IP-XACT REGISTER MAP GENERATOR

by

Onur Balcı

Engineering Project Report

Yeditepe University

Faculty of Engineering and Architecture

Department of Computer Engineering

2014

IP-XACT REGISTER MAP GENERATOR

APPROVED BY:

Assoc. Prof. Dr. Sezer Gören Uğurdağ ……
(Supervisor)

Asst. Prof. Dr. Dionysis Goularas ……

Dr. Mustafa Mutluoğlu ……

DATE OF APPROVAL: 27/05/2014

Dedicated to X…

ACKNOWLEDGEMENTS

First of all I would like to thank my advisor Assoc. Prof. Dr. Sezer Gören Uğurdağ for her guidance and support throughout my project. Also I would like to thank to Dialog Semiconductor for supporting this project, and my work.

Also I would like to thank to my friends, especially Uğur Kınık and Oktay Elipek for their help. Finally I would like to thank my family for their support throughout my project.

ABSTRACT

IP-XACT REGISTER MAP GENERATOR

With development of integrated circuit technology, importance of electronic design automation is increasing. Dozens of tools and standard exist for making automation process efficient. However, these standards and tools are not compatible with each other hundred percentage. That is why the industry made IP-XACT standard. IP-XACT perfectly work with new designs, but it does not provide compatibility with old standards. Old designs must be converted to new IP-XACT standard. This project aims to design a tool which convert old register designs to IP-XACT format for purpose of speeding this transition.

ÖZET

IP-XACT REGISTER MAP GENERATOR

Entegre devre teknolojinin gelişmesiyle birlikle, elektronik devre otomasyonun önemi artıyor. Otomasyon sürecini daha verimli hale getirmek için düzinelerce araç ve standart var. Bu standartlar ve araçlar birbiriyle tam uyumlu değil, o yüzden firmalar IP-XACT standardını oluşturdular. IP-XACT yeni çıkan tasarımlarda mükemmel çalışıyor, fakat eskiye yönelik bir desteği bulunmuyor. Eski tasarımların, yeni standarda dönüştürülmesi gerekiyor. Bu proje geçiş sürecini hızlandırmak adına, eski register tasarımlarını IP-XACT'e dönüştüren bir araç geliştirmeyi amaçlıyor.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS / ABBREVIATIONS

IC	Integrated Circuit
IP	Intellectual Property
EDA	Electronic Design Automation
IEEE	The Institute of Electrical and Electronics Engineers
ASCII	American Standard Code for Information Interchange
XML	Extensible Markup Language
XS	XML Schema
XSD	XML Schema Definition
CSV	Comma-Separated Values
PHP	PHP: Hypertext Preprocessor
JS	JavaScript

1. INTRODUCTION

Electronic devices are everywhere in our life and they can do more every day. They can handle so many tasks, because of their hardware, more specifically their chip is capable do more. Of course these capability is about agreements between companies. They use each other components inside their chip

Of course developing a chip is not that much easy Even though, companies design chip with tools and manufacture it with automation systems, the chips are getting smaller and complex. Most of problems are happened when integrating a third party IP core, which means an IC component design ready to use. The companies or who design IP core use own standard. Generally this standard work with couple of expensive tools.

The industry realized that problems and developed a new standard which is called IP-XACT for storing IP core information. Although, IP-XACT solves most of industry's problem, IP cores which have old document format standards, must be converted for later use. This study aims to make tool that can be used for this transition.

Report sections organized as follows. Section II explains the development of industry, problem of industry, and what is IP-XACT. Section III describes flow of tool, and how each stage can be developed. Tool's constraints are explained in section IV. Section V explains the system architecture and implementation. Section VI presents results of tool in many aspect

2. BACKGROUND

2.1. Integrated Circuit

Integrated Circuits can be defined as a circuit in which all or some of the circuit elements are inseparably associated and electrically interconnected so that it is considered to be indivisible for the purposes of construction and commerce. [1] Because of ICs generally made of purpose, manufacturers can integrate components permanently and optimize connection between components. Due to these indivisibility, ICs can be manufactured cheaply and work faster with optimization.

Nowadays, ICs are used in wide variety of device. Almost any electronic equipment on market use some kind of IC as a component. These ICs generally made by third party companies like Qualcomm, Broadcom, AMD, and go on. That's the one of ideas behind the IC production. An electronic company is not necessarily to expert on every single part of its device. Using chips of third party companies also have advantages too. Since these companies specialize on some purpose, they can manufacture these ICs cheaper. These ICs usually work faster than general purpose ones, and have unique designs and features.

2.2. Intellectual Property Core

With growth of integrated circuit industry, the companies want to hold their chips and its design layout's rights. That is how the intellectual property core term born. Briefly, Intellectual property core –shortly IP Core or IP block, is a functional hardware module which can be reused for more complex IC design.

IP cores have huge impact on chip industry. Licensor companies-vendor, spread their development costs to several chip maker, so chip makers can buy and develop IP cores cheaper. Therefore, chip makers have a chance to spend more resource on developing their chips' features. These chain cost reductions are one of the key factor which makes Moore Law persistence. [2]

Today, IP cores have wide variety of functional blocks. Processors are the brain of an electronic device, thus they are the most popular Variety of interface also licensed as IP blocks on market such as USB, Ethernet, and LCD display. Finally, there are some hardwired IP cores licensed too like MP3 decode unit, Fast Fourier Transform (FFT) unit.

2.3. Electronic Design Automation

Many years, IP cores integrated and routed by hand. Basically, whole design of integrated circuit was done by manually. It come up with huge development costs, and it is almost impossible to test an IC without manufacturing it. With the development of computer technology, companies want to take out these costs from the equation. New software was developed for this purpose. They generally called EDA tools.

EDA is initialism for electronic design automation which sometimes can be referred as Electronic Computer-Aided Design (ECAD). EDA flow is usually consist of four main which are design, synthesis, place and route, and finally documentation.

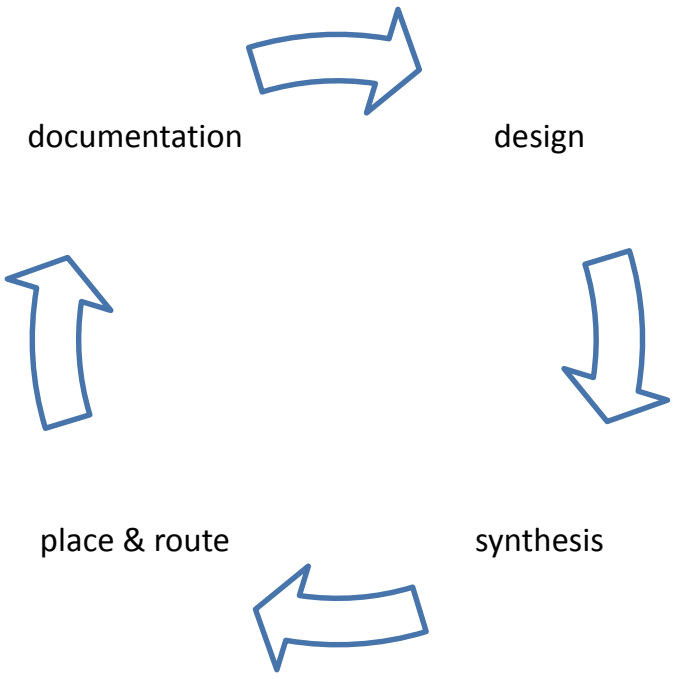

Figure 2.1: Chip development flow

IP reusing at design is the most effective way of developing new chip. However, reusing of IP cores is not always easy. Since EDA tools mostly developed by private companies, they are tend to use their IP core standard. Even look like there is standard exist, actually it does not. Most of IP cores especially register and memory maps, served in HTML, CSV, Spreadsheets and many other documents.

2.4. IP-XACT

Processing IP cores which served in various document types, is increasing to cost of EDA tools. Software companies spend enormous time to developing an algorithm for each document type. The problem is not always that much simple. Since the companies use their own standard in each document type, there are not enough consistency between documents. It makes hard to use EDA tools. Companies either convert these files to a format which compatible with their tools, or manually enter the EDA tools.

At 2003, The SPIRIT consortium launched to resolve the problem of multi-vendor system design. After the couple year work, they come up with an XML-based standard which called IP-XACT. When SPIRIT achieve the complete IP-XACT specification v1.2, IP-XACT transferred to IEEE and IEEE P1685 process began. [3]

The purpose of IEEE P1685 is defined as "Providing a well-defined XML Schema for meta-data that documents the characteristics of Intellectual Property (IP) required for the automation of the configuration and integration of IP blocks; and to define an Application Programming Interface (API) to make this meta-data directly accessible to automation tools." [4] The IP-XACT approved by IEEE at 2009 and published at 2010 [5]

IP-XACT standard defines some set of constraints for each component, interface and etc. Every IP-XACT file must follow these constraints for validation. All constraints are marked either mandatory or optional. Mandatory fields must be define, otherwise IP-XACT file will not be valid. These constraints explained in IP-XACT document which can be accessed on IEEE web site, and XSD files published on Accellera [6] – SPIRIT Consortium merged with Accellera at 2009.

2.4.1. Data Types

IP-XACT standard have couple of data type definition for fields. Some of these types inherited from XML standard, and some of are defined with regular expression rules. All used type definitions will be listen in this section. For organizing purpose, these type definitions are separated with two namespace which are ‘xs’ for XML, and ‘spirit’ for IP-XACT data types.

2.4.1.1. xs:string

String data type may contain any Unicode character that are allowed by XML. There are only two exceptions for it. Less than (<) and ampersands (&) symbols must be escaped respectively with ‘<’ and ‘&’

2.4.1.2. xs:Name

Name is one of the data type of XML. It derived from xs:string, so it should follow all rules of string. Name type have also several extra rules. It must be consist of letters, digits, underscores (_), colons (:), hyphens (-) and periods (.). Name type cannot start with digit, hyphens and periods.

2.4.1.1. xs:positiveInteger

This data type can only consist of with digits and optionally can have a leading plus sign (+). Leading zeros also permitted too. 1 (one) is the minimum value that it can take.

2.4.1.1. spirit:scaledInteger

Scaled integer can take all integer value. All recognized string by java.lang.Long.decode() are also supported in this data type. Additionally, it support scale suffix which can be written in upper case or lower case. The permitted suffixes are K for kilo (2^{10}), M for mega (2^{20}), G for giga (2^{30}), and T for tera (2^{40}).

