Semantics analyzing expression editors in IP-XACT design tool Kactus2

Mikko Teuho, Esko Pekkarinen, Timo D. Hämäläinen

Tampere University of Technology, Tampere, Finland
mikko.teuho@tut.fi, esko.pekkarinen@tut.fi,
timo.d.hamalainen@tut.fi

Abstract. This paper presents parameter and expression editors of the design tool Kactus2. It is aimed at digital System-on-Chip (SoC) designs based on IEEE 1685 IP-XACT XML metadata standard. SoC's are constructed by assembling parametrized components using generators for hardware language code and design configuration. The key challenges are the management of dependencies between thousands of parameters, as well as immediate validation and evaluation while editing. The expression editor in this paper has been designed to overcome these challenges. The editors include real-time syntax, semantic analysis and the use of UUIDs behind user displayed parameter names. The implementations for these have been published in Kactus2 v2.8 open source code, written in C++/Qt5, and consisting of 3000 LoC in the release. An independent industrial user on the SoC domain has verified the correctness, completeness and usability of the new solutions. The designed editors significantly improve the SoC parameter editing and design configuration.

Keywords: IP-XACT · Kactus2 · Electronic Design Automation · System-on-Chip · Expression analysis · Semantic analysis

1 Introduction

This paper presents new features and implementation solutions for a design tool called Kactus2. It is aimed at digital systems design, especially System-on-Chip (SoC) designs which can include hundreds of Intellectual Property (IP) blocks possibly representing millions of lines of hardware description language (HDL) code [1]. Kactus2 uses IEEE standards 1685-2009 and 1685-2014 "IP-XACT" XML metadata that is aimed at unified specification for electronic design automation, IP vendors, and system design communities [2]. IP-XACT specifies how IP blocks are packaged and assembled independent of the design tool, implementation language or vendor used in creating the IP block [3]. IP-XACT 2014 templates include 24 XSD files, totaling of 790 elements and 241 attributes. In addition, so called vendor extensions can multiply these numbers, which shows that IP-XACT is a complex specification. It has been widely adopted in VLSI industry recently accompanied by FPGA vendors.

SoC designs are increasingly based on automatic generation of HDL code, HW dependent SW device drivers and automated building of the SW stack. IP-XACT at-

tempts to specify a standard for all of this by a single point of control. The main HW related tasks are parametrization of the IP blocks, instantiation and connection of the blocks and design configuration.

Parameters are heavily used e.g. to construct structures and memory regions accessible to SW. Parameters can be propagated through hierarchy, but global references are not allowed to guarantee reuse of each IP block. Parameters defined in an IP-block can be overridden for each instance in a design. There are six parameter attributes like strict typing or allowable numerical range for the different purposes. These can be used to explicitly define the values that a parameter can be given.

The challenge is how to manage even thousands of parameters and attributes that may depend on each other and form long chains in expressions. Since the user cannot comprehend the whole system at once, a tool should validate, and when possible, evaluate parameter values on-the-fly while editing. A mistake in referencing, renaming or copy-pasting can be very difficult to find afterwards.

Another challenge is how to enter the parameter information in a tool. Some related tools resemble XML code editors, and some offer very large table sheets. Both are prone to human errors and reduce the usability. An XML code should not be visible for a system designer.

The need for an open source tool easing the reuse, configuration and assembly of SoC designs motivated the development of Kactus2 at Tampere University of Technology. Even stronger motivation was to create an outstanding user interface hiding the complexity of IP-XACT, since the tools in this domain are commonly known to be very difficult to use.

Kactus2 was launched in 2010. It is implemented in C++ and Qt5, and the current release v2.8 has ~390k LoC. Kactus2 is actively being used in several companies, which also have supported the development. There are over 7800 installer downloads by Aug 2015, which is a very good number in this specific domain. Direct feedback from the users confirms the good usability.

