user experience in Kactus2. A separate tab for memory remaps was thus rejected.

Another suggestion for the memory remap editor was placing it alongside the memory maps. In this solution, the memory remaps could be constructed as items belonging to a selected memory map. Both the memory maps and memory remaps can then be displayed alongside each other, making it easier to cross-reference the basic functionalities of memory maps and memory remaps. This was selected as the method for creating and modifying memory remaps.

In this solution, new memory remaps can be created through a context menu associated with a selected memory map. Since this is not immediately obvious, the memory remap abilities of a memory maps editor are explained in the context help file.

Combining the managing of the memory maps and the memory remaps into the same editor requires reforming the editor from a table view into a tree view. The memory maps are the child items under the root of the tree, while each memory remap is constructed as a leaf node to the containing memory map.

Some of the fields in the memory maps editor are only editable for memory maps, while others are editable only for memory remaps. The AUB can only be changed for a memory map, but changing its value causes all the associated memory remaps to change their AUBs accordingly. A remap state can only be given to a memory remap. Default is shown as the remap state for a memory map. This can be observed in Figure 5.5. For implementation of a single memory map within a memory map editor, see section 5.1.

5.5 Scrollable ribbon

The toolbar used in Kactus2 displays the icons and names of the available tools. If the window is smaller than the required space of the toolbar, some of the tool groups are minimized into a selection group, through which the tools can still be accessed.

Figure 5.7 displays the developed replacement for the toolbar. The icons displaying the names of the tools took up a lot of space in the ribbon. This has been revised by enlarging the tool icons and removing their names. The names are still visible in the tool tips of the tools if the user hovers the cursor over one of them. The name was removed to reduce the screen space required by the toolbar and to emphasize the recognition and connection of the tool icons and the associated functions. This, according to Nielsen [25], increases the consistency of the application and provides a more pleasant user experience.

Minimizing the tool groups into selections provides more space for the tool bar ribbon, but at the same time causes more navigation within the toolbar. This needles navigation can be eliminated by removing the group minimizing. Thus the user can always see the available tools within a group without having to open the tool selection. However, this creates a new problem with the ribbon: displaying all the available tools requires a lot of horizontal screen space. This can be solved by allowing the tool ribbon to scroll left and right.

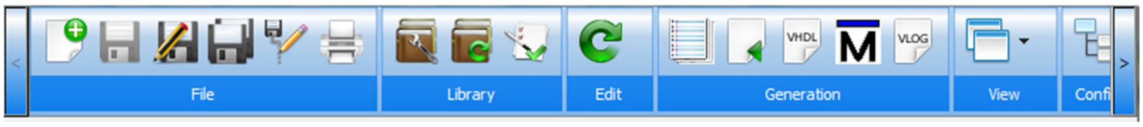


Figure 5.7 *Scrollable toolbar.*

The scrolling can be achieved through the use of the mouse wheel, or by using the two buttons placed on the sides of the ribbon, as shown in Figure 5.7. The scrolling of the ribbon is achieved by catching the mouse wheel events. This is then translated as horizontal movement of the scroll bar with a calculated step provided by the movement of the mouse wheel.

It is possible that the user cannot, or does not wish to use the mouse wheel to scroll the toolbar. To accommodate this situation, two buttons are situated in the left and right sides of the toolbar. The buttons scroll the toolbar respectively left and right by the amount needed to show the next tool bar group. The buttons are enabled only when the width of the scrollbar exceeds the width of the window and the corresponding button is disabled if the tool bar is scrolled all the way to the last group on either side of the toolbar.

5.6 Parameter ID support

The introduction of expressions and parameter referencing to values introduced problems to the generators of Kactus2. Parameter referencing is achieved through the use of UUIDs of the parameters. The IDs are not human readable, so the preferred way is to change them into the names of the referenced parameters, thus requiring changes to the handling of parameters in all of the affected generators.

The basic structure for incorporating parameter references to a generator is to add an expression formatter class. This class replaces ID references found in the given expression with name references using a parameter finder class. The parameter finder handles the finding of parameters within a specified structure. The parameter finder is divided based on where the parameter references are found. This parameter finder division is shown in Figure 5.8.

The ComponentParameterFinder finds parameters from the currently active component. All of the parameter IDs in the currently examined expression are checked for a matching parameter name. It is used in forming expressions within a single component.

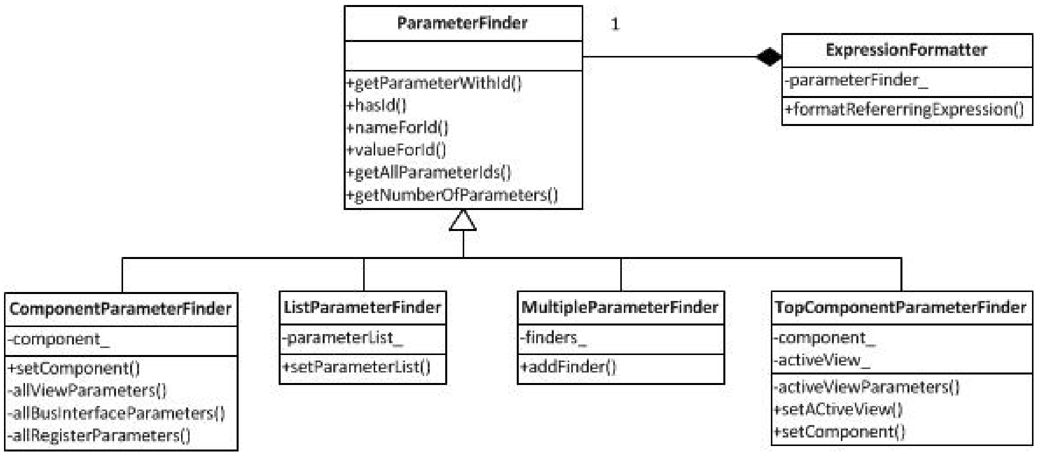


Figure 5.8 *Class diagram of the ParameterFinders used in ExpressionFormatter.*

ListParameterFinder finds parameters from a given list used to define the group of available parameters. This is used in the expression editors of the configurable values of a configurable elements editor.

MultipleParameterFinder finds parameters using multiple parameter finders. It is used in the file import tool and the component instance editor. In the file import tool it is used to find parameter values from the current component and the parameter values of the imported component. In the component instance editor it is used to contain finders for the configurable element values and the parameter values of the component containing the design housing the component instance.

TopComponentParameter finder finds parameters from the top component of a hierarchical view. Unlike the component parameter finder, this parameter finder locates only the view parameters and module parameters related to the currently active view. It is used together with a component parameter finder to form a multiple parameter finder for the configurable element editor.

All of the parameter finders use a given ID to find the desired parameter. The expression formatter then uses this name to change the referenced IDs into the names of the referenced parameters, thus displaying the expressions in a more human readable way.

5.6.1 Verilog Generator

To find the references made in the generated Verilog file, an expression formatter containing a component parameter finder is used. The parameter finder helps in the formatting of all the expressions used in the parameter values together with the left and right bounds of ports contained within the component.