2.4.1.2. spirit:scaledNonNegativeInteger

A scaled non negative integer can take a value in {0, 1, 2 ...}. Mathematically, it defined with Z^* or N. Since, this data type derived from spirit:scaledInteger, all applicable rules for it are enforced for scaled non negative integer. The magnitude suffixes can be used for this data type.

2.4.1.3. spirit:scaledPositiveInteger

Scaled positive integer are the integers in Z^+ or N^+ set $\{1, 2, \ldots\}$. All parent data type's constraints are inherited to scaled positive integer. The magnitude suffixes can be used for this data type too.

2.4.1.1. spirit:accessType

Differently from the others, access types are enumerated values for purpose. It represent accessibility of a memory unit or specifically registers for this study. There are several choices for this type.

- **read-write:** Both read and write operations can be done. Read operation return the value of unit, and write value changes value
- **read-only:** Only read operation is allowed for the register. Write transaction is forbidden.
- **write-only:** Only write operation can be done for this unit. Read operation returns undefined result.
- **read-writeOnce:** Same with read-write, both read and write operations applicable on this register. Unlike the read-write, after a reset event, only the first write operation can be done on this register.
- **writeOnce:** Like the read-writeOnce, but differently, read operation returns undefined result.

2.4.2. Registers

Because of IP-XACT aims to standardization of IP blocks, it have more than 50 descriptions. These descriptions are categorized as interface, component, design, abstractor, generator, and configuration. In this study, only register components are focused.

Registers are simple and limited memory blocks of electronic systems. Typically, they are used for configuring the system, reporting of system's status, and buffering data. Like other components, IP-XACT defines some constraints for register. Since IP-XACT is an XML that is also based on tree structure, all registers must belong a parent element. For general use, register blocks should be wrapped by "registerFile". There are also some child

elements are defined under the register blocks which can be seen at Figure 2.2. Mandatory elements of register are explained at the end of section.

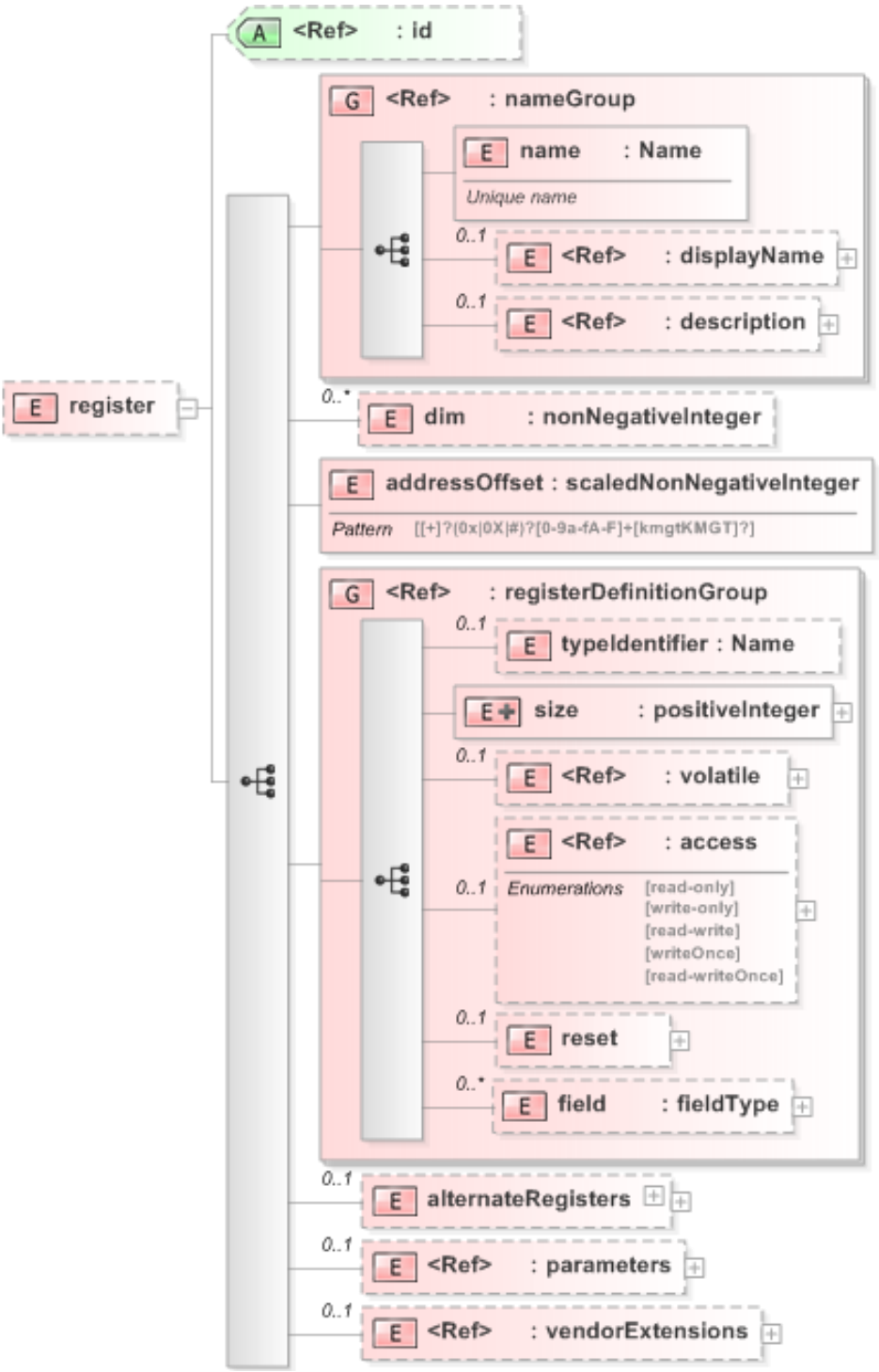

Figure 2.2: IP-XACT register schema

Register name is defined under “name” element and it is mandatory for every register. Name field is defined with xs:Name data type.

Address offset is another mandatory element for registers. It defines the how much offset register have from the base address of block. The offset must be defined in scaled non negative integer.

Finally size of the register must be defined. Size defines the how many bits will reserved for register. For the register size, scaled integer cannot be used, it must be defined as positive integer.

2.4.3. Register Fields

Field elements are the smaller bits of a register. Fields are used for defining which part of register is responsible about which function. The field element contains following mandatory elements. All elements of field contain can be seen at Figure 2.3.

Name element is describing the name of register. Like the register name, field name also defines with xs:Name data type.

The offset of field where the bit field starts is defined at bitOffset element. The element is restricted with non-negative integer type.

The size of the field is defined with bitWidth element. Type of bitWidth is positive integer.

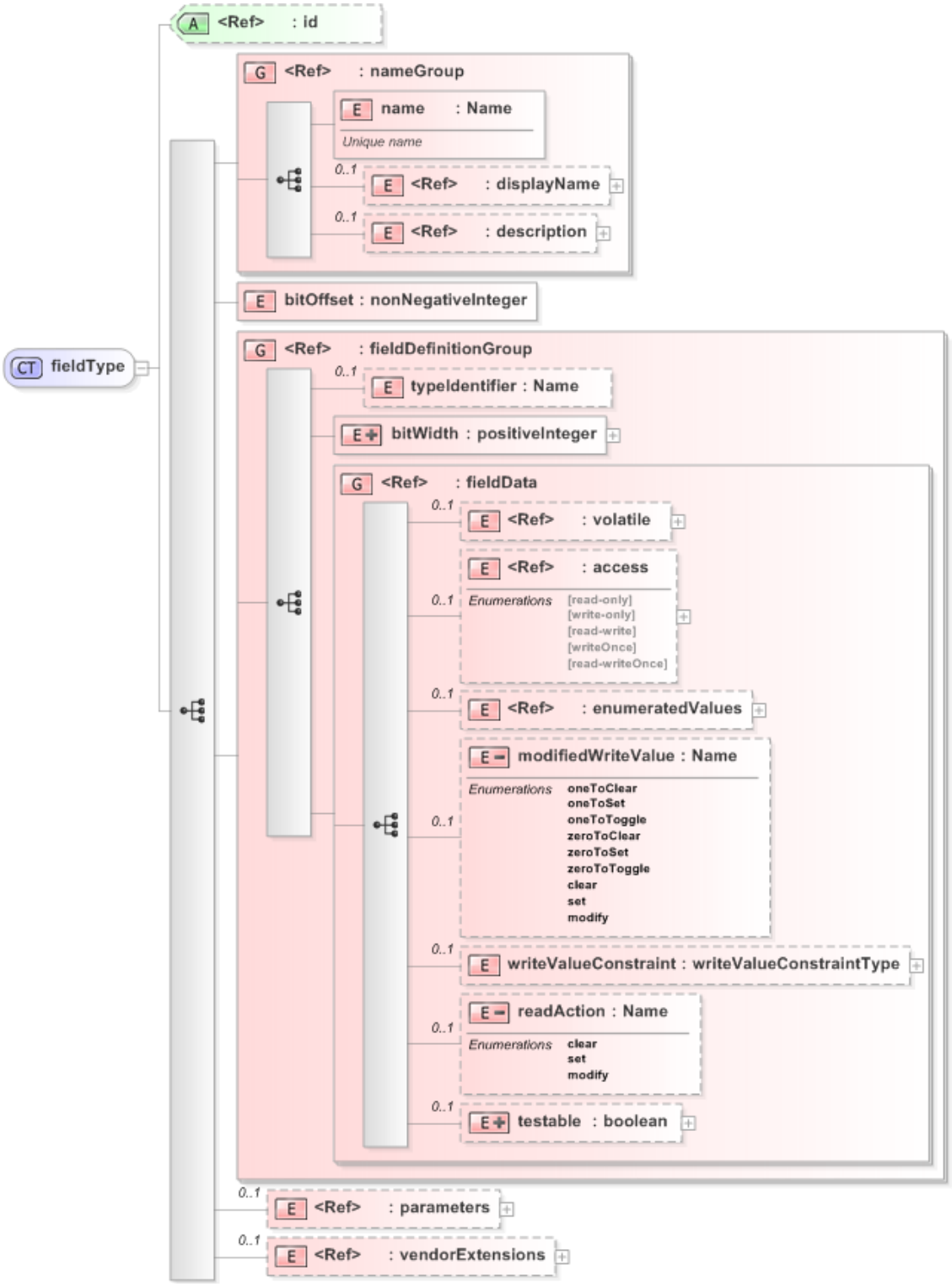

Figure 2.3: IP-XACT field schema

2.5. Problem Definition

Even though, document formatted IP cores are used in industry, they are not satisfactory for automation. These IP cores can be integrated with couple of expensive tools, or manually entered by an operator. These process make designing a chip more expensive and slower than planned. Considering the present ICs push the seven billion transistor limit [7], designing a chip is getting complicated every day. IP-XACT solves all these integrating problems, all existing IP cores must be converted to new IP-XACT format. That is why, there are lots of company wants to convert owned IP cores with fast, reliable method without spending thousands of dollar. Dialog Semiconductor is one of these companies. This study is supported by them. They are not only support the study, but also they collaborate while defining the project requirements. These constraints can be examined under coverage, reliability and performance concerns.

