Automation of Waiver and Design Collateral generation for scalable IPs

Gopalakrishnan Sridhar, Senior Logic Design Engineer/Micro-architect, Intel Technology India Pvt Ltd, Bengaluru, India (*gopalakrishnan.sridhar@intel.com*)

Vadlamuri Venkata Sateesh, Logic Design Engineer, Intel Technology India Pvt Ltd, Bengaluru, India (*vadlamuri.v.sateesh@intel.com*)

Midhun Krishna, Logic Design Engineer, Intel Technology India Pvt Ltd, Bengaluru, India (*midhun.krishna@intel.com*)

***Abstract*— In order to achieve faster time to market for an SOC, the IP design cycle to add new features has to be shortened to the maximum extent possible. A majority of this time is used to rerun design tools and to make sure the design meets the various quality guidelines. This time can be greatly reduced by writing the waivers for various design tools, RTL top file and other collaterals as templates. These templates take the IP top level parameters which are specific to each SOC as inputs and generate the design collaterals specific to them. This paper explains this technique and also analyzes the different challenges faced. Adoption of this flow would save considerable time both for the designer and the integrator of the IP.**

Keywords*— *Design tools, Perl Template Toolkit, Scalability, Intel Proprietary Integration tool, waiver, Spyglass, Debug signals

I. INTRODUCTION

An IP developer has to cater to multiple System on Chip (SOC) customers. Every new SOC who wishes to use the IP have different feature requests. These new features may not be needed in legacy IPs. The IP designer adds new features to the design and parametrizes them in order to maintain legacy. In this way even smaller IPs tend to have many parameters. Each SOC customer comes up with their own version of the existing parameters.

In order to make Scalable IPs, the IPs have to develop features that it thinks may become useful to the SOC in the long run and parametrize them. The approach mentioned in the paper helps to achieve this goal.

The first part of the paper deals with simplifying the waiver generation for the different design tools. Given a new set of parameters for each SOC configuration the designer has to run the tools and develop waivers from scratch to make sure he doesn't waive valid RTL fixes needed for that particular configuration of the parameters.

Instead of doing this activity each time the designer can write the waivers in a scalable fashion based on parameters. In this way he saves time by not reviewing the waivers each time a new configuration comes up. Perl template tool kit (PTT) based waiver files can be generated for each configuration to parametrize the waivers for each of the design tools. In this way by giving a parameter set we can get the waivers for the particular configuration.

In order to write the parametrized waiver file, the designer has to check each warning/error the tool reported, make sure that it can be waived and see the parameter dependence of the parameter. If the reported warning to be waived is based on top level parameter, then the designer can create the waiver easily in the master waiver file by using simple PTT constructs. On the other hand if the warning is based on a derived parameter that's involves a complex equation of many top level parameters, Designer has to rewrite the equation of the derived parameter in his template waiver file. This may be error prone, the paper discusses an efficient way to avoid this by using build summary report generated by Intel Proprietary Integration tool(based on Core tools) to compile the values of the derived parameters that can be used directly in the template files.

A. Perl Template Toolkit

The Template Toolkit [1] is a fast, flexible and highly extensible template processing system which is open source. By using PTT we can create a template file. On running the PTT with the relevant input parameters we can print statements selectively.

Any of the configuration files, waivers and RTL top file can be constructed based on the top level parameters, process technology in a generic way.

PTT is a powerful presentation language which supports all standard templating directives, e.g. variable substitution, includes, conditionals, loops. It has full support for complex data types including hashes, lists, objects and subroutine references. It has a basic syntax and is simple to use.

This paper utilizes the power of the Perl template toolkit to convert the design delivered to the customer in a template form. Once a customer downloads the IP, he generates the design files based on his parameter requirements on the IP. In this way he will see only the waivers for tool runs, top file inputs and outputs that are customized to his configuration.

B. Scalability

In order to ensure quality we have various tool runs to be performed on the design before sign off. Most tools throw warnings and errors that cannot be fixed in the design. These need to be waived and the designer develops waiver files for each tool. The warnings that need to be waived is top level parameter dependent. For example, if a certain piece of code is there based on parameter, the tool waivers for the lines of code inside this generate statement will also have to used only based on this parameter.

Each customers has a different set of parameters. Hence, the waiver generation process has to be redone from scratch in order to avoid false waivers and bugs due to them.

Similarly in case of Debug signal list generations too certain signals will only be used if a certain parameter is enabled. Having a generic signal list is a waste of resource. The same can apply to retention lists and isolation lists for UPF too. Generating waiver files, UPF isolation and retention lists and debug signal lists for each new customer configuration from scratch is a tedious task that takes several weeks. The process becomes more redundant for each new customer.

As mentioned earlier for an IP to become Scalable IP, an IP needs to develop features predicting industry need and parametrize them. Thus an IPs has to become scalable across multiple customers and also have parameters for potential future customers.

Thus having a template based collaterals can reduce the time to generate waivers and collaterals. This becomes easy for the designer to quickly generate the appropriate collaterals if he has the knowledge of the customer parameter list. The same is true for the integrator. The overall time can be reduced from weeks to days.