In addition to the parameters and ports of the top component, the Verilog generator lists the connections, the connected interfaces and the default values of ports. The default values of the ports are formatted using component parameter finders created for each of the components.

5.6.2 HTML Document Generator

As the file generated by the document generator can consist of multiple components, an expression formatter factory is used. This allows the dynamic creation of expression formatters using component parameter finders. The finders are then assigned a component referenced by the component instance in question. The selected finder is used in formatting the expressions in the component referenced by a component instance and the configurable element values of the component instance.

5.6.3 Memory map Header Generator

The memory map header generator is divided into three modes: the local, the global and the system memory map header generators. These three modes differ in their handling of memory map header generation, although the basic operating principle is the same.

The local memory map header generator generates a memory map header for a single component, with a component parameter finder used for the expression formatter. The generator takes the address blocks used in the local memory maps of the contained address spaces. The address blocks with usage value “reserved” or “memory” are defined with start and end locations. With a usage value of “register” or “undetermined”, the registers belonging to the address block are defined.

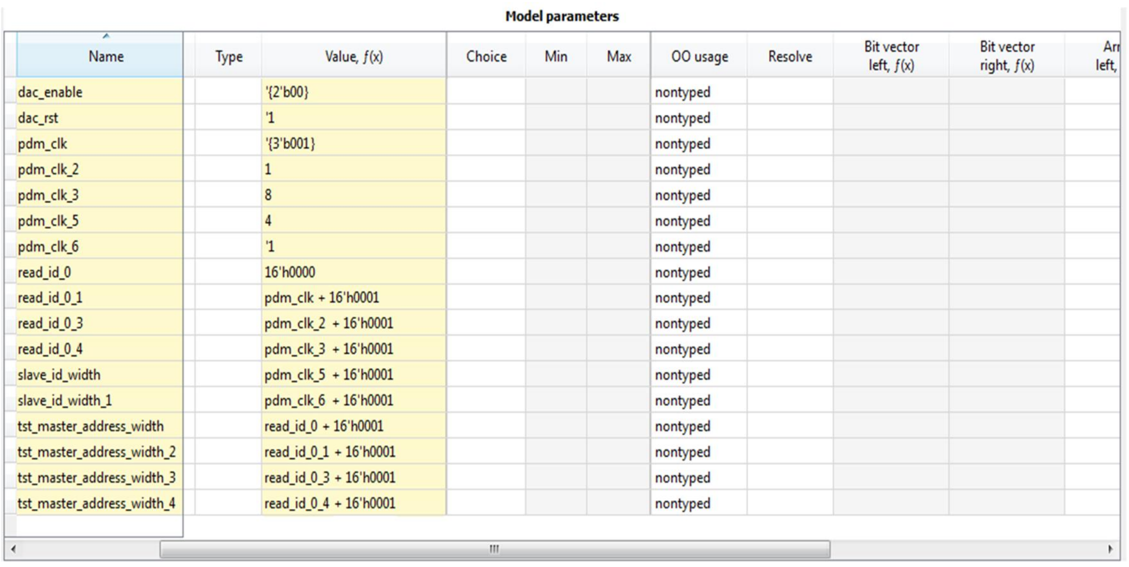
A global memory map header generator generates header files for each connected master interface belonging to the component instances of the design involved in the generation. Each of the files contains the definition of the registers associated with the slave interface that the master interface is connected to, taking into account any bridge or channel components between the two interfaces.

The system memory map header generator generates header files for each component instance in a system design. The files contain the C and C++ files contained within the file sets of the target component referenced by the component instance.

5.7 Locked column in parameter editor

A locked column depicts a column, which stays visible to the user regardless of the current position of the scrollbar. This helps in crowded tables to recognize which item the user is currently editing.

By default, Qt does not provide this kind of functionality in tables. The solution presented in Figure 5.9 is based on an existing example in the documentation of Qt [33]. It involves two views of the same table with one view stacked on top of the other. One view contains the desired locked columns, while the other contains the rest of the table. This creates an illusion of a table with one column locked in place.



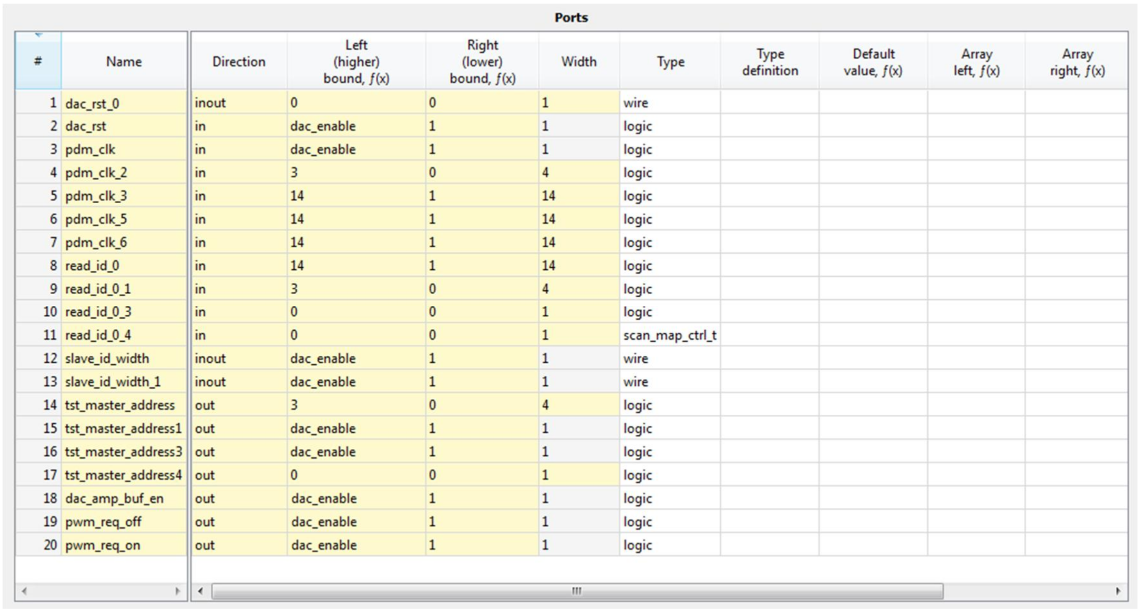
Model parameters

Name	Type	Value, ƒ(x)	Choice	Min	Max	OO usage	Resolve	Bit vector left, ƒ(x)	Bit vector right, ƒ(x)	Arr left,
dac_enable		'{2'b00}				nontyped				
dac_rst		'1				nontyped				
pdm_clk		'{3'b001}				nontyped				
pdm_clk_2		1				nontyped				
pdm_clk_3		8				nontyped				
pdm_clk_5		4				nontyped				
pdm_clk_6		'1				nontyped				
read_id_0		16'h0000				nontyped				
read_id_0_1		pdm_clk + 16'h0001				nontyped				
read_id_0_3		pdm_clk_2 + 16'h0001				nontyped				
read_id_0_4		pdm_clk_3 + 16'h0001				nontyped				
slave_id_width		pdm_clk_5 + 16'h0001				nontyped				
slave_id_width_1		pdm_clk_6 + 16'h0001				nontyped				
tst_master_address_width		read_id_0 + 16'h0001				nontyped				
tst_master_address_width_2		read_id_0_1 + 16'h0001				nontyped				
tst_master_address_width_3		read_id_0_3 + 16'h0001				nontyped				
tst_master_address_width_4		read_id_0_4 + 16'h0001				nontyped				

Figure 5.9 *Table view with a locked column.*

Moving between the two tables can be accomplished via capturing the key events related to both tables. After capture, the event is passed to the other table if the current selection is in the locked column or the column next to it. After the event is passed, the other table can then place the selection in the correct position within its own table.