Firstly, the project aims to develop an application that is able to convert most of IP cores to IP-XACT format. Since, the variety of document format are enormous, popular spreadsheet formats are the only focused ones. Because of covering all spreadsheet template is technically impossible, a general template for spreadsheet files must be defined. The tool must be able to accept popular spreadsheet formats which are in this template format, as an input. Another coverage concern is about environment. The tool must be run on UNIX based system specially named as Red Hat and Ubuntu, since the Red Hat is used on most of company's server, and Ubuntu supports the most of hardware and software on the market.

Secondly, output file always must be valid for IP-XACT standard. If there is a mandatory field is missing in spreadsheet, tool must warn operator with an error message. Otherwise, if no error is given, the output file must be satisfy IEEE 1685-2009. Validation tests must be done with a third party tool for increasing the reliability. Abstractor Editor [8] is developed by EDAUtils for industrial usage. It can edit IP-XACT file and validate it. Output of register map generator can be tested with Abstractor Editor.

Lastly, performance of the tool must be satisfactory. In this sense, generating IP-XACT core must be completed in reasonable time like 5 seconds. For testing this time duration, it must be done with several register core which are having different configuration.

3. METHODOLOGY

3.1. Determining File Type

There are two possible way to determining a file type, filename extensions and internal metadata which is also called file headers.

Filename extensions, are the most popular method used by many operation systems for determining file type. When a filename split with dot ".", last group of letters called extension of that file. For instance, let "some picture.2014.jpeg" is full name of file. "some picture.2014" is the filename and "jpeg" is extension of file. However, file extensions are not always reliable. Sometimes, files saved without extension, even worse saved with wrong extension.

Internal metadata is the second way to identify file format. That method usually uses the first couple of bytes file determine its type. Lots of image and archive format use these method. Despite internal metadata look like bullet proof solution, it have huge drawback. Since the file have to be opened each time when determining its format, the operating system or programs uses more memory and spend more time.

Even though only spreadsheet formats are covered in this study, still there are too many spreadsheet formats like MS Excel (.xls, .xlsx), open document format (.ods), and comma separated values (.csv)

3.2. Parsing Spreadsheet

Each spreadsheet format have unique file structure to storing information. Some of them store information as ASCII file, and other store as binary file. That is why, single parser is not enough for all covered spreadsheets format.

Although high level programming languages try to provide internal library for parsing these documents, internal libraries are not enough for parsing these variety of spreadsheet formats. External libraries are developed by third parties for this purpose. Almost all popular

programing languages are supported by these libraries. So, after identifying file format, any of related external library can be used as parser.

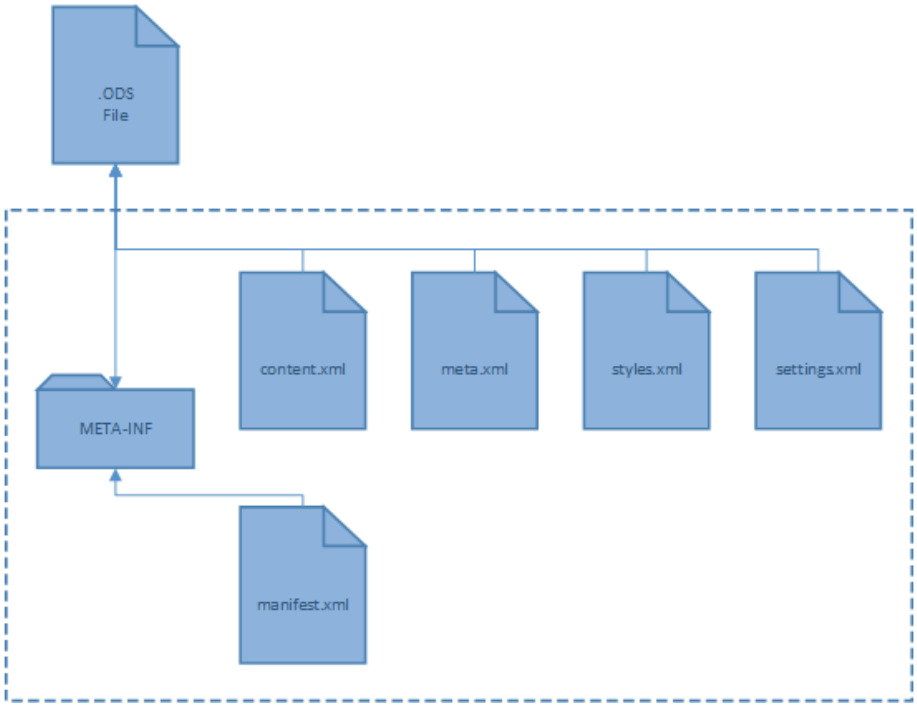

Figure 3.1: Open Document Spreadsheet (.ods) file structure

3.3. Generating Result

IP-XACT is a XML based file, so there are two convenient method to produce file. Printing information algorithmically and using an external template engine are these practices. Each approach having its advantages and disadvantages on the other one.

```
<html>
	<head>
		<title>[% title %]</title>
	</head>
	<body>
		<h1>Articles</h1>
		[%  FOREACH article IN articles %]
			<div>[% article %]</div>
		[%  END %]
	</body>
</html>
```

Figure 3.2: Example template file for html output

Exporting output in algorithm is most classical way to saving data in file. Despite its performance advantages, this method usually avoided in projects. Mixing algorithm and output is not best practice, since changing something in algorithm can change output too. It is also brake down the abstraction principle in software engineering.

Template engines are specifically designed with considering software engineering rules. Despite of its extra computational load, template engines are quite fast, and not performance killers. Additionally, they reduce lots of development cost. For clarification, different pages can be generated, only with setting couple of parameter at Figure 3.2. As can be seen, the algorithm and output format completely separated by each other.

3.4. Scripting Language

Although any of script language can be used in this study, Perl come forward. Reasons of why Perl is good choice for this study are

- Perl supported by almost all UNIX based system
- Perl can manipulate string slightly faster than other options
- Perl's regular expression engine is have more feature than POSIX standard of regular expression
- Including the Dialog Semiconductor, Perl is one of the most popular scripting language at IC industry.

For web interface, there are many server side languages that can be used for this study like PHP, ASP, and etc. As a server side language, PHP is chosen for several reason. Firstly, PHP can easily setup on UNIX based systems. Essentially, one of concern of this project is compatibility with UNIX based systems. Second reason is, running an external script in PHP code is not hard. Because of the architecture of project that will be explained another section, register map generator script must be externally run in web service. Finally, as author of this study, I am more familiar with PHP.

4. ANALYSIS

The tool must be satisfy several requirement to be accepted as useful for industry. To achieve this purpose more efficiently, a collaboration is done with Dialog Semiconductor and most of requirements is defined after this collaboration. In this document, the significance of requirements is specified by using “MUST”, “MUST NOT”, "REQUIRED", “SHOULD”, and “MAY” keywords as described in RFC 2119. [9].

As it briefly explained at section 2.5, the requirements of tool can be examined under three sections that are coverage, reliability, and performance. The following list made of the some featured requirements of study, all of this requirements will be explained in related sections.

- The application MUST take a spreadsheet file and generate an IP-XACT file with respect to input file
- Generated IP-XACT file MUST satisfy requirement of IEEE 1685-2009 standards. Some of related requirement of IP-XACT explained at section 2.4
- IP-XACT file MUST be generated least than 30 second
- Memory consumption of application MUST NOT be exceed more than 5 MB
- The application MUST run on UNIX based system, especially on Linux distributions such Red-Hat, and Ubuntu
- It SHOULD be written in Perl and use an template engine
- All external libraries SHOULD be installed as local library
- It MAY have an web user interface

4.1. Coverage

Coverage requirements intend to maximize field usage of tool. These requirements composed of increasing number of IP core can be translated, and working on different configuration of system as many as possible.

Older IP cores generally sold in spreadsheet format. The reason of that, spreadsheets are more interpretable by computer as much as how human readable they are. In this sense, the tool MUST be able to generate IP-XACT file from a spreadsheet. Even it is possible to cover all spreadsheet file formats, developing a tool which covers all file format is unnecessary and time loss. The industry was used only a couple of spreadsheet format. Covering only these popular formats is REQUIRED.

Like the spreadsheet file formats, it is not possible to cover all semantic formats. The computer cannot easily find relation between rows and columns even the proper header is given, since the almost all cells are same kind of number. Inferring these numbers requires huge computation power and a machine learning process which is another are of computer science. That is why the tool MUST define a template format for spreadsheet. The template MUST be flexible as much as possible.

Covering only inputs is not enough for this study. The tool MUST be cross-platform. In order to meet this requirement, an interpreted language MUST be used for developing tool. Due to hardware companies make their work flow, and set up workspaces decades ago, most of them uses Red-Hat as a core server, and Perl for automating things. Thus, Perl SHOULD be used for developing this tool, and the tool MUST be run at least UNIX based system like Red-Hat.

4.2. Reliability

Considering the tool is going to be used for standardization, It MUST be reliable for all aspect. Most important requirement about this topic is the output MUST be satisfy constraint of IP-XACT definition. Related IP-XACT definitions are described at section 2.4.

There are three main constraint about IP-XACT. First one is the output file is MUST be wrapped by register file element, so that the EDA tools can use this file. Second constraint is all essential elements of register MUST be defined. If they cannot be defined, an error message MUST be given, so an operator can complete this missing information. Finally, necessary field elements MUST be seen at output too.

Supplying all necessary information is not enough for creating IP-XACT file. The output MUST have some specific parent-child relation as defined in XML schema.