This paper presents new solutions developed for Kactus2 for SoC design configuration, which were implemented in v2.8. The new contributions are following:

- A new approach to expression editors.
- Real-time validation and semantic analysis of the parameters involved in defining the high-level SoC design configuration.

This paper is organized in the following way. Chapter 2 discusses the related work of expression editors. Chapter 3 introduces the solutions for the expression editor of Kactus2. Chapter 4 contains the evaluation of the created mechanics. Chapter 5 concludes this paper.

2 Related work

The early IP-XACT tools were Eclipse-based XML editors, which are still available for many variations as commercial tools. The closest free tool to Kactus2 is EDAUtils IP-XACT Solution [4], but it has much less features and is basically an IP-XACT

XML editor. All true competitors are commercial tools, but they deny any comparisons and publication of the user interface in their licensing terms. In addition, commercial tools have often been customized, which complicate comparison. Thus, Kactus2 is currently the only usable open source IP-XACT tool.

For the related work, we consider more common solutions outside IP-XACT domain. The most important part is an expression editor that should

1. Have a WYSIWYG editor,
2. Support basic mathematical operators,
3. Support nested sections in parentheses,
4. Have constants, parameters and operators adding, copying, pasting and moving within and between expressions,
5. Offer only existing parameters to be inserted and rejecting any other strings,
6. Validate the formula when typing,
7. Evaluate the value of the expression immediately,
8. Hide the parameter IDs and display only parameter names to the user.

The parameter references are based on unique IDs (UUID), which means the parameter name can be changed without breaking the dependencies. In addition, when two IP blocks are integrated with the same parameter name but different meaning, UUIDs help resolving the issue.

Marques et al. have presented WIRIS OM Toolset [5] for managing mathematical expressions. This application tries to construct outputs of mathematical fragments written in a suitable markup language. WIRIS OM Toolset has been incorporated into the LeActiveMath and WebALT project [5]. LeaActiveMath is a learning tool system designed for high school, college or university level teaching [6]. WebALT is an application that uses existing standard to represent mathematical equations on the web together with existing linguistic technologies in order to create a language-independent mathematical learning platform [7]. The TextMathEditor of WebALT allows the construction of mathematical expressions using a predetermined grammar. The WIRIS Formula Editor of the WIRIS OM Toolset has been incorporated within this math editor.

Lee et al. have developed MathCast [8], an open source equation editor for mathematical expressions. The produced equations can be used in documents, graphically rendered picture files or within MathML Presentation 2.0, format for describing mathematics [9]. A tool is included within MathCast to author XHTML web pages with mathematical equations.

Design Science Inc. offers MathType, a commercial equation editor tool for Windows and Macintosh [10]. It contains a wide range of mathematical functions and customization options, with an ability to export these equations to Mathematical Markup Language (MathML). MathType also understands the TeX and LaTeX typesetting. MathType is compatible with a multitude of applications & websites, such as Adobe InDesign and Microsoft Office tools.

Design Science Inc. offers another commercial mathematical equation tool, called MathFlow [11]. MathFlow is a MathML Toolbox standard, offering tools for editing, displaying and accessing mathematical notations on websites, applications and ser-

vices. It can be incorporated into a suite of other software, such as Oxygen [12]. The MathFlow toolbox consist of editors, equation composers and document composers.

All related editors have features from 1 to 7, e.g. WYSIWYG, use of previously constructed constants, parameters and operators and nesting. MathType also supports handwriting but this is not important for SoC design. However, none of the editors support our requirement 8. We also need to consider how operands and operations are defined for the expression editor.

2.1 Related standards

There are two standards for capturing mathematical equations: OpenMath [13] and MathML [9]. There is a large overlap between these two communities. Both attempt to create an extensible standard for representing the semantics of mathematical objects. As a web based system, MathML is based on XML in order to help browsers natively render mathematical expressions. OpenMath endorses XML and binary formats, although a custom encoding can be used. It is primarily targeted towards the semantic meaning and content of mathematical objects instead of focusing on representing the objects. This approach is used in MathML. However, OpenMath has received critique for its use of general mathematics [14].