An alternative solution to the locked column is given in Figure 5.10. Instead of having the two table views with one on top of the other, the views are set side by side. A small separator is placed between the views for visual identification and to allow the size of the views to be changed. This solution gives the user the impression of a locked column.



Ports

#	Name	Direction	Left (higher) bound, ƒ(x)	Right (lower) bound, ƒ(x)	Width	Type	Type definition	Default value, ƒ(x)	Array left, ƒ(x)	Array right, ƒ(x)
1	dac_rst_0	inout	0	0	1	wire				
2	dac_rst	in	dac_enable	1	1	logic				
3	pdm_clk	in	dac_enable	1	1	logic				
4	pdm_clk_2	in	3	0	4	logic				
5	pdm_clk_3	in	14	1	14	logic				
6	pdm_clk_5	in	14	1	14	logic				
7	pdm_clk_6	in	14	1	14	logic				
8	read_id_0	in	14	1	14	logic				
9	read_id_0_1	in	3	0	4	logic				
10	read_id_0_3	in	0	0	1	logic				
11	read_id_0_4	in	0	0	1	scan_map_ctrl_t				
12	slave_id_width	inout	dac_enable	1	1	wire				
13	slave_id_width_1	inout	dac_enable	1	1	wire				
14	tst_master_address	out	3	0	4	logic				
15	tst_master_address1	out	dac_enable	1	1	logic				
16	tst_master_address3	out	dac_enable	1	1	logic				
17	tst_master_address4	out	0	0	1	logic				
18	dac_amp_buf_en	out	dac_enable	1	1	logic				
19	pwm_req_off	out	dac_enable	1	1	logic				
20	pwm_req_on	out	dac_enable	1	1	logic				

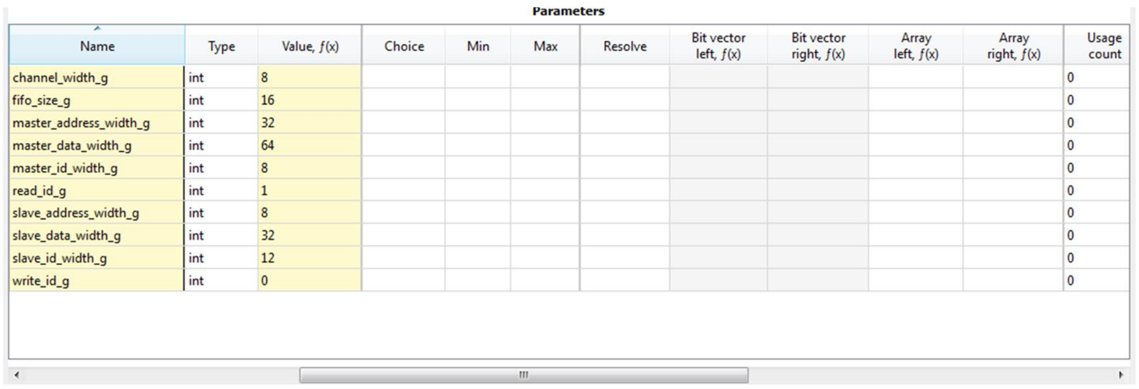
Figure 5.10 *Alternative locked column.*

Both of the approaches achieve the locked column effect. However, of them contain issues regarding the selection model of a view. The view with the locked column must change selection to the one containing the rest of the model when the selection moves from the locked column to the right. The same applies when changing selection to the left in the view containing the rest of the model. In addition, selecting items from both of the tables causes problems. If the selection is begun in the top view, no selection can be made on the view behind, because the views are constructed as separate entities and thus do not have a common focus.

A third solution for the locked column is displayed in Figure 5.11. It incorporates a remarkably different approach compared to the solutions presented above. Instead of having two views showing different parts of the model, only one view is used. The scrollbar hides and reveals the columns that are not currently visible by examining its current position. This creates a scrolling view with the selected column locked in place.

This approach eliminates some of the problems of selection, but incorporates new ones, mainly related to the correct scaling of the scrollbar and the selection of multiple fields. The scrollbar scaling compared to the diminishing table is easily solved through setting the step of the scrollbar to accommodate the size of the table.

Selecting multiple fields moves the viewport of the table. If the view is moved off screen, the locked column will disappear from view, which is not desirable. Since the behavior of the scrollbar has been changed, a similar multiple selection is needed to get the disappeared columns to reappear.