4.3. Performance

As a result of being an automation, tool MUST complete its part with using reasonable resource. In this sense, performance constraints can be time limitations and memory limitations. Since performance of tool is based on input, the constraints must be defined for maximum input size for industry. Generally register files are small, but sometimes it can contain hundreds of register, so 1000 register can be a good limitation for constraints. It also can be assumed an average register is contains four fields. In brief constraints can be defined for a spreadsheet which contains 5000 rows.

Since the tool used for automation, it must be generate output in a certain time. For allowing batch conversion and speeding design, the output of 5000 rows MUST be given in 5 seconds. For the small input files, it should take about 1 second.

As much as being fast, the script should be lightweight, because of it is just a tool in design process. Considering the data size and using lots of external libraries, 50 MB memory SHOULD be used while processing.

5. DESIGN AND IMPLEMENTATION

The tool consists of two part that visualized at Figure 5.1 . First part of tool is a register map generator that takes spreadsheet file generate proper IP-XACT file. The map generator can only run in command line interface, so it need a UI to easy use. Second part is a web interface. Web interface of tool works as a wrapper. It takes the spreadsheet with simple user interface, and pass as a input to register map generator. After that, the output file is displayed as a tree for user because of it is more human readable.

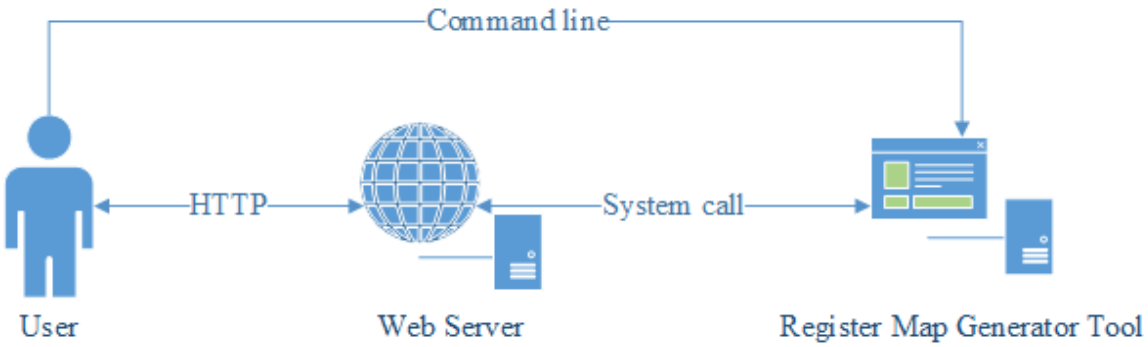

Figure 5.1: Architecture of project

5.1. Register Map Generator

Generating the register map is the main part of this study. All data taken from a spreadsheet which can be formatted as one of xls, xlsx, ods, and csv formats.

The generator can handle multiple spreadsheet format as long as the sheet templates are same. In other word, information which given in spreadsheet should follow some set of rules. These rules should be flexible as much as they possible. The template which is given at Table 5.1, is selected for this study, after collaboration with Dialog Semiconductor.

Table 5.1: IP core spreadsheet template

register name	**register address**				register description
	field name	**field range**	field reset	field access	field description

To make clear these cells, data cells explained as below. The bold cells are the mandatory ones in order to generate valid IP-XACT file.

Register data:

- **Register name** is the name of the register. If this cell empty, the generator assumes it is a field row. It can be take any value permitted in xs:Name
- **Register address** is absolute address of register. Offset of register will be calculated with respect to this value. The default base address used in calculation is 0. The data type of address value is non negative scaled integer.
- Register description is usually explains purpose of register. Description's data type defined as xs:string.

Field data:

- **Field name** is the name of field. Generally it defined with three capital letters. Like the all other names, its data type is xs:Name.
- **Field range** is the cell that defines what the first and last bit of field is. The offset and size of field information is extracted from this cell. Field range can be written in two different formats. All formats can optionally wrapped and separated with any non-digit character. The integer values will be extracted with \d regular expression, so all other non-digit values will be ignored. Some valid examples are shown at Table 5.2.
    - If the field is only consist of single bit, the range value can be defined with position bit.
    - If field size is greater than 1, it must be defined with start bit, and end bit. The order of start and end bits are not important. Lower value always considered as start position.

Table 5.2: Some valid input for field range

Input	Parsed Values (Offset, Size)	
3	3	1
[3]	3	1
position:3	3	1
3:3	3	1
[0:3]	0	4
0,3	0	4
s:0,e:3	0	4
[4:1]	1	4
end 4 | start 1	1	4

- Field reset, is the value of a register at reset. As it defined in IP-XACT definition, the reset value is scaled non negative integer.
- Field access, describes the accessibility of the register. Since the most of IP core uses same abbreviations. Field access cell is not case sensitive, so both upper and lower case presentation is permitted. The following ones are the acceptable ones.
    - RW for read-write
    - R for read-only
    - W for read-writeOnce
    - WO for writeOnce
- Field description, is the explanation of field. It is restricted with xs:string.

The tool consist of three part. A main script for automation process and two classes for storing information. Due to classes used for storing are really bounded and Perl do not have real object oriented class mechanism, these classes merged to one module.

```
main_procedure:
    rows = read(spreadsheet)
    registers = []
    foreach row in rows
       if row[0] not empty then
          append register to registers
          validate_register(row)
          register = new Register(row)
       else
          validate_field(row)
          register->add_field(row)
       end
    end
    append register to registers
    tempate(registers)
```

Main script handles calling parser, extracting data, dereferencing information, and calling the template engine. Dereferencing step is necessary for using template engine, because Perl encapsulates object with bless function[1]. "main_procedure" shows the pseudo code of the main script, and the implementation of algorithm can be seen at "APPENDIX B: MAIN SCRIPT"

5.1.1. Reading Data

First step of tool, read data from spreadsheet file which based on template format. As described before, each spreadsheet format must be parsed with special parser. In order to achieve that, spreadsheet formats are identified by its extension.

After the distinguishing file formats, three main formats come up. These are Open Office, Microsoft Excel, and CSV format. Each spreadsheet format is parsed with own parser.

[1] It used for association between object and package.

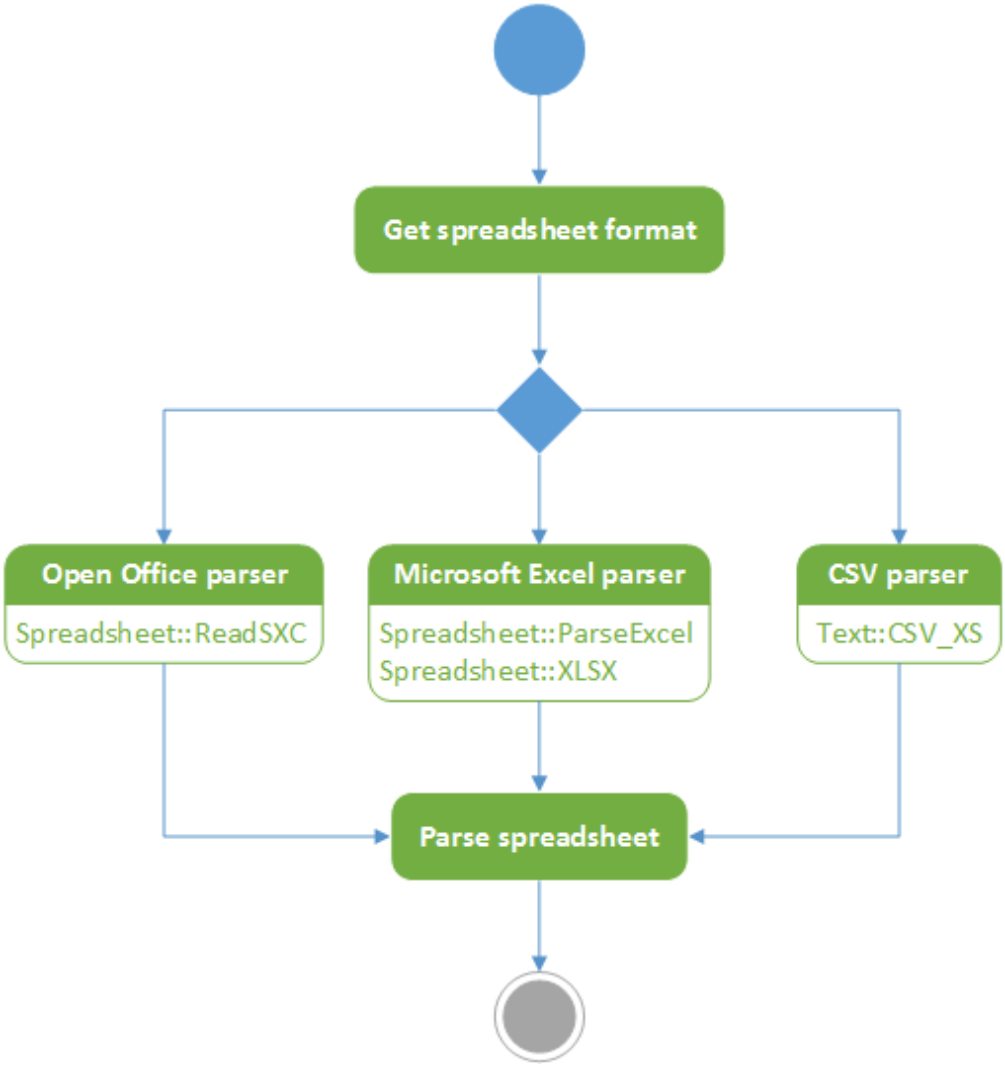

Figure 5.2: Parse flow of spreadsheet

5.1.2. Extracting Information

When parsing stage completed, all parsed row must be processed row by row. The detailed steps can be seen at Figure 5.3. The reason of that, IP cores do not give every mandatory information directly. Before the computing specific data, the row must be identified as either register or field.

Identification between register and field rows is based on first column – column 'A' at spreadsheet file. Unless first column of row is empty, the row is marked as register. Otherwise it is assumed field row. After labeling rows either register or field, some calculation done with row specific data.