Since both the MathML and OpenMath are based on XML [9] [13], it could be incorporated within Kactus2. Although these could help in displaying the constructed equations within the expression editor, the semantic analysis is the important part of the IP-XACT standard. Both of these standards contain semantic tags for constructing the expressions. Following the structure, an analysis can be formed. For example the following example sentence

$$x^2 + 4x + 4 = 0 \quad (1)$$

could be constructed using the MathML semantics[9] as follows:

```
1.    <apply>
2.        <plus/>
3.        <apply>
4.            <power/>
5.            <ci>x</ci>
6.            <cn>2</cn>
7.        </apply>
8.        <apply>
9.            <times/>
10.           <cn>4</cn>
11.           <ci>x</ci>
12.       </apply>
13.      <cn>4</cn>
14.   </apply>
```

Since these standards describe the expressions in XML format, parsing one sentence for editing requires resources. Comparing to the expression editor developed for Kactus2, an expression is contained within a single string-variable. The managing of

the equation becomes resource consuming as the XML tags of the expression need to be re-applied every time the user wants to replace a value. Displaying the data of a table containing tens of parameters each containing between one to three expressions could possibly cause long loading times in opening and changing tables.

Regardless of the merits of existing solutions, none of these are easy to integrate into Kactus2. The equation editors presented here are applications, and removing the functionality of expressions from them could prove difficult. Kactus2 operates using IP-XACT standard defining the parameters that are used in its expressions. These expressions are structured according to System Verilog. To ensure that the parameter references can be handled properly, together with a reliable and immediate expression validation and evaluation, a custom expression editor was implemented for Kactus2. The editor would be difficult to implement in other, non-IP-XACT applications.

3 Equations in Kactus2

Expressions are used in Kactus2 to capture mathematical equations and parameter references. The references are constructed using the unique identifier of the referenced value.

3.1 Expressions in Kactus2

In IP-XACT, the expressions are stored in an XML file. The XML element depends on the correct location of the expression. Following is a fragment of IP-XACT file of a *component* containing parameters. The ** is used as an operator for exponentiation:

```
<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>
```

This example displays three parameters, with parameter *clk_enable* containing a similar equation as in equation (1). In this example, the string uuid followed by a list of characters is a universally unique identifier (UUID) [15] of a parameter value. Every parameter value within Kactus2 has this identifier. When an XML file is read, Kactus2 checks if a parameter contains a unique identifier. If this identifier is not found, a new UUID is created.

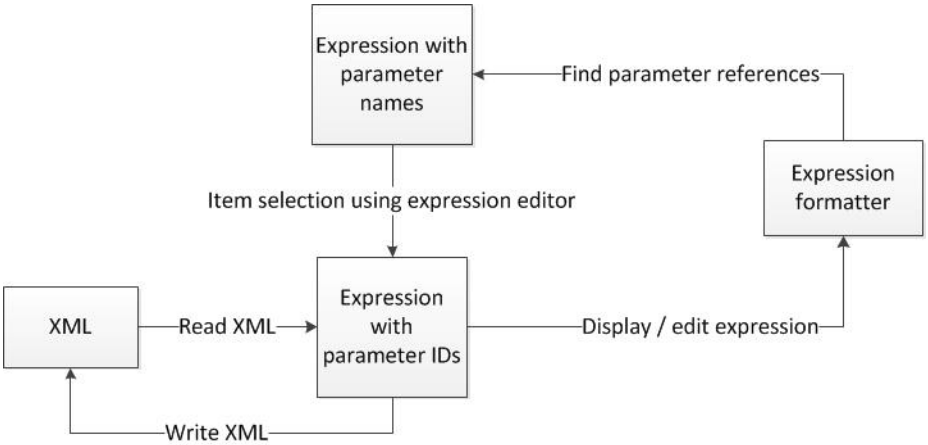

Fig. 1. Expression reading and displaying in Kactus2

The flowchart for handling of XML files in Kactus2 is shown in **Fig. 1**. As shown above, the XML file contains the references of an expression as unique identifiers to the referenced parameter value. When an expression is displayed in a table of Kactus2, the expression formatter of the table changes the unique identifier to a more human readable name of the parameter. Only the name is displayed for user editing the expressions.