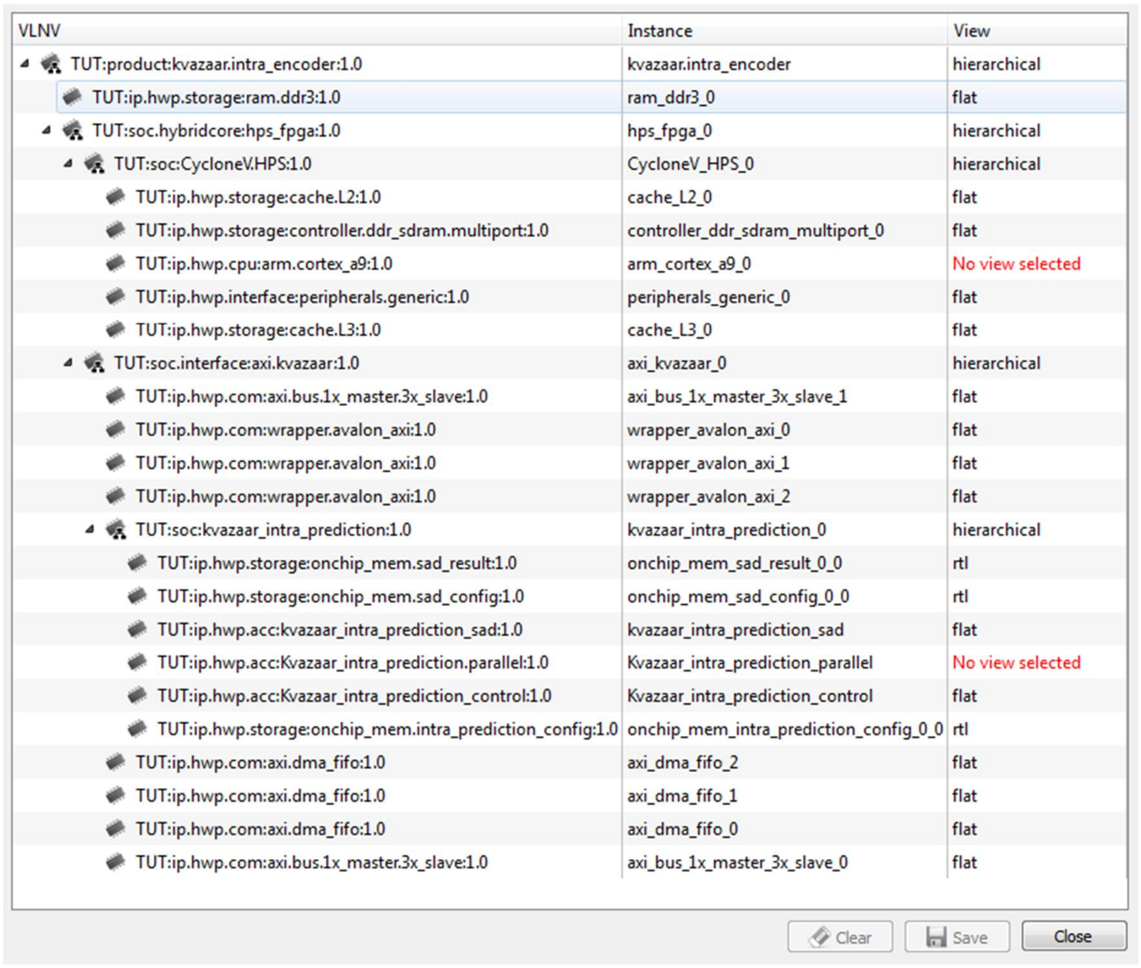
Parameters

Name	Type	Value, ƒ(x)	Choice	Min	Max	Resolve	Bit vector left, ƒ(x)	Bit vector right, ƒ(x)	Array left, ƒ(x)	Array right, ƒ(x)	Usage count
channel_width_g	int	8									0
fifo_size_g	int	16									0
master_address_width_g	int	32									0
master_data_width_g	int	64									0
master_id_width_g	int	8									0
read_id_g	int	1									0
slave_address_width_g	int	8									0
slave_data_width_g	int	32									0
slave_id_width_g	int	12									0
write_id_g	int	0									0

Figure 5.11 *Another alternative locked column.*

5.8 Product level view configuration tool

The designed view configuration tool is displayed in Figure 5.12. The left column displays VLNV of the component referenced by the instance with an icon displaying whether the selected view for the component is hierarchical or non-hierarchical. A hierarchical column can be expanded to show all the component instances contained within its design. The middle column displays the instances located within the design associated with the selected views. The right column displays the currently selected view for the component instance. Only the values in the view column are editable in the view configuration tool.



VLNV	Instance	View
TUT:product:kvazaar.intra_encoder:1.0	kvazaar.intra_encoder	hierarchical
TUT:ip.hwp.storage:ram.ddr3:1.0	ram_ddr3_0	flat
TUT:soc.hybridcore:hps_fpga:1.0	hps_fpga_0	hierarchical
TUT:soc:CycloneV.HPS:1.0	CycloneV_HPS_0	hierarchical
TUT:ip.hwp.storage:cache.L2:1.0	cache_L2_0	flat
TUT:ip.hwp.storage:controller.ddr_sdram.multiport:1.0	controller_ddr_sdram_multiport_0	flat
TUT:ip.hwp.cpu:arm.cortex_a9:1.0	arm_cortex_a9_0	No view selected
TUT:ip.hwp.interface:peripherals.generic:1.0	peripherals_generic_0	flat
TUT:ip.hwp.storage:cache.L3:1.0	cache_L3_0	flat
TUT:soc.interface:axi.kvazaar:1.0	axi_kvazaar_0	hierarchical
TUT:ip.hwp.com:axi.bus.1x_master.3x_slave:1.0	axi_bus_1x_master_3x_slave_1	flat
TUT:ip.hwp.com:wrapper.avalon_axi:1.0	wrapper_avalon_axi_0	flat
TUT:ip.hwp.com:wrapper.avalon_axi:1.0	wrapper_avalon_axi_1	flat
TUT:ip.hwp.com:wrapper.avalon_axi:1.0	wrapper_avalon_axi_2	flat
TUT:soc:kvazaar_intra_prediction:1.0	kvazaar_intra_prediction_0	hierarchical
TUT:ip.hwp.storage:onchip_mem.sad_result:1.0	onchip_mem_sad_result_0_0	rtl
TUT:ip.hwp.storage:onchip_mem.sad_config:1.0	onchip_mem_sad_config_0_0	rtl
TUT:ip.hwp.acc:kvazaar_intra_prediction_sad:1.0	kvazaar_intra_prediction_sad	flat
TUT:ip.hwp.acc:Kvazaar_intra_prediction.parallel:1.0	Kvazaar_intra_prediction_parallel	No view selected
TUT:ip.hwp.acc:Kvazaar_intra_prediction_control:1.0	Kvazaar_intra_prediction_control	flat
TUT:ip.hwp.storage:onchip_mem.intra_prediction_config:1.0	onchip_mem_intra_prediction_config_0_0	rtl
TUT:ip.hwp.com:axi.dma_fifo:1.0	axi_dma_fifo_2	flat
TUT:ip.hwp.com:axi.dma_fifo:1.0	axi_dma_fifo_1	flat
TUT:ip.hwp.com:axi.dma_fifo:1.0	axi_dma_fifo_0	flat
TUT:ip.hwp.com:axi.bus.1x_master.3x_slave:1.0	axi_bus_1x_master_3x_slave_0	flat

Figure 5.12 *View Configuration Tool.*

The view configuration tool allows a selection of views to be used in the generators of Kactus2. A view can be selected through the right column, from which a list of views is made available. The selectable views are contained within the component referenced by the component instance. If the selected view is hierarchical, the icon on the left column is changed to show this and all the component instances located within the design connected to the view are placed under the component instance. If no view is selected, or a non-existing view is selected, the text is coloured red.

The save button allows the currently selected configuration to be stored in the vendor extensions of the design configuration referenced by the view. The button is enabled only when the selected view configuration is valid, i.e. all the component instances have a valid view selected.

When the tool is activated, it checks for any existing view configuration. If a view configuration is found, it constructs the tree according to it. The stored view configuration can be removed through the use of the clear button, enabled only when a view configuration exists in the referenced design configuration.

The view configuration tool can be used either in a design or a component. In a design, the tool constructs a hierarchical tree consisting of the currently active view in the design editor. When it is used in a component, the tool shows only the component with no view selected. The user can then select the view for the top component and if the selected view is hierarchical, a hierarchical tree is constructed.