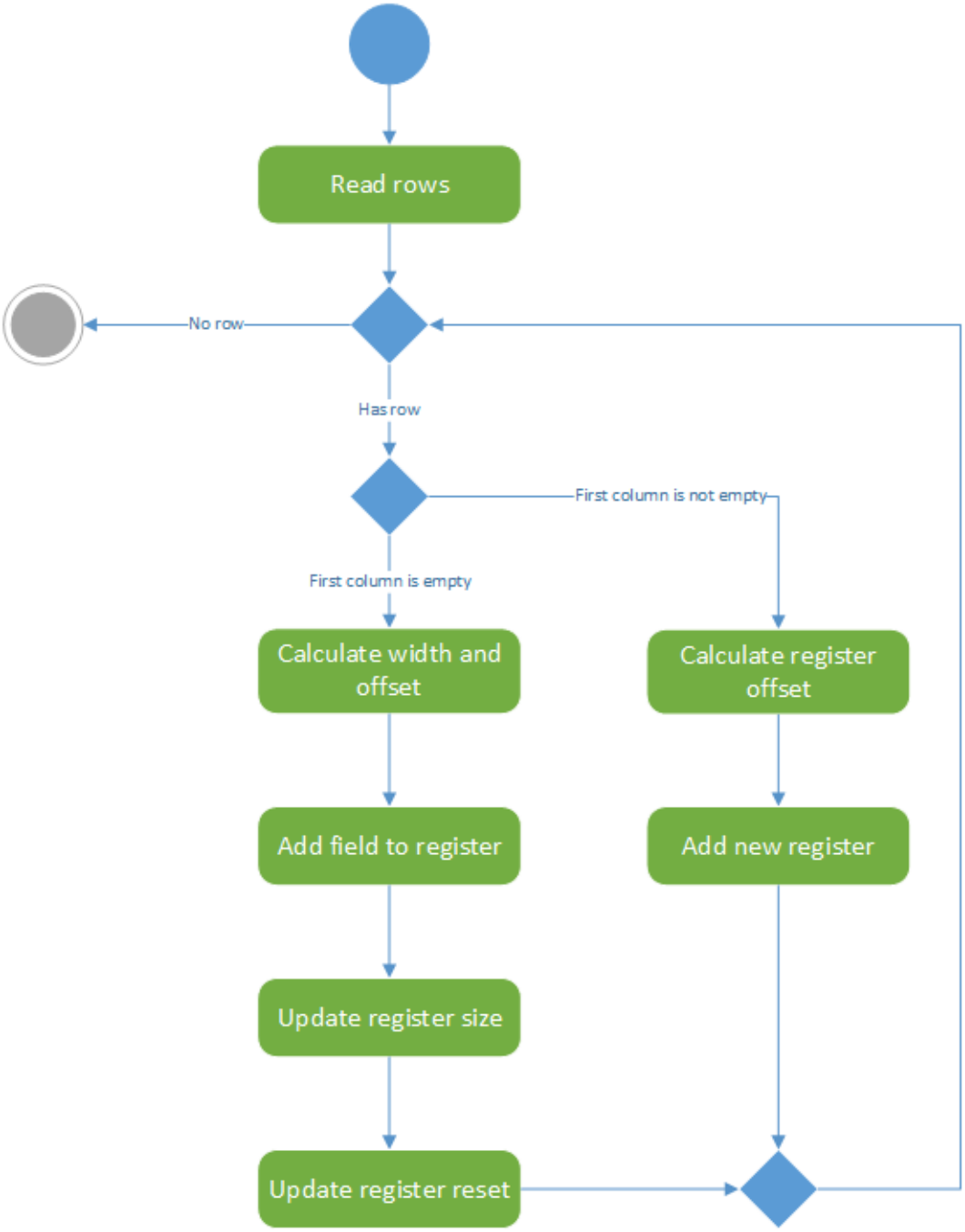

Figure 5.3: Algorithm of extracting process

The tool uses two class, more accurately Perl modules, for storing these information. It design like that because at the end of script, information should be to template engine as an array of object. With classes, extracted data can be easily dereferenced and send to template engine. Design of classes can be seen at Figure 5.4. The implementation of module can be seen at “APPENDIX C: REGISTER MODULE”.

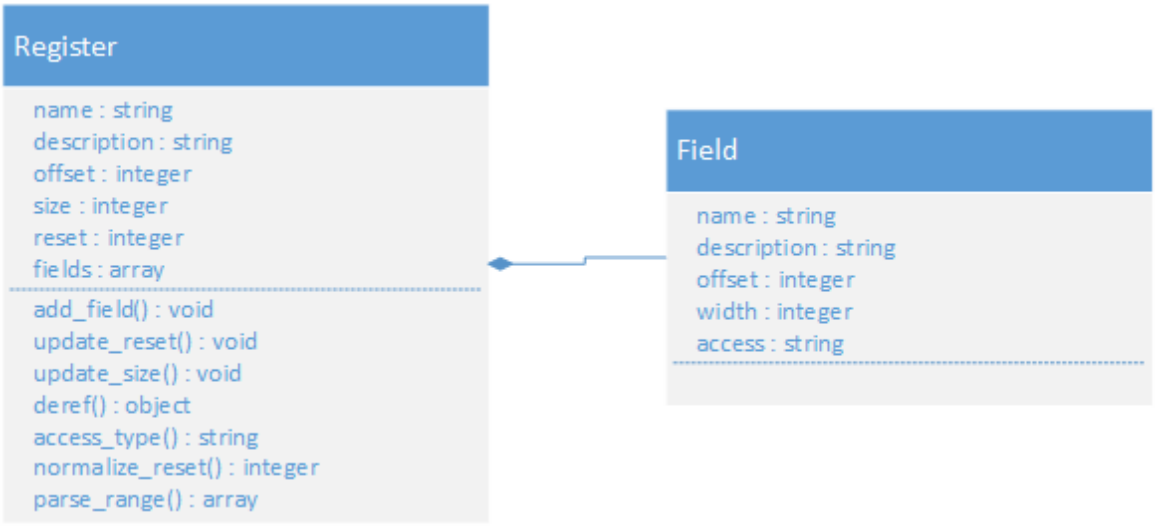

Figure 5.4: Architecture of classes used for storing

For register rows, calculation of address offset enough. Address offset can easily calculated with subtracting base address of register file to absolute address of register.

$$registerOffset = registerAddress - baseAddress \quad \textbf{(5.1)}$$

For field rows, several calculation must be done. Firstly, offset and width of field must be calculated via using range cell data. The offset and width values calculated with equation 5.2 and 5.3 where the x values are parsed integers from range cell. If range field only have one integer, width of field will be assumed as 1, and the offset value is that integer.

$$fieldOffset = \min_{\forall x} x_i \quad \textbf{(5.2)}$$

$$fieldWidth = |x_1 - x_2| + 1 \quad \textbf{(5.3)}$$

Secondly, size of the register must be calculated with field width, since the register row does not contain this information. In order to calculate register size, all field width are summed up with size of field. (Equation 5.4)

$$\text{registerSize} = \sum fieldWidth \quad \textbf{(5.4)}$$

Last calculation is made to find reset value of register, since register field does not need to supply its reset value. Because of every field row only define its reset value, all reset values are scaled with their offset, and then summed up.

$$\text{registerReset} = \sum fieldReset * 2^{fieldOffset} \quad \textbf{(5.5)}$$

5.1.3. Generating IP-XACT

The final step of register map generator tool is building the IP-XACT file. Generating process done with a template engine which called 'Template Toolkit'. The template engine takes two parameters to generate an IP-XACT file (Figure 5.5).

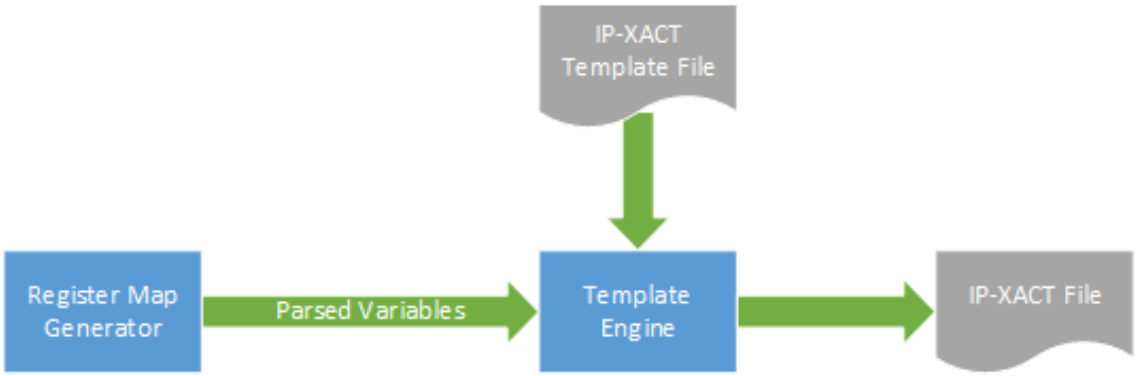

Figure 5.5: Flow of template engine

Even though, IP-XACT template file is static, it has very important role when validating an IP core. The template file must be prepared regarding to IP-XACT documentation. If it have a problem, all generated file will be invalid for IP-XACT. "APPENDIX A: IP-XACT TEMPLATE FILE" contains full template file. Since, the template file have variable that need to replaced, it cannot be validated with any IP-XACT or XSD validator. All necessary test must be done with manually.

5.2. Web Interface

Although, the tool generally will be used with an automated tool, it is nice to have a user interface for it. Web user interface is a frontend part of this project. The user can upload

a spreadsheet file via browser and able to generate IP-XACT file. The generated file can be examined on browser. Figure 5.6 shows flow of web user interface.