3.2 The expression editor

A section of the ***Parameters editor*** of Kactus2 is shown in **Fig. 2**. This editor is used to create and manage the parameters of a component. The name column displays the parameter name, while the type column describes the type of the value. If the selected type cannot contain the given value, the type is displayed in red.

Name	Description	Data type	Type	Value, *f*(x)	Choice
port_width			int	-2	
port_range			int	14	
clk_enable			real	port_width**2+4*port_width+4	

Fig. 2. Parameters editor of Kactus2

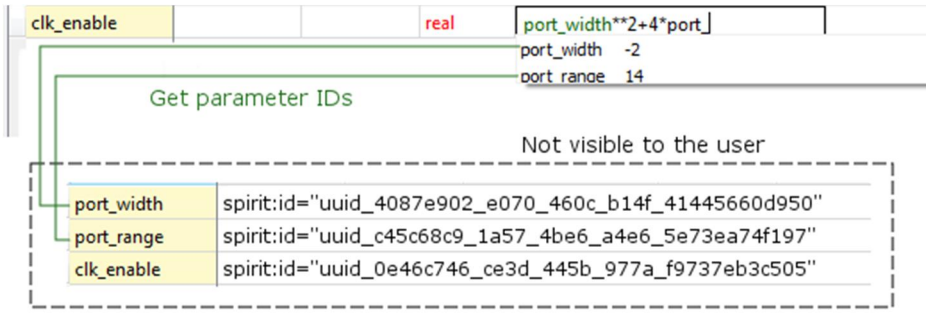

Fig. 3. Reference selection in the expression editor.

The symbol f(x) in the header informs that the value can be given as an expression. The value of the expression is displayed as a tooltip of the containing field. The expression itself is visible when viewing or editing a parameter. The parameter values can also be given as arrays. Every value of an array can be given a different expression, but multidimensional arrays are not supported in Kactus2.

While constructing an expression, the Kactus2 expression editor offers a list of possible completions for an incomplete word. This can be observed in **Fig. 3**, where the value of parameter *clk_enable* is being edited. The available selections are determined by the location of the expression editor. In the **component editor**, these completions are selected from the parameters of the currently active component. In the **configurable element editor**, the completions are selected from the configurable element values.

Each selection consists of the name of the corresponding parameter and its value, while the unique identifier of the parameter is hidden from the user. When a completion is selected, the expression editor retrieves this unique identifier and inserts it to the edited value. Thus the expression itself is constructed using the unique identifiers of the referenced parameters. When the finished expression is displayed to the user, the **expression formatter** changes the unique identifier of the referenced parameter to the name of the parameter.

The expression editor will colour green the valid parameter references of an expression. Values containing invalid references are coloured red. All the references are made using the unique IDs of parameters, but are displayed using the parameter names. This is because the parameter IDs are not human readable.

The selection management is handled by the **parameter completer** class. **Fig. 4** displays the structure of this completer. This class creates the available selections for the user when selecting a completion to an unfinished word, as seen in **Fig. 3**.