To accommodate the new view configuration tool, the tool bar ribbon is extended to contain a new group, the Configuration tools. This group will be used to contain tools related to configuring the generator related specifics of the selected component, component instances or design. In version 2.8, only the view configuration tool is located within the group.

View configurations can be used in the generators supported by Kactus2. If a running generator finds a stored view configuration in the design configuration, the generator will use the views contained within it instead of using the currently selected active views of the design. In version 2.8, Quartus Project generator has been modified to accept the new view configurations.

6. ANALYSIS

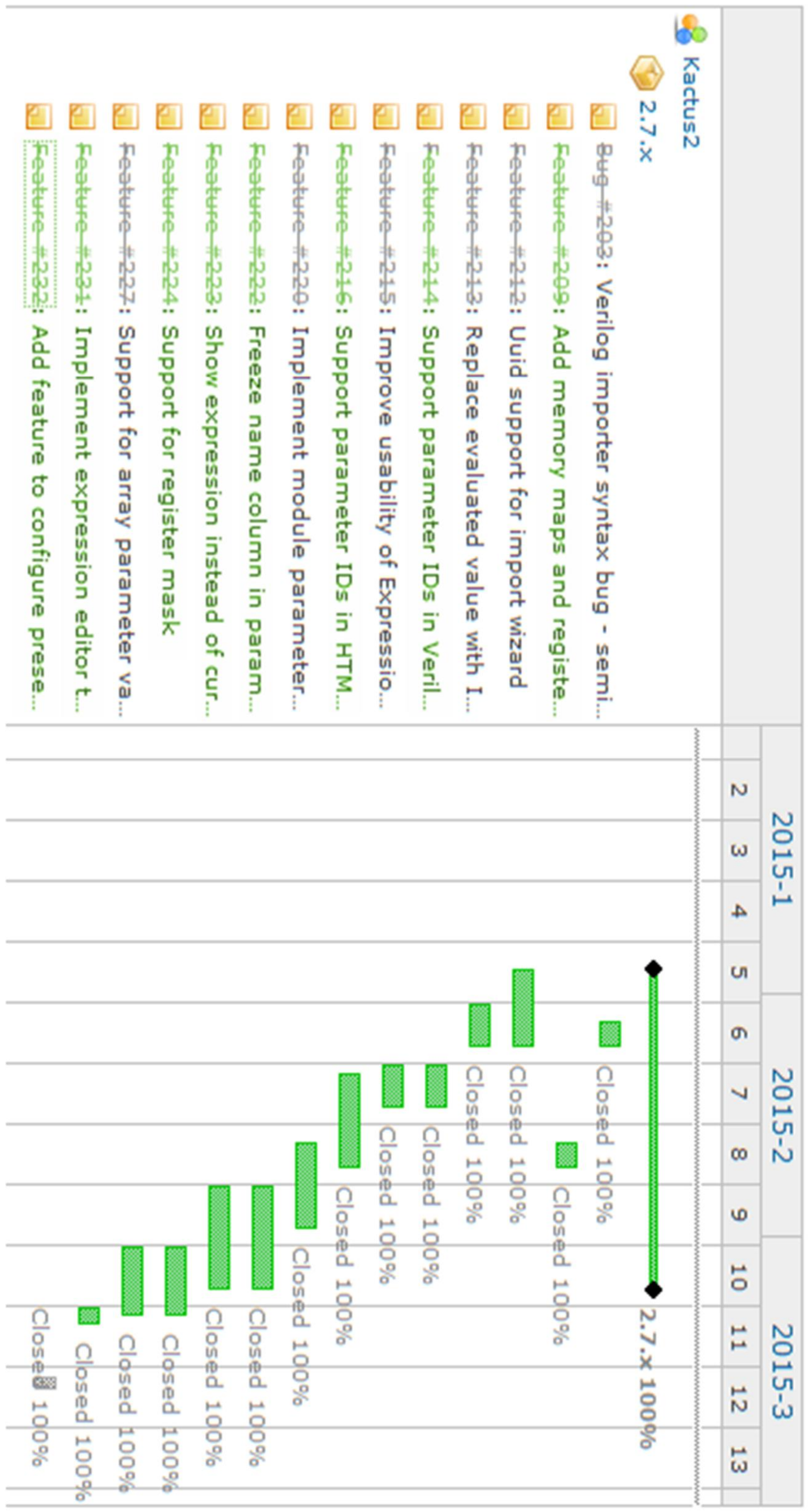
The development of Kactus2 version 2.8 took approximately 600 hours, and consisted of about 16000 lines of code. The Gantt chart for the previous version 2.7.x is shown in Figure 6.1, while the chart for version 2.8 is shown in Figure 6.2. Issues marked in green have been the main interest in this thesis. The implemented features consisted of the following:

- Upgrading the memory map editor to display information regarding the item being edited.
- Incorporating memory remap and remap state editors to the component editor.
- Error correction within the IP-XACT library of Kactus2.
- Upgrading the design level configurable element value editor to allow the editing of parameters contained within the referenced component.
- Improving the toolbar.
- Adding support for expressions and parameter references to the generators.
- Changing the table editors to accommodate possible locked columns.
- Integrating a version number to the components created using Kactus2.
- Incorporating a product level view configuration tool.

Table 6.1 displays the estimated time used and the estimated lines of code for each implementation. The time estimates are taken from the Rednub page [38]. It can be observed that the new features took more development time compared to improving the existing ones.

Table 6.1 *Estimated time spent for each issue.*

Identifier	Feature	Time spent (h)	Lines of code
Feature #9	Memory map editor shows edited item	20	400
Bug #121	Show errors in all library items	4	100
Feature #177	Design level configuration	32	900
Feature #192	RemapStates and memoryRemap	27	1000
Feature #193	Tabs for memory remaps	55	800
Feature #206	Scrollable ribbon	30	300
Feature #214	Support parameter IDs in Verilog generation	5	200
Feature #216	Support parameter IDs in document generation	32	1000
Feature #217	Support parameter IDs in generated header files	37	1300
Feature #221	Vendor extension for Kactus2 version	5	100
Feature #222	Locked column in parameter editor	15	300
Feature #236	Group configurable element values	13	500
Feature #237	Product level view configuration tool	50	1400
Total		325	7400


Kactus2

2.7.x

- ~~Bug #203~~: Verilog importer syntax bug - semi...
- ~~Feature #209~~: Add memory maps and registe...
- ~~Feature #212~~: Uuid support for import wizard
- ~~Feature #213~~: Replace evaluated value with I...
- ~~Feature #214~~: Support parameter IDs in Veril...
- ~~Feature #215~~: Improve usability of Expressio...
- ~~Feature #216~~: Support parameter IDs in HTM...
- ~~Feature #220~~: Implement module parameter...
- ~~Feature #222~~: Freeze name column in param...
- ~~Feature #223~~: Show expression instead of cur...
- ~~Feature #224~~: Support for register mask
- ~~Feature #227~~: Support for array parameter va...
- ~~Feature #231~~: Implement expression editor t...
- ~~Feature #232~~: Add feature to configure prese...