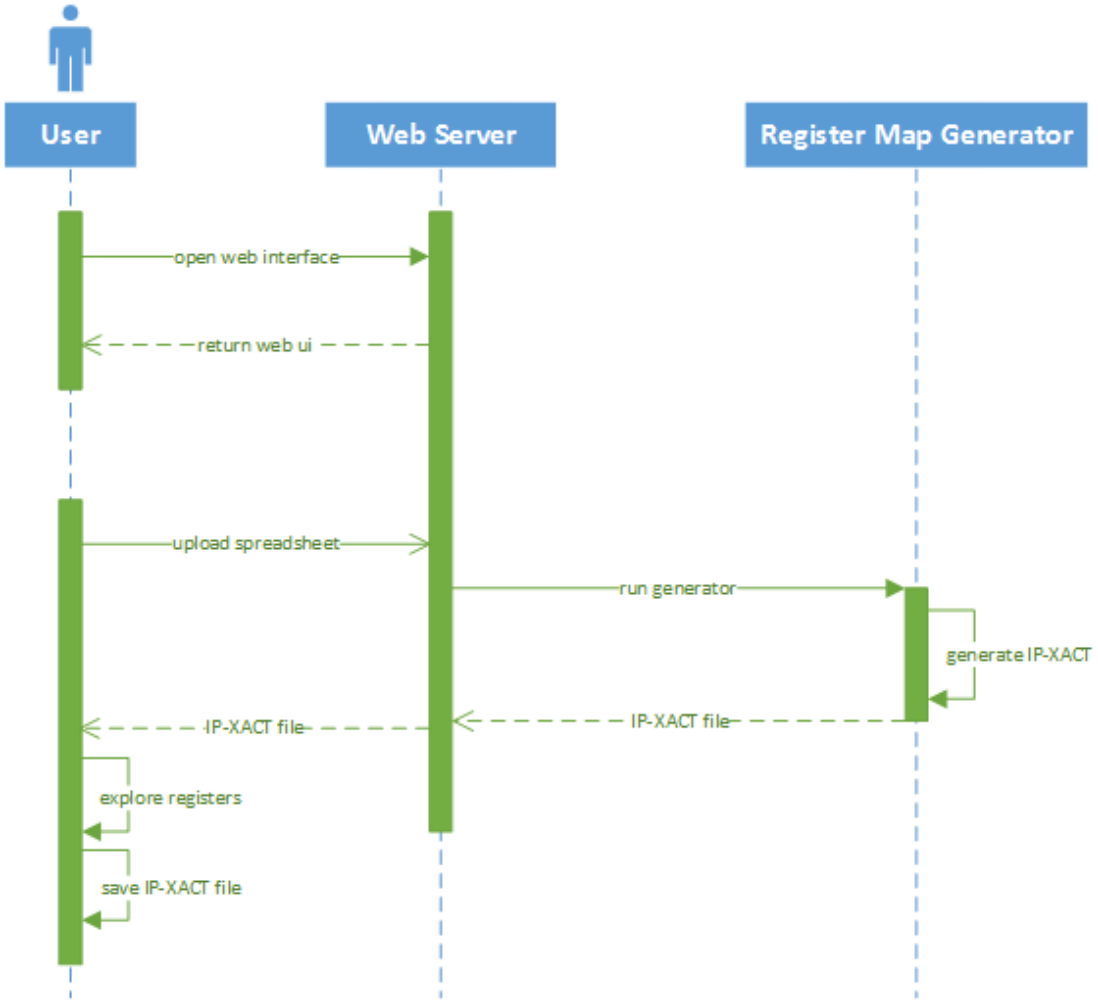

Figure 5.6: Web interface application flow

Web user interface functionality handled by JavaScript. This script handles interaction with user. It is algorithm can be seen at “frontend_procedure”.

```
frontend_procedure:
        uploadButton.clickable = false
        status = “uploading”
        upload file via Ajax request
        status = “processing”
        wait(ajax)
        parse(reponse)
        if response valid then
           generate IP-XACT tree
        else
           list errors
        end
        status = “complete”
        uploadButton.clickable = true
```

At first step user see a screen (Figure 5.7) which is providing to uploading file to server. When user click “Choose File” button native file selector open. After that user select a proper spreadsheet and click “Upload” button.

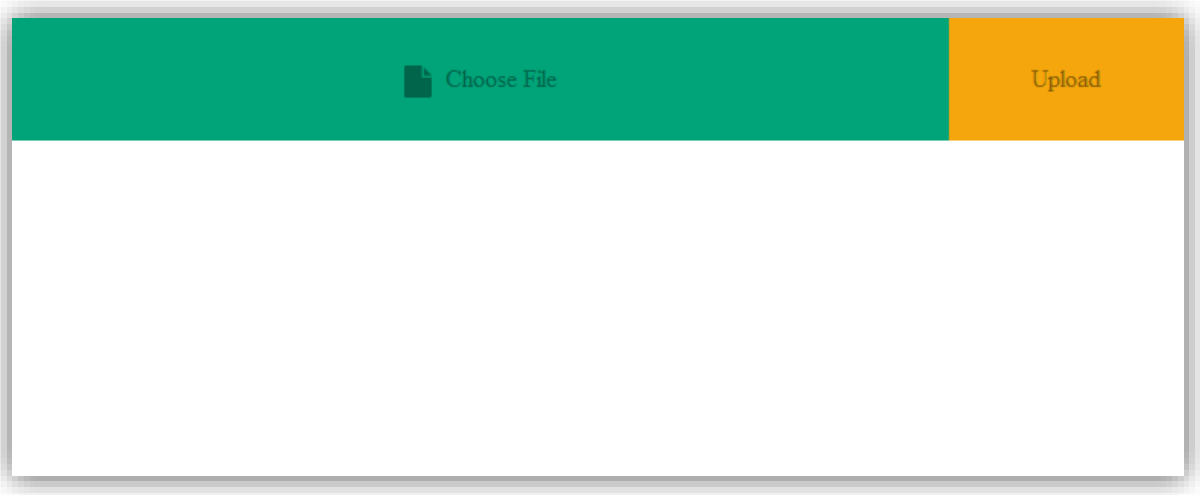

Figure 5.7: Main screen of Web UI

While user uploading process continue, user will be informed. Below the buttons user can see uploading progress with a thin dark grey progress bar that can be seen at Figure 5.8. After upload button pressed, uploading new files blocked for that window until whole generating process complete. The reason of that, web user interface calls IP-XACT map generator tool, and it is impossible to cancel it remotely.

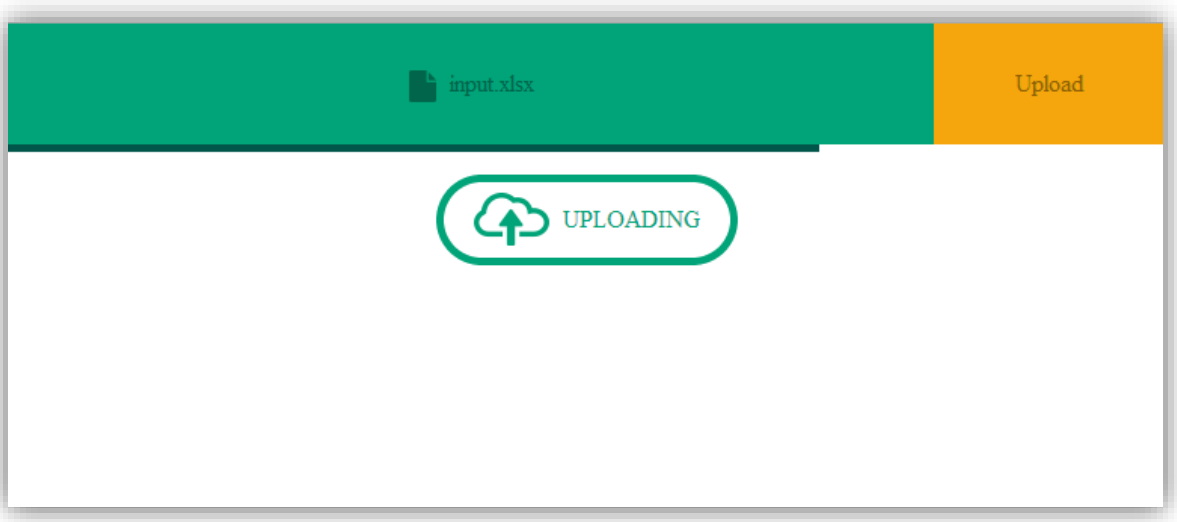

Figure 5.8: Uploading screen of Web UI

When file uploading is complete, that is also mean progress bar reaches the end of window, server start to generate IP-XACT file. At this time user cannot be informed by server progress of it, since the tool not supply progress information about generating.

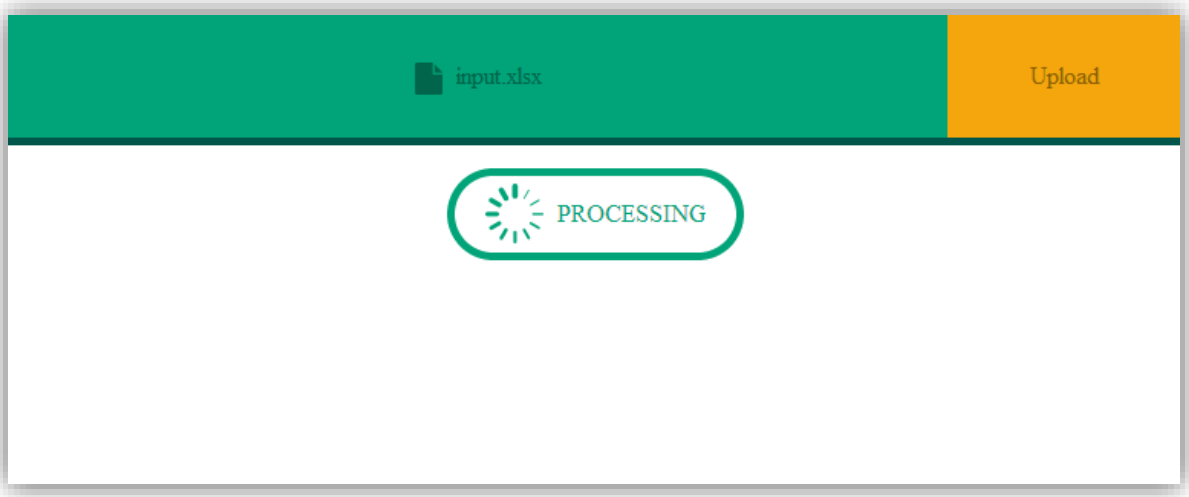

Figure 5.9: Processing screen while server generating IP-XACT

After all generating process done, an XML tree is shown. User is able to explore generated file. If user want to download generated IP-XACT core file, it can be done via clicking “Save” button (Figure 5.10).