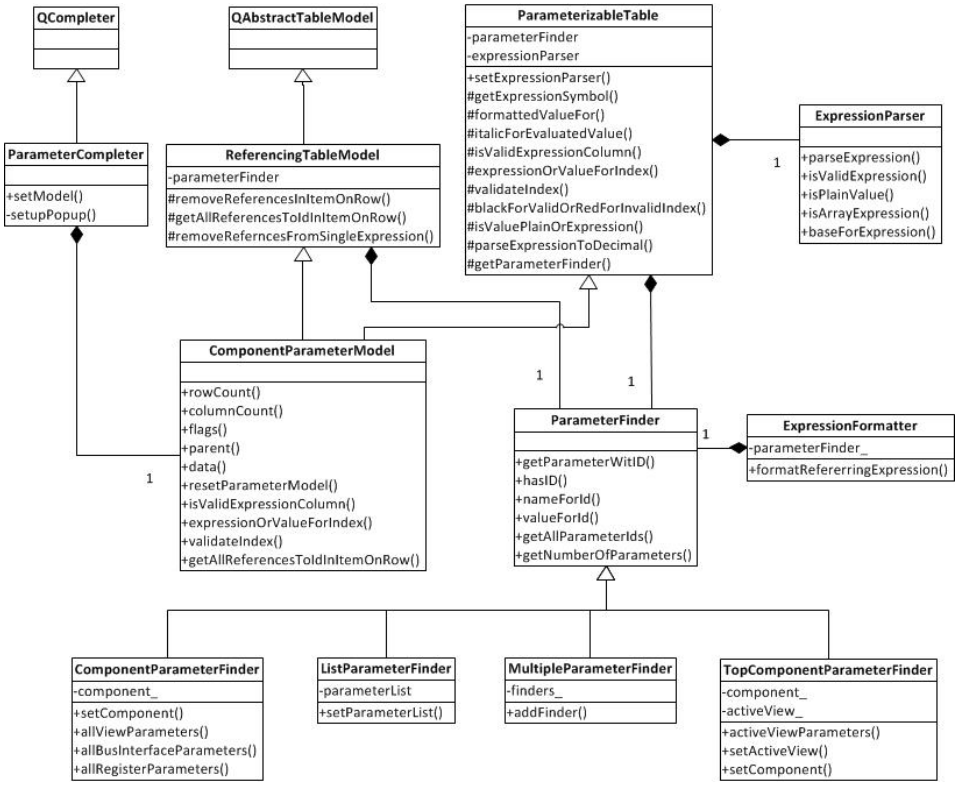

Fig. 4. The structure of **parameter completer** class in the expression editor.

The parameter completer is derived from the QCompleter class of Qt, and uses a component parameter model to construct the model seen in the selection. This model retrieves the parameters according to its currently selected **parameter finder** class. It is based on the table editors of Kactus2 and thus is derived from both the **referencing table model** class and the **parameterizable table** class. These handle the basic functionalities of a table view in Kactus2. An **expression parser** class is attached to the **parameterizable table** class. This parser evaluates the values of expressions. The **parameter finder** class is used to find referable parameters, with its subclasses determining the place where these parameters are found.

In version 2.8 of Kactus2, the classes contributing to the expression editor consist of approximately 3000 lines of C++/Qt code. This includes the expression managing and parsing, reference selection and expression formatting.

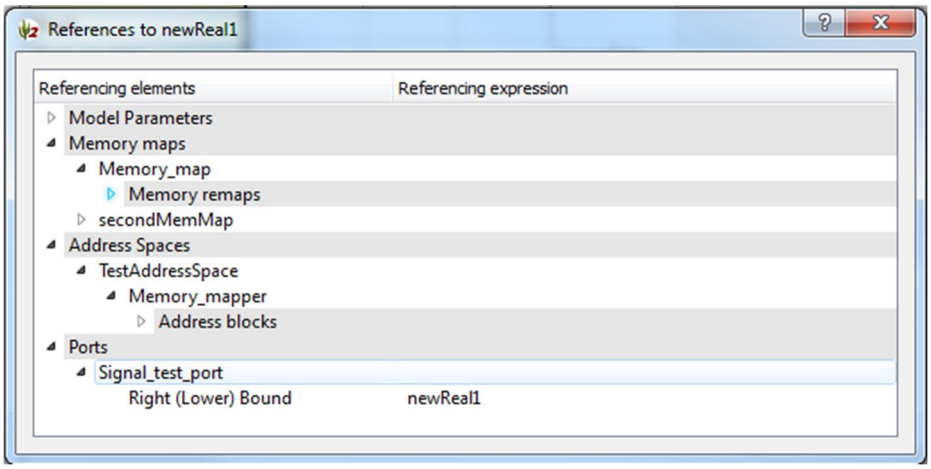

Fig. 5. View of the parameter reference tree.