Figure 6.1 *Kactus2 Gantt chart for version 2.7.x [38].*

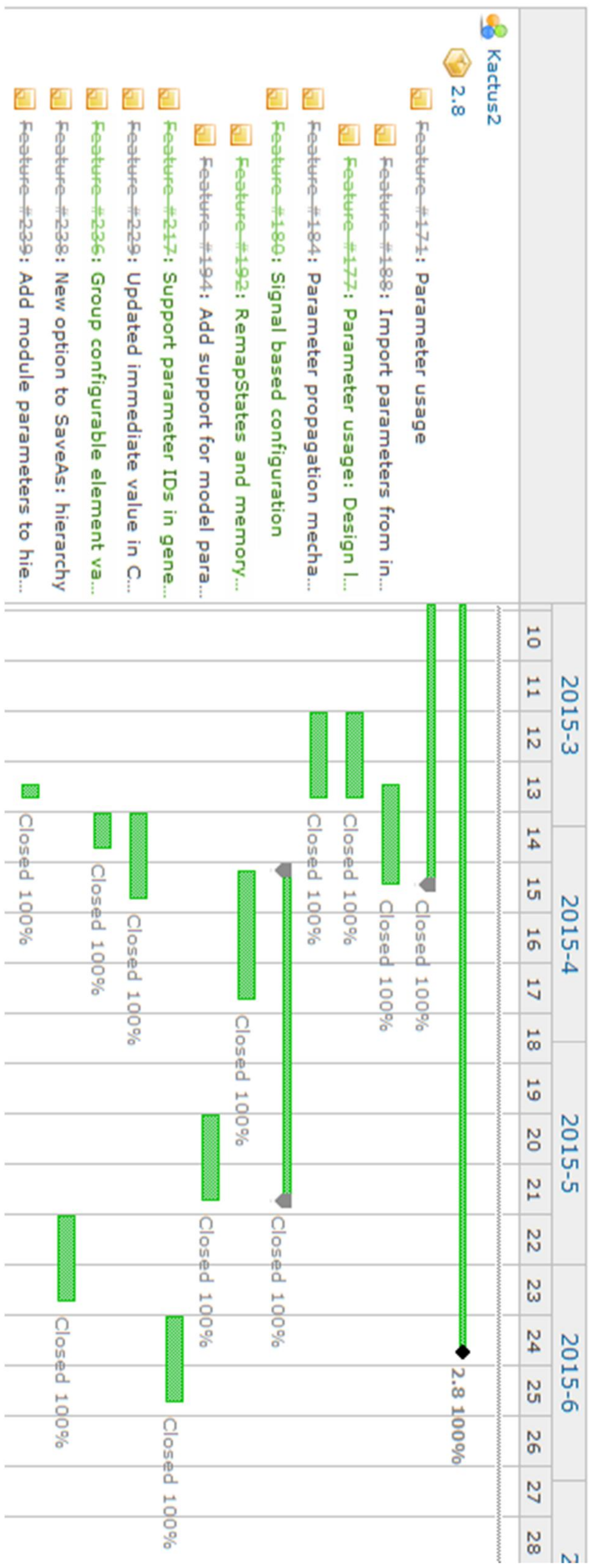

Kactus2
2.8
~~Feature #171~~: Parameter usage
~~Feature #188~~: Import parameters from in...
~~Feature #177~~: Parameter usage: Design l...
~~Feature #184~~: Parameter propagation mecha...
~~Feature #180~~: Signal based configuration
~~Feature #192~~: RemapStates and memory...
~~Feature #194~~: Add support for model para...
~~Feature #217~~: Support parameter IDs in gene...
~~Feature #229~~: Updated immediate value in C...
~~Feature #226~~: Group configurable element va...
~~Feature #238~~: New option to SaveAs: hierarchy
~~Feature #239~~: Add module parameters to hie...

Figure 6.2 *Kactus2 Gantt chart for version 2.8 [38].*

7. KACTUS2 VERSION 2.8 VERIFICATION

Kactus2 version 2.8 was published on June 15th 2015. This is the latest release version at the time this thesis was written. In addition to the features covered in chapter 5, the changes in version 2.8 consist of upgrading the parameter importing from separate files, upgrading the parameter propagation mechanics, expanding the use of expressions to memory maps, incorporating hierarchical saving as an option, the addition of module parameters to the hierarchical views of a component and updating the visualization of memory maps.

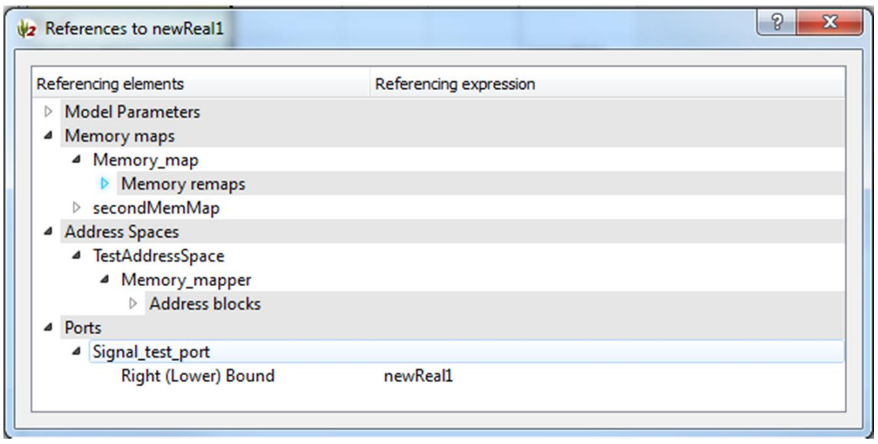
Testing of the developed version has been performed through manual testing as well as unit testing within Kactus2's own testing solution, using the Qt testing environment as its framework. Manual testing is performed through the debugging and testing of the implemented software. The unit tests are built according to guidelines given in the wiki Guides for Developers in Redmine [38]. This testing suite was incorporated in version 2.6 of Kactus2. Each of the new features is implemented in a testing solution representing its functionalities. The solutions can then be run separately or run together as a group.

The new and improved features incorporated in version 2.8 have been tested using both of the methods explained above. Despite extensive testing, some of the features contain minor malfunctioning or missing functions. The features are examined, re-evaluated and repaired during the transition to the next version of Kactus2.

7.1 Expressions in Kactus2

The system for expressions used in Kactus2 is mostly completed. The expressions work well for calculations and referencing. However, there are a few issues requiring further improved. These are mostly related to the editing of the contained expression and the usage calculation of the parameters involved in the expression.

The expression format used in Kactus2 possesses a minor inconvenience. When an expression is used to refer a parameter, the parameter gets a signal for increasing its usage count. When a reference to a parameter is removed from the expression, a signal is emitted to decrease the usage count. When a field is made possible to accept expressions, the signals must be connected to the parameters.