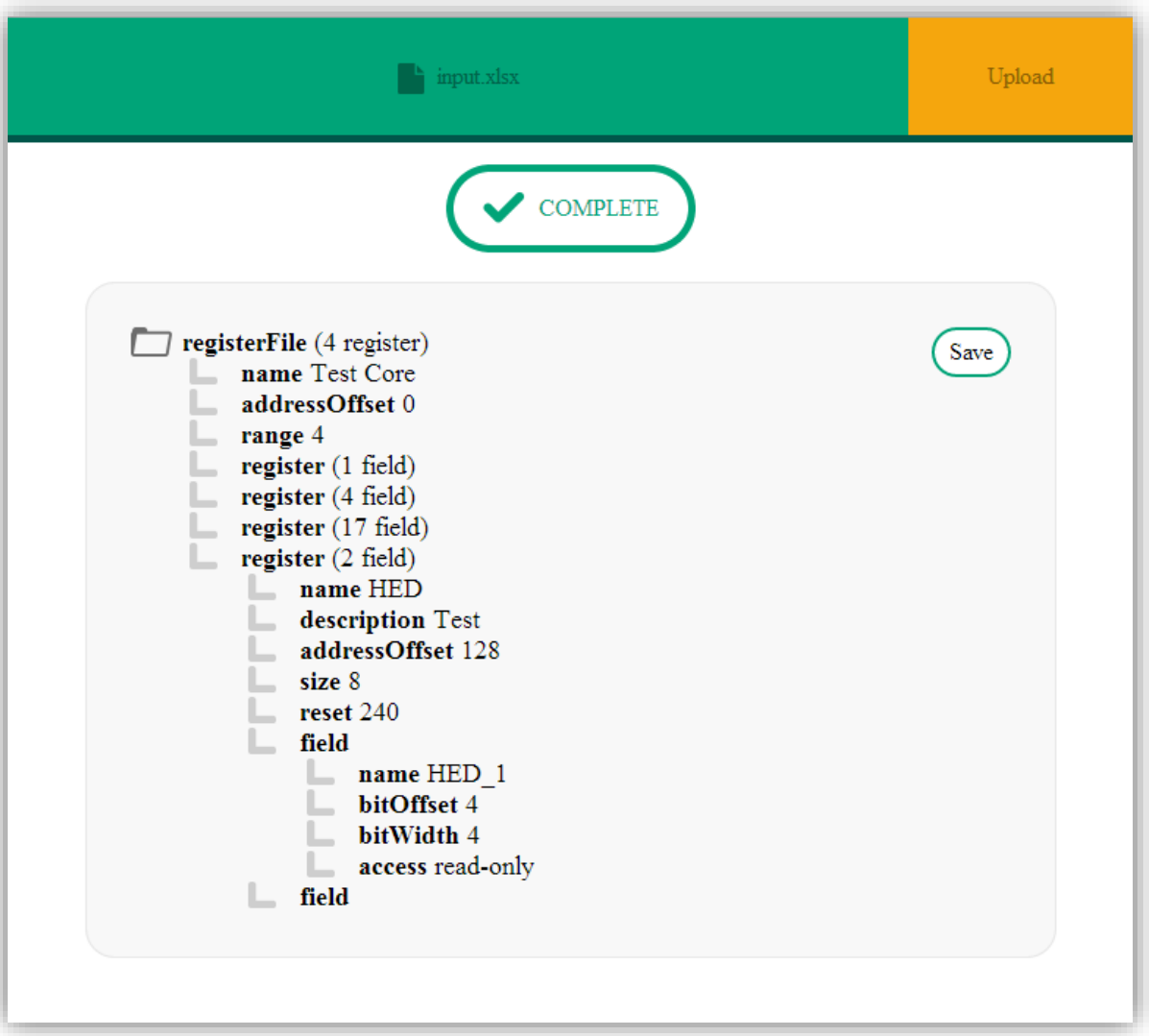

Figure 5.10: Generating successful displaying result to user

5.3. Libraries

In this study, all libraries are open-source and free. Because of the project aims, reducing cost for companies and speeding up the transition of document IP cores to IP-XACT format, the open source libraries are intentionally chosen. As a course of their nature, open-source libraries more reliable than the other ones.

5.3.1. Template Toolkit

Template toolkit is an open-source, flexible and extensible template engine which is written it Perl. Some components of template toolkit is written in C for maximizing its speed. Although, it is written in C and Perl, it is not necessary to know these languages. Template engine has a specific syntax for processing. All variables, conditions, and functions are wrapped between "[%" and "%]". The detailed manual can be seen at its official website. [10]

5.3.2. Spreadsheet Modules

Modules is Perl equivalent of libraries. As mentioned before, the tool use various modules for parsing different spreadsheet format. Five modules are used for this operation.

5.3.2.1. Spreadsheet::Read

Read module is used for determining file format, and calling the module that is proper for that format. Basically, it is a wrapper or universal interface for spreadsheet parsing modules. Figure 5.2 is also shows the flow of this module.

5.3.2.2. Main Parsers

Four main parser modules is used in this study. Since deep detail of these parsers is unnecessary for this study, only names and whatever they parse are listed below.

- Spreadsheet::ParseExcel module is able to parse Excel 95-2003 spreadsheet files.
- Spreadsheet::XLSX is complementary of ParseExcel. It can parse Excel 2007 format.
- Spreadsheet::ReadSXC is designed for extracting data from Open Office spreadsheet files.
- Text::CSV_XS provides to ability of parsing and creating comma-separated values.

5.3.3. jQuery

jQuery is a fast, small, cross-platform, and open source JavaScript library. Manipulating HTML, handling events, making Ajax requests are much simpler and compatible with than native JS. After the releasing of jQuery at 2006, it began to spread rapidly. Nowadays, jQuery is the most popular JS library on the web [11]. Couple of biggest companies, including Google, Microsoft, and IBM, use jQuery too.

6. TEST AND RESULTS

Before the design of system, couple of requires determined. In order to determine either tool is successful or not, it must be tested about these requirements. Tests of tool are consist of three main category.

The tool tested on a VPS[2] server which is hosted by Digital Ocean. Its configuration is

- 512 MB memory
- 1 Core, 2 GHZ processor
- Ubuntu 12.04.3 LTS server edition
- Perl 5.14.2

6.1. IP-XACT Validation

Validation of output is the most important constraint about this study. In order to validate output a third party tool used which is called Abstractor. Abstractor is developed for giving a complete IP-XACT solution needed any design organization. It can do many thing about IP-XACT, however validation feature is only interest.

Validation result of an output taken from register map generator is tool, is shown at Figure 6.1. As can be seen, Abstractor inform validation of file completed successfully. That means the output file follows all rules specified in IP-XACT XML schema.

[2] Virtual Private Server

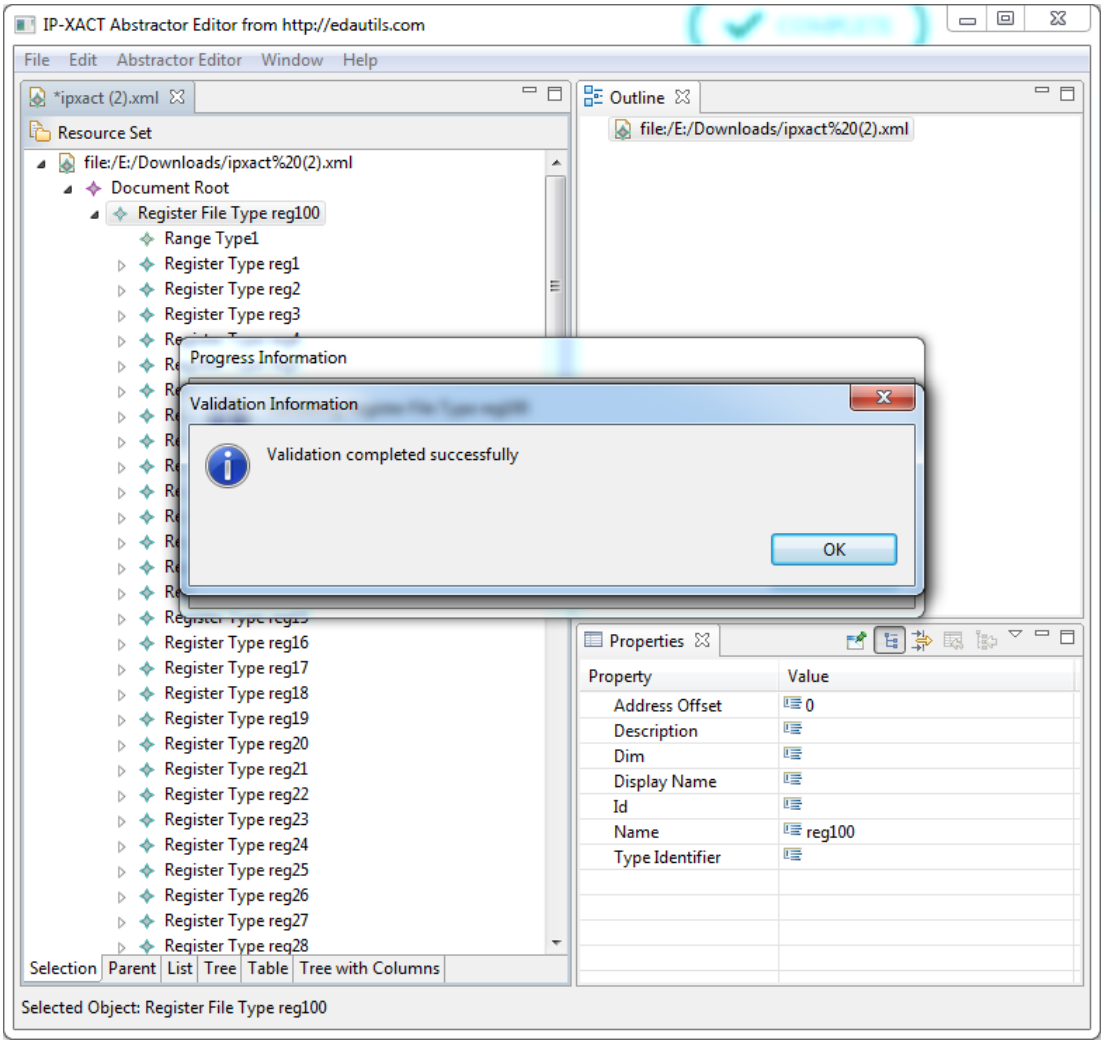

Figure 6.1: Abstractor validation result

6.2. Time

Time consumption is one of the most important requirements for this project, since it will be used in automation process. As is defined at Section IV, it must be less than 5 sec for 5000 row, approximately 1000 register. Since there are lots of small register files, it was assumed generating an average file must be completed under 1 second. As can be seen at Figure 6.2, tests are done with 250, 500 and 5000 rows. The mean of 250, and 500 row is about 0.5 second which is faster than expected.

Figure 6.2: Benchmark results of tool (time)

6.3. Memory

Average memory usage is estimated about 50 MB for large input, like 5000 rows. Memory usage results are shown at Table 6.1. The memory tests are done by Valgrind that is a tool for memory profiling. Valgrind gives the peak memory usage of system in period of execution.