3.3 Parameter reference tree

IP-XACT based designs can include very complex parameter reference chains. To help managing this, Kactus2 offers a parameter reference tree view with the expression editor. The tree visualizes all of the locations where the parameter has been referenced within the component together with the expression, in which the parameter has been referenced. This reference calculation helps the user to be informed of the importance of a parameter. A warning is given if a user tries to remove a parameter that has been referenced at least once. An example of the reference tree is depicted in **Fig. 5**.

The parameter reference tree is constructed on demand if a parameter has been referenced in an expression. This is determined by the usage amount column of a parameter. Keeping this usage count constantly in a correct state requires the tracking of all the expression editors of Kactus2.

Whenever a reference is selected in the expression editor, a signal is sent to the **parameter reference calculator** class. This signal informs the reference calculator class to increase the usage count of the referenced parameter. When a reference is deleted or changed, a signal is sent to inform the reference calculator to decrease the amount of references made to the parameter.

3.4 Editing an expression

When an expression is edited, the expression editor tracks the current location of the editing. When the user removes or inserts characters into the expression, this location is used to determine what is in the current position. If the position contains a reference to a parameter value, a signal is emitted to inform that the reference has been removed and the usage count of that parameter should be decreased, thus affecting the parameter reference tree.

When the user changes location in the expression, the **parameter completer** attached to the expression editor displays a list of possible references to complete the word at the current location.

If a new reference is not selected for the edited word, the XML file containing the edited parameter value is changed. The uuid of the referenced value is replaced with the edited name of the parameter. For example this XML contains parameter clk_counter. Its value is an expression containing a reference to parameter *port_range*:

```
<spirit:parameter type="int">
  <spirit:name>clk_counter</spirit:name>
  <spirit:value spirit:id=
  "uuid_5ecaeb33_fd7f_4512_acbf_5684d504af02">
    uuid_c45c68c9_1a57_4be6_a4e6_5e73ea74f197+4
  </spirit:value>
</spirit:parameter>
```

The uuid of parameter port_range is displayed in the value element of the parameter *clk_counter*. When the value of the *clk_counter* is edited by removing a character, the uuid is replaced by the edited name of parameter port_range:

```
<spirit:parameter type="int">
<spirit:name>clk_counter</spirit:name>
  <spirit:value spirit:id=
  "uuid_5ecaeb33_fd7f_4512_acbf_5684d504af02">
    port_rage+4
  </spirit:value>
</spirit:parameter>
```

The analysis of a modified expression is performed after the editing is finished. The new expression is evaluated and the results are displayed in the tooltip of the finished expression.

3.5 Semantic analysis of an expression

Kactus2 follows the System Verilog syntax for evaluating expressions. The validity of an expression is handled in the expression parser classes. Invalid expressions are colored red and the results are given as a string containing the text N/A.

Regular expressions are used to determine the validity of an expression. These define the available operators and functions that can be given in the expressions of Kactus2. A valid expression within the expression editor of Kactus2 contains at least one operator or a string literal. These operators can be negative or positive. Expressions can contain more operators, but they must be connected with mathematical operands. String literals are accepted as valid expressions.