Referencing elements	Referencing expression
Model Parameters	
Memory maps	
Memory_map	
Memory remaps	
secondMemMap	
Address Spaces	
TestAddressSpace	
Memory_mapper	
Address blocks	
Ports	
Signal_test_port	
Right (Lower) Bound	newReal1

Figure 7.1 *The parameter reference tree.*

Parameters feature a column for displaying the amount of references made to them as well as a reference tree for displaying all the fields where the selected parameter has been referenced. This reference tree is shown in Figure 7.1. The left column of the tree contains the place of the referencing expression within the tree structure of the component editor. The right column displays the referencing expression itself. To fully incorporate the expression system to a new field, this tree must also be modified to include the new expression capable field or fields.

There are still fields in Kactus2 that, according to the IP-XACT standard can accommodate expressions. These include fields such as the base address of the master bus interface and the mirrored master bus interface, as well as the range and remap address of a mirrored slave bus interface. The expressions will be expanded to the fields that are able to contain them in the future versions of Kactus2.

7.2 Unresolved issues

With the problems in the current version of locked columns, the feature regarding the locking of a column is still in development. The two alternative solutions presented in this thesis solve some of the issues, but both of them present new problems of their own. These issues are explained in section 5.7.

While the view configuration tool is working as expected, currently only one of the generators makes use of it, the Quartus Project generator. The other generators currently contained within Kactus2 and the compatible plugin generators do not make use of the active views of the component instances. However, it is good to keep the view configuration tool in mind when developing and integrating new generators or other methods for Kactus2.

The memory map header generation for a system design is currently being re-designed. The idea is to expand the list of files accepted to the generated file, with the system memory map header generator gathering all the generated memory map header files.

The method of searching for the files is still open. In addition, there is an issue regarding the search of a design containing hierarchical component instances which contain themselves as sub-items. Currently the search is terminated if this is found to occur, and a warning is given for an incompatible design. Other issues concern the included memory map header files, and issues involving what to do if a component referenced by the component instance does not contain a local memory map header file, or if a component referenced by the component instance contains a global memory map header file in addition to the local memory map header file.

Furthermore, the memory map header generation contains issues related to the inclusion of opaque and transparent bridges within a component design. The bridge components are used to describe an interconnection between two interfaces within the same component. [3] [4]. The transparent bridge is currently supported within Kactus2, while the opaque bridges are considered as transparent bridges. The implementation for resolving the opaque bridges is still in development.

The visualization of the memory maps in the component editor of Kactus2 could be improved further. Currently, the visualization only displays either all the memory maps of the component, or a single memory remap within a memory map. Improvements to it would include displaying all the memory remaps along with the memory maps. This would help in the comparison in the placement of memory map items between memory maps and memory remaps. The problem regarding this multi memory map – visualization is centered on the space given for the memory map visualization. As seen in Figure 2.4, there is limited space available. If the visualization would receive more space, the memory remaps could be drawn next to their memory maps.

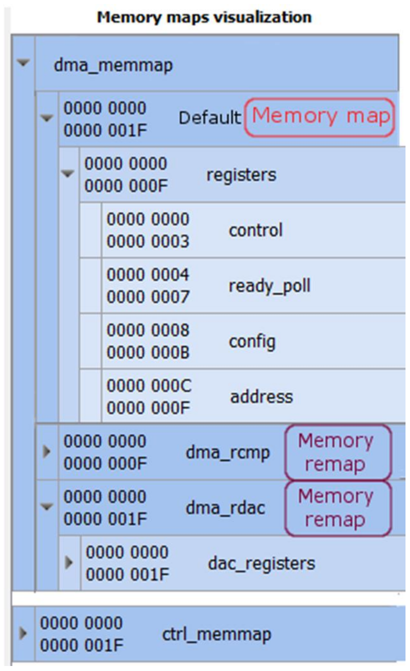

Memory maps visualization
dma_memmap
0000 0000 0000 001F Default
Memory map
0000 0000 0000 000F registers
0000 0000 0000 0003 control
0000 0004 0000 0007 ready_poll
0000 0008 0000 000B config
0000 000C 0000 000F address
0000 0000 0000 000F dma_rcmp
Memory remap
0000 0000 0000 001F dma_rdac
Memory remap
0000 0000 0000 001F dac_registers
0000 0000 0000 001F ctrl_memmap

Figure 7.2 *A suggestion for memory remap visualization.*

One option to visualize the memory maps together with the memory remaps would be to construct the visualization tree in the same manner as the component editor tree constructs the memory maps tree. This approach is shown in Figure 7.2. The memory remaps are constructed under the memory map, with the memory map itself contained within a default item. In the figure, the *dma_rcmp* and *dma_rdac* are memory remaps of the *dma_memmap* memory map. The address blocks belonging to each memory remap are placed under the containing memory remap and the registers are placed under their respective address blocks.

8. CONCLUSIONS

This thesis presented the development methods and features developed for Kactus2 version 2.8, and it has been considered a success. Compared to 2.7, 2.8 consists of approximately 20 000 more lines of code. The preliminary version of the Kactus2 expression editor was further improved and the expression parsing was incorporated to the generators. Independent industrial users have verified the correctness, completeness and usability of the new solutions.

The implementation of 2.8 did not face any serious setbacks, although some features had to be cut off. The system memory map header generation contains some unresolved issues related to the bridge and channel components.

One of the main intentions of Kactus2 is to provide the best user interface in its class. According to the industrial users, 2.8 greatly improved the general usability compared to the previous version.

The user interface of Kactus2 is evolving constantly. New features for improving the usability are devised when upgrading parts of the old systems or creating and incorporating new ones. This is to ensure that Kactus2 is the most user friendly open source tool for SoC and system design.

Although the listed features are developed specifically for Kactus2, it is possible to requse them in other IP-XACT based system design tools. Kactus2 is currently using the 1685-2009 IP-XACT standard with extension covering elements from the 1685-2014 standard. In the future, Kactus2 will be changed to fully support the latest IP-XACT version.

One experimental approach is Kactus3D. Kactus3D offers a three dimensional approach to the visualization of the SoC designs. A 3D view helps understanding large, hierarchical designs, instead of the current design, in which each level is in its own own 2D view.

REFERENCES

[1] R.A. Bergamaschi, J. Cohn. The A to Z of SoCs. IEE/ACM International Conference on Computer Aided Design. 10–14 Nov. 2002. pp. 791 – 798

[2] A. Kamppi, J-M. Määttä, L. Matilainen, E. Salminen, T.D. Hämäläinen. Kactus2: Extended IP-XACT metadata based embedded system design environment. 2012 1st International Workshop on Metamodelling and Code Generation for Embedded Systems, MeCoEs. 6p

[3] IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating and Reusing IP within Tool Flows, IEEE Std 1685-2009. The Institute of Electrical and Electronics Engineers, Inc. 18 Feb. 2010. 373 p.

[4] IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating and Reusing IP within Tool Flows, IEEE P 1685/D5, The Institute of Electrical and Electronics Engineers, Inc., 12 Sep. 2014, 510 p.