Table 6.1: Benchmark results of tool (memory)

Spreadsheet size (rows)	Memory Usage (MB)
50	15.9
250	17.0
500	18.3
5000	49.5

7. CONCLUSION

Electronic design automation is efficient and cheap way to design and manufacture a chip. However, it has couple of problem when using another company's design due to lack of standard. IP-XACT is especially developed for solving this problem. Still, it cannot solve old standardization problem. In order to get rid of these problems, old IP cores must be converted the IP-XACT format.

In this study, a tool designed and implemented for only register files which stored in spreadsheets. Based on the results, success of tool can be concluded. It meets all requirements which are defined. These requirements are mainly fast, reliable and cheap solution for generating IP-XACT register map. If it is necessary similar tools can be developed for other IP cores. For example a tool which generate CPU file or interface file.

BIBLIOGRAPHY

[1] "Integrated Circuit (IC)," [Online]. Available: http://www.jedec.org/standards-documents/dictionary/terms/integrated-circuit-ic. [Accessed 24 April 2014].

[2] S. E. Thompson, "Power, cost and circuit IP reuse: The real limiter to Moore's Law over the next 10 years," in *VLSI Technology Systems and Applications (VLSI-TSA), 2010 International Symposium on*, 2010, pp. 88-89.

[3] V. Berman, "Standards: The P1685 IP-XACT IP Metadata Standard," *Design Test of Computers, IEEE,* vol. 23, no. 4, pp. 316-317, 2006.

[4] "IEEE P1685," [Online]. Available: http://www.eda.org/spirit-p1685/. [Accessed 10 May 2014].

[5] "IEEE Standard for IP-XACT, Standard Structure for Packaging, Integrating, and Reusing IP within Tools Flows," *IEEE Std 1685-2009,* pp. C1-360, 2010.

[6] A. S. Initiative, "XML Schema," [Online]. Available: http://www.accellera.org/XMLSchema/SPIRIT. [Accessed 11 May 2014].

[7] C. Maxfield, "Xilinx announces world's highest capacity FPGA," 10 November 2011. [Online]. Available: http://www.eetimes.com/document.asp?doc_id=1260468. [Accessed 14 April 2014].

[8] EDAUtils, "IP-XACT (IEEE 1685 -2009) Solutions," [Online]. Available: http://www.edautils.com/ip-xact.html. [Accessed 20 May 2014].

[9] S. Bradner, "RFC 2119," March 1997. [Online]. Available: http://www.ietf.org/rfc/rfc2119.txt. [Accessed 19 April 2014].

[10] T. Toolkit, "Template::Manual," [Online]. Available: http://template-toolkit.org/docs/manual/index.html. [Accessed 18 May 2014].

[11] I. SimilarTech, "JavaScript Technologies Analysis and Web Usage Statistics," [Online]. Available: http://www.similartech.com/categories/javascript. [Accessed 18 May 2014].

APPENDIX A: IP-XACT TEMPLATE FILE

```
<?xml version="1.0" encoding="UTF-8"?>
<spirit:registerFile
xmlns:spirit="http://www.spiritconsortium.org/XMLSchema/SPIRIT/1685-2009">
   <spirit:name>[% name %]</spirit:name>
   <spirit:addressOffset>[% addressOffset %]</spirit:addressOffset>
   <spirit:range>[% range %]</spirit:range>
[% FOREACH register = registers -%]
   <spirit:register>
      <spirit:name>[% register.name %]</spirit:name>
      [%- IF register.desc %]
      <spirit:description>[% register.desc %]</spirit:description>
      [%- END %]
      <spirit:addressOffset>[% register.offset %]</spirit:addressOffset>
      <spirit:size>[% register.size %]</spirit:size>
      [%- IF register.access %]
      <spirit:access>[% register.access %]</spirit:access>
      [%- END %]
      [%- IF register.reset %]
      <spirit:reset>
         <spirit:value>[% register.reset %]</spirit:value>
      </spirit:reset>
      [%- END %]
[% FOREACH field = register.fields -%]
      <spirit:field>
         <spirit:name>[% field.name %]</spirit:name>
         [%- IF field.desc %]
         <spirit:description>[% field.desc %]</spirit:description>
         [%- END %]
         <spirit:bitOffset>[% field.offset %]</spirit:bitOffset>
         <spirit:bitWidth>[% field.width %]</spirit:bitWidth>
         [%- IF field.access %]
         <spirit:access>[% field.access %]</spirit:access>
         [%- END %]
      </spirit:field>
[% END -%]
   </spirit:register>
[% END -%]
</spirit:registerFile>
```

APPENDIX B: MAIN SCRIPT

```
use strict;
use warnings;
use diagnostics;
use File::Basename;
# include local module
use FindBin qw($Bin);
use lib "$Bin/perl5/lib/perl5";
use lib "$Bin/perl5/lib/perl5/x86_64-linux-gnu-thread-multi";
use Getopt::Long;
use Spreadsheet::Read;
use Template;
use lib "$Bin";
use Register;

# get arguments
my %args;
GetOptions(\%args,
        "input=s",
        "output=s"
) or die "Usage ./script.pl --input input.xls(x)";

# mandotary check
die "Missing --input!" unless $args{input};

# check input file  exist
my $input = $args{input};
if ( !-e $input) {
   die "File not exist!";
}
my @error;
# remove extension for register file name
my $basename = basename($input);
(my $registerFileName = $basename) =~ s/\.[^.]+$//;
#if is not valid
unless (validate($registerFileName, 'name')) {
   $registerFileName = "register_File";
   push @error, "register file name error";
}

# Read input File
my $book  = ReadData($input);
my @rows = Spreadsheet::Read::rows ($book->[1]);
# first 6 row is unnecessary
splice @rows, 0, 6;

# calculate base address for registers' offset
#my $address = $book->[1]->{cell}->[2];
#my $baseAddress = min(grep { defined && $_ =~ /^(\d)+$/ } @$address);
# hardcoded since dialog wants like that
my $baseAddress = 0;
```

```
my (@registers, $register, $i);
$i = 7;
foreach my $row (@rows) {
   # if first col is empty, it's a field, otherwise register
   if ($row->[0]) {
      # excel column format
      # 0    1      2      3      4      5
      # name  address                    description
      if($register) {
         #push if we create new register
         push @registers, $register->deref();
      }
      validate_register($row);
      $register = new Register(
         $row->[0],
         $row->[5],
         $row->[1] - $baseAddress
      );
   } else {
      # excel column format
      # 0    1      2      3      4      5
      #      name   range   reset  access  description
      # range format [end:start]
      validate_field($row);
      my ($offset, $width) = Register::parse_range($row->[2]);
      $register->add_field(
         $row->[1],
         $row->[5],
         $offset,
         $width,
         $row->[3],
         $row->[4]
      );
   }
   $i++;
}
push @registers, $register->deref();

# run template and generate xml
my $tt = Template->new({ABSOLUTE => 1});

my $templateData = {
   name => $registerFileName,
   addressOffset => $baseAddress,
   range => scalar @registers,
   registers => \@registers
};

my $size = scalar(@error);
if ($size > 0) {
   print "<errors>\n";
   foreach my $err (@error) {
      print "\t<error>$err</error>\n";
   }
   print "</errors>";
} else {
   if ($args{output}) {
```

```
        $tt->process("$Bin/ipxact_template.tt", $templateData, $args{output}) || die $tt->error;
        print "saved to $args{output}\n";
    } else {
        my $xml = $tt->process("$Bin/ipxact_template.tt", $templateData) || die $tt->error;
    }
}

sub validate_register {
    my ($row) = @_;
    unless (validate($row->[0], 'name')) {
        push @error, "register name is not valid at $i";
    }
    unless (validate($row->[1], 'number')) {
        push @error, "register adress is not valid at $i";
    }
}

sub validate_field {
    my ($row) = @_;
    unless (validate($row->[1], 'name')) {
        push @error, "field name is not valid at $i";
    }
    if ($row->[3]) {
        unless (validate($row->[3], 'number')) {
            push @error, "field reset is not valid at $i";
        }
    }
}

sub validate {
    my ($value, $type) = @_;
    if (defined $type) {
        $type = lc($type);
        if ($type eq "name") {
            return $value =~ /^[a-zA-Z_:]([a-zA-Z0-9_:.])*$/;
        } elsif ($type eq "number") {
            return $value =~ /^([+]?(0x)?[0]*[0-9a-f]+[kmgt]?)$/i;
        }
    }
}
```

APPENDIX C: REGISTER MODULE

```
package Register;
use List::Util qw(min);

sub new {
   my $class = shift;
   my ($name, $desc, $offset) = @_;
   my $self = {
      name => $name,
      desc => $desc,
      offset => $offset,
      size => 0,
      reset => 0,
      fields => [],
   };
   bless $self, $class;
   return $self;
}

sub add_field {
   my $self = shift;
   my ($name, $desc, $offset, $width, $reset, $access) = @_;
   my $field = {
      name => $name,
      desc => $desc,
      offset => $offset,
      width => $width,,
      access => access_type($access)
   };
   $self->update_reset(normalize_reset($reset), $offset);
   $self->update_size($width);
   push @{$self->{fields}}, $field;
}

sub update_reset {
   my $self = shift;
   my ($reset, $offset) = @_;
   # if not defined accept it's all zero
   if(defined $reset && $reset ne "") {
      $self->{reset} += ($reset * (2 ** $offset));
   }
}

sub update_size {
   my $self = shift;
   $self->{size} += shift;
}

sub deref {
   my $self = shift;
   my %uhash = %$self;
   return \%uhash;
}
```

```
#static funtions
sub access_type {
   my $type = shift;
   if (defined $type) {
      $type = uc($type);
      if ($type eq "RW") {return "read-write";}
      elsif ($type eq "R") {return "read-only";}
      elsif ($type eq "W") {return "write-only";}
      elsif ($type eq "RWO") {return "read-writeOnce";}
      elsif ($type eq "WO") {return "writeOnce";}
   }
   return undef;
}

sub normalize_reset {
   my $value = shift;
   if(defined $value) {
      return ($value =~ /^0x[\da-f]+$/i) ? hex($value) : $value;
   }
   return $value;
}

sub parse_range {
   my (@ranges, $size, $offset, $width);
   @ranges = shift =~ /(\d+)/g;
   $size = scalar(@ranges);
   if ($size == 2) {
      $offset = min(@ranges);
      $width = abs($ranges[0] - $ranges[1]) + 1;
   } else {
      $offset = min(@ranges);
      $width = 1;
   }
   return ($offset, $width);
}

#package included
return 1;
```