In addition to the regular expression containing the form of the equation, the parentheses and braces within the expression are examined. Parentheses are used to con-

struct equations with priority calculations. Valid places for open parentheses are before the first operand or before any subsequent operands. The closing parentheses are placed after any subsequent operands. To handle the construction of multiple parentheses containing functions, the expression parses checks if the expression contains the same number of opening and closing parentheses.

Braces identify a value containing an array of values. A valid array contains an equal number of opening and closing braces. A value within an array can be constructed similarly as a basic value, i.e. it can contain expressions. Multidimensional arrays are not supported in the current version of Kactus2.

The expression editor is compatible with the decimal, hexadecimal, octal and binary number formats. The basic functionalities of the *ExpressionParser* class of Kactus2 parses any given expression to a decimal number. The *ValueFormatter* class allows formatting any expression to a desired number format.

3.6 Expression evaluation

The evaluation of the expressions in Kactus2 is performed by the *ExpressionParser* class and its subclasses. The flowchart for expression analysis is shown in **Fig. 6**. The expression parser starts by checking any given expression for its validity. An N/A is returned if an expression is found to be not applicable in Kactus2. It then checks if the given expression is an array of expressions. If it is an array, the expression parser evaluates all the expressions contained within it.

The equation is divided into a list of string type units. Each unit contains either an operator, or a word delimiter. Using this list of units, the expression parser calculates the value for the given expression using the standard arithmetic formulas.

Parentheses used in the expression are solved in a similar manner to the mathematical functions. The beginning and ending parentheses are first matched. Then each of the expressions contained within the parentheses is then calculated separately, beginning with the innermost expression.

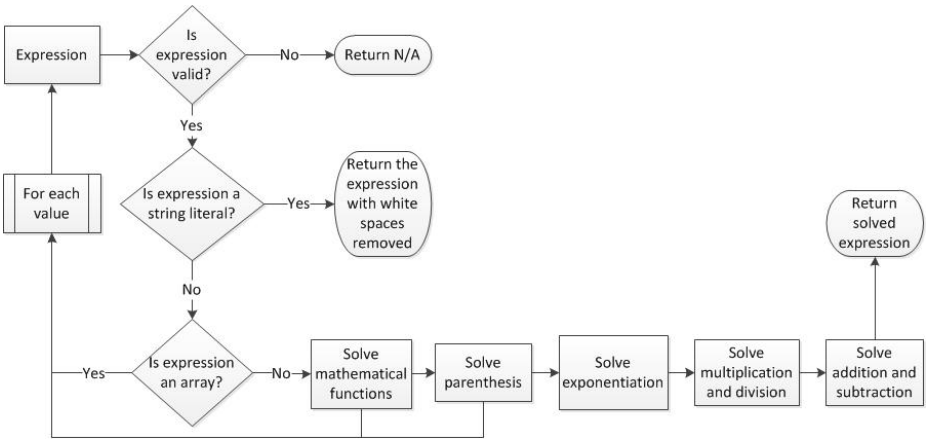

Fig. 6. The flowchart for expression analysis used in the expression editor of Kactus2.

4 Evaluation of the equation mechanics

The semantic analysis of Kactus2 handles the basic functionalities of a mathematical equation editor. Comparing to other software designed to create and manage complex mathematical structures, the expression editor of Kactus2 may seem simple. Kactus2 does not support the drawing of mathematical functions, or some the basic functionalities such as calculating square roots. However, the expression editor is currently purposed to handle the mathematical needs of IP-XACT based SoC designs.

The speed of the expression editor depends on the amount of parameters currently contained within a component. A test was conducted on a Win7 operating system and 4 processors (x86-64, WOW64) with a component containing approximately a hundred parameters. Writing an expression causes no lag. Less than a second of lag can be perceived in constructing the list of selectable references. Editing an expression does not incur any more lag than the construction of the reference selections when editing sections of the expression. The expression editor presented in this paper follows the System Verilog structure to validate and analyze any expressions given to it. This structure contains the clog2 (ceiling of log2) function.