[5] E. Salminen, T.D. Hämäläinen, M. Hännikäinen. Applying IP-XACT in product data management. 2011 International Symposium on System on Chip (SoC). 31 Oct 2011 – 2 Nov 2011 pp. 86–91.

[6] Kactus 2 home page. [WWW] [Referenced on 16 Mar. 2015] Available at: http://funbase.cs.tut.fi.

[7] A. Kamppi, L. Matilainen, J-M. Määttä, E. Salminen, T.D. Hämäläinen, M. Hannikainen. Kactus2: Environment for Embedded Product Development Using IP-XACT and MCAPI. 14th Euromicro Conference on Digital System Design (DSD). 31 Aug 2011 – 2 Sept 2011. pp. 262 – 265

[8] A. Kamppi. Library Management Implementation on Kactus2 IP-XACT tool. Tampere University of Technology. Dec. 2012. 112 p. Available at: http://dspace.cc.tut.fi/dpub/handle/123456789/21289

[9] Mentor Graphics Corporation. HDL Designer. WilsonVille, OR, USA. [WWW] [Referenced on 24 June 2015] Available at: http://www.mentor.com/products/fpga/hdl_design/hdl_designer_series

[10] Mentor Graphics Corporation. Lockheed Martin Space Systems Company Success. WilsonVille, OR, USA. [WWW] [Referenced on 20 Oct 2015] Available at: https://www.mentor.com/products/fv/success/lockheed_martin_fpga_success

[11] M-Files. [WWW] [Referenced on 25 June 2015]. Available at: https://www.m-files.com/

[12] CSIR. Scineric Workspace. [WWW] [Referenced on 25 June 2015]. Available at: http://scineric.csir.co.za/

[13] S. Abdi, Y. Hwang, L. Yu, H. Cho, I. Viskic, D. Gajski. Embedded System Environment: A framework for TLM-based design and prototyping. Rapid System Prototyping (RSP). 21st IEEE International Symposium on Rapid System Prototyping. 8–11 June 2010. pp. 1–7

[14] Magillem. [WWW] [Referenced on 25 June 2015]. Available at: http://www.magillem.com

[15] ARM. Socrates Design Environment. [WWW] [25 June 2015]. Available at: http://www.arm.com/products/system-ip/ip-tooling/socrates.php

[16] Xilinx. Vivado Design Suite. [WWW] [Referenced on 25 June 2015] Available at: http://www.xilinx.com/products/design-tools/vivado.html

[17] Nivalis Solutions. IP Packager and Design assembly. [WWW] [Referenced on 25 June 2015] Available at: http://nivalissolutions.com/

[18] EDAUtils. IP-XACT solution. [WWW] [Referenced on 25 June 2015]. Available at: http://www.edautils.com/ip-xact.html

[19] H. Nikolov, M. Thompson, T. Stefanov, A. Pimentel, S. Polstra, R. Bose, C. Zissulescu and E. Deprettere. Daedalus: Toward composable multimedia mp-SoC design. Design Automation Conference 2008, DAC 2008, 45th ACM/IEEE. 8–13 June 2008. pp 574–579

[20] F. Balarin, Y. Watanabe, H. Hsieh, L. Lavagno, C. Passerone and A. Sangiovanni-Vincentelli. Metropolis: an integrated electronic system design environment. Computer, vol. 36, Issue 4. April 2003. pp. 45–52

[21] C. Haubelt, T. Schlichter, J. Keinert and M. Meredith. SystemCoDesigner: Automatic design space exploration and rapid prototyping from behavioral models. Design Automation Conference, 2008, DAC2008, 45th ACM/IEEE. 8–13 June 2008. pp 580–585

[22] J. Eker, J. Janneck, E. Lee, J. Liu, X. Liu, J. Ludwig, S. Neuendorffer, S. Sachs and Y. Xiong. Taming Heterogeneity – The Ptolemy approach. Proceedings of the IEEE, vol. 91, Issue 1. Jan 2003. pp. 127–144

[23] Synopsys. Corebuilder. [WWW] [Referenced on 25 June 2015] Available at: http://www.synopsys.com/dw/ipdir.php?ds=core_builder

[24] Microsoft. Excel. [WWW] [Referenced on 31 July 2015] Available at: https://products.office.com/en-us/excel

[25] J. Nielsen. 10 Usability Heuristics for User Interface Design [WWW] [Referenced on 8 June 2015]. 1 Jan 1995. Available at: http://www.nngroup.com/articles/ten-usability-heuristics/

[26] Kactus2: The Library and VLNV. Tampere University of Technology. 14 Nov. 2013. Available at: http://funbase.cs.tut.fi/doc/Kactus2_vlnv.pdf

[27] Mentor Graphics Corporation. ModelSim. WilsonVille, OR, USA. [WWW] [Referenced on 15 June 2015] Available at: http://www.mentor.com/products/fv/modelsim/

[28] IEEE Standard VHDL Language Reference Manual, IEEE Std 1076-2008 (Revision of IEEE Std 1076 – 2002), The Institute of Electrical and electronics Engineers, Inc. 2009. 626 p.

[29] Multicore Communications API working group [WWW] [15 June 2015] Available at: http://www.multicore-association.org/workgroup/mcapi.php

[30] Mentor Graphics Corporation. PADS. WilsonVille, OR, USA. [WWW] [Referenced on 15 June 2015] Available at: http://www.pads.com

[31] Microsoft. Microsoft Visual Studio 2010 [WWW] [Referenced on 16 June 2015]. Available at: https://msdn.microsoft.com/en-us/library/dd831853%28v=vs.100%29.aspx

[32] Whole Tomato software. Visual Assist [WWW] [Referenced on 16 June 2015] Available: http://www.wholetomato.com/

[33] Qt. [WWW] [Referenced on 1 Apr. 2015] Available at: http://www.qt.io/

[34] Business Wire. Handheld 3d imaging system for medical applications relies on Qt Commercial. Virtual-Strategy Magazine, 8 May 2012. [WWW] [Referenced on 30 June 2015] Available at: http://www.virtual-strategy.com/2012/05/08/handheld-3d-imaging-system-medical-applications-relies-qt-commercial#axzz3eXfwTFP9

[35] M. Fowler. GUI Architectures. 18 July 2006. [WWW] [Referenced on 14 Aug. 2015] Available at: http://martinfowler.com/eaaDev/uiArchs.html

[36] Redmine. [WWW] [Referenced on 17 Mar. 2015] Available at: http://www.redmine.org

[37] Source Forge. [WWW] [Referenced on 18 Mar. 2015] Available at: http://sourceforge.net/

[38] Kactus2 Project Management. [WWW] [Referenced on 29 June 2015] Available at: http://kactus2.cs.tut.fi/

[39] M. Teuho, E. Pekkarinen, T. Hämäläinen. Semantic analyzing expression editors in IP-XACT design tool Kactus2. 14th Symposium on Programming Languages and Software Tools (SPLST). 9 – 10 Oct 2015, Tampere, Finland.