The standards OpenMath and MathML are not necessary to be implemented in Kactus2. The expressions in Kactus2 are stored in simple string variables compared to the XML tags used in both of these standards. Accessing the correct XML tag to change a single operator of an expression can be considered more resource consuming, compared to the handling of expressions contained a string. Additionally, as Kactus2 is developed for SoC design, the expression editor contained within it does not need very complex mathematical functions.

5 Conclusions

The Kactus2 expression editor presented in this paper allows the creation of mathematical expressions displaying parameter names to the user while hiding the references made to the parameter IDs. These parameters can be located within different parts of a component. The different parameter finders of Kactus2 are used to specify the location of the usable parameter references. This can be either the currently active IP-XACT component, the component referenced by the currently active IP-XACT component instance, the configurable elements of an IP-XACT component instance or the top component containing a design housing the IP-XACT component instance.

The expression editor shown in this paper allows the defining of mathematical equations. The editor can support the basic arithmetic formulas, as well as understand different formats for the input of the data. These are the decimal, hexadecimal, octal and binary formats.

Future work includes extension of the current semantic analysis for IP-XACT vendor extensions, which have specified nodes in the XML tree but custom content. This would be implemented by an extensible metamodel for the items of the expressions.

The benefits of the expression editor of Kactus2 are improved usability and reduced number of errors in SoC designs. Using the UUIDs as references allows relia-

ble equation construction with the desired parameters. The use of parameter names in place of the referenced UUIDs in the expression editor help in transforming these equations into more human readable.

Without the expression formula and references validation, the IP-XACT XML files could contain serious errors that can be difficult to find and verify in a large SoC design. As the expression validation is built into the expression editor, the IP-XACT based SoC designs cannot contain parameter reference errors in Kactus2. Thus the presented expression editor improves the SoC design process productivity and quality significantly compared to plain IP-XACT XML editors that are still widely used.

6 References

1. 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
2. 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.
3. 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.
4. EDAUtils. IP-XACT solution. [WWW] [Referenced on 14 Aug. 2015]. Available at: http://www.edautils.com/ip-xact.html
5. D. Marquès, R. Eixarch, G. Casanellas, B. Martínez. WIRIS OM Tools: a Semantic Formula Editor. Mathematical User-Interfaces Workshop 2006. 10 Aug. 2006. 8p.
6. Language-Enhanced, User-Adaptive, Interactive e-Learning for Mathematics. [WWW] [Referenced on 7 Aug. 2015] Available at: http://www.leactivemath.org
7. WebALT: Web Advanced Learning Techniques. [WWW] [Referenced on 7 Aug. 2015] Available at: http://www.webalt.net/index_eng.html
8. T. Lee, T. Chekam. MathCast: The open source equation editor. [WWW] [Referenced on 7 Aug. 2015] Available at: http://mathcast.sourceforge.net/home.html
9. W3C. MathML. [WWW] [Referenced on 7 Aug. 2015]. Available at: http://www.w3.org/Math/
10. Design Science. MathType 6.9. [WWW] [Referenced on 7 Aug. 2015]. Available at: http://www.dessci.com/en/products/mathtype/
11. Design Science. MathFlow. [WWW] [Referenced on 7 Aug. 2015]. Available at: http://www.dessci.com/en/products/mathflow/default.htm
12. Oxygen XML editor. [WWW] [Referenced on 19 Aug. 2015] Available at: http://www.oxygenxml.com/
13. Open Math Society. [WWW] [Referenced on 7 Aug. 2015]. Available at: http://www.openmath.org/society/index.html
14. R. Fateman. A Critique of OpenMath and Thoughts on Encoding Mathematics. University of California, Berkeley, Computer Science Division. 17 Jan. 2001. 10p
15. P. Leach, M. Mealling, R. Salz. A Univerally Unique Identifier (UUID) URN Namespace. Network Working Group. July 2005. Available at: http://www.ietf.org/rfc/rfc4122.txt