II. RELATED WORK

A. Template based waiver changes

As discussed earlier PTT requires a list of input variables to be passed to its templates in order to generate output files. When converting tool waiver files to templates, a particular waiver may be based on a top level parameter or a local parameter. The local parameter is internal to the design and may be derived from multiple top level parameters. For example, the local parameter in the below example is derived from multiple top level parameter with a mux logic.

```
localparam                DN_P_STR_DATA_DEPTH    =
                                  (TD_WIDTH==31) ? ((IPPPRIM_AXI_CLOCK_CROSSING_MODE==1) ?
                                (8'h1C + IPP_P_TGT_ISM_PIPE_EN[7:0]): (8'h20 +
IPP_P_TGT_ISM_PIPE_EN[7:0])) :
                                  (TD_WIDTH==63) ? ((IPPPRIM_AXI_CLOCK_CROSSING_MODE==1) ?
                                  (8'h14 + IPP_P_TGT_ISM_PIPE_EN[7:0]) : (8'h10 +
IPP_P_TGT_ISM_PIPE_EN[7:0])) :
                                  (TD_WIDTH==127) ? ((IPPPRIM_AXI_CLOCK_CROSSING_MODE==1) ?
                                  (8'h10 + IPP_P_TGT_ISM_PIPE_EN[7:0]) : (8'h8 +
IPP_P_TGT_ISM_PIPE_EN[7:0])) :
                                  (TD_WIDTH==255) ? ((IPPPRIM_AXI_CLOCK_CROSSING_MODE==1) ?
                                  (8'hE + IPP_P_TGT_ISM_PIPE_EN[7:0]) : (8'h6 +
IPP_P_TGT_ISM_PIPE_EN[7:0]) ) :
                                  (TD_WIDTH==511) ? ( (IPPPRIM_AXI_CLOCK_CROSSING_MODE==1) ?
```

Figure 1. Local Parameter Declaration

If these local parameters have to be used as conditions to generate certain waivers in the waiver template then they need to be recalculated in the template file, using the top level parameters and the result used for the waiver. This is an error prone process since its needs waiver templates to change every time the derivation of this local parameter changes.

B. Intel Proprietary Integration Tool

In order to avoid this error prone recalculation, the power of Intel Proprietary Integration Tool (IPIT) to calculate the local parameters of the top file and print them in its report was used. Intel Proprietary Integration tool is a tool used for the integration of different IPs. It helps integrate protocol signals and makes integration easier.

IPIT prints the values of parameters and local parameters of the top module in IPIT build summery. Any parameter/local parameter which maybe mathematical expression or arrays is calculated by IPIT. The calculated values are printed in this report either in Hex or Dec as shown in figure.

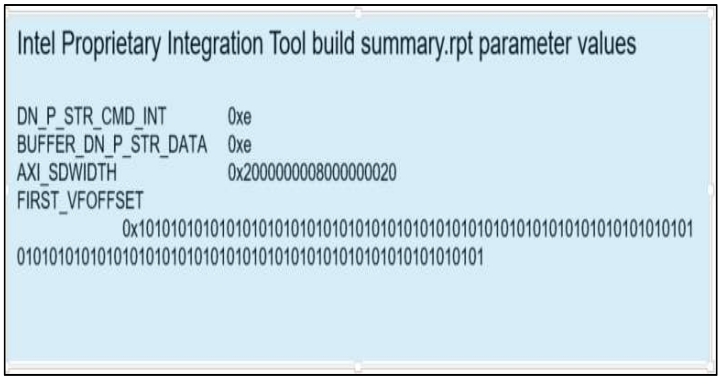

Figure 2: Intel Proprietary Integration tool build summary report

This format of parameter values that is generated from the IPIT build summary reports is easy for the script to extract and decode unlike decoding from parameter/local parameter declarations.

A script for generating the contents of the build summary file on standalone would be sufficient. This can be optimized by working with the IPIT team in the future.

C. *Parameter Extraction Functions*

A reusable PTT script was developed to extract the parameter values from the Intel Proprietary Integration tool build summary. This required various functions to be developed in order to work with various types of parameters.

Certain parameters are multi-dimensional arrays, functions to split them into usable format were developed.

Table I. Sub Routine functions used in PTT

Sub Routines used	**Description fo Sub Routine**	
	Sub Routine	***Description***
1	DEC_TO_HEX(n)	Convert a Dec number into Hex number.
2	f_log2(n)	Number of bits required for a Dec number
3	num_to_array(n,c)	Chop a number "n", each with "c" number of characters and join by ",".
4	DEC_TO_BIN(n)	Convert a Dec number into Hex number and then to Bin number.
5	HEX_STRING_TO_DEC_NUM(num_to_array(DEC_TO_HEX(n),c)).split(',')	Convert a decimal number "n" into an array, the number of characters in each element is decided by "c".
6	size_arr(n)	To get the size of an array.

The converted parameter values will be used by the master template file to generate the required output file for each configuration. An Example is shown in Figure 3.

```
parameter  [NUM_AXIS_PORTS-1:0][31:0] AXI_SDWIDTH        = {32'd512,32'd128,32'd32}

Collage build summary output
AXI_SDWIDTH             0x2000000008000000020

Hexadecimal values converted using Perl script to Decimal value
AXI_SDWIDTH=9444732966289046241312

Output after PTT subroutines
[% AXI_SDWIDTH_arr              =
HEX_STRING_TO_DEC_NUM(num_to_array(DEC_TO_HEX(AXI_SDWIDTH),-8)).split(',') %]

AXI_SDWIDTH_arr.0  → 32
AXI_SDWIDTH_arr.1  → 128
AXI_SDWIDTH_arr.2  → 512

[% FOREACH i IN [0 .. size_arr(AXI_SWID_WIDTH_arr) ] %]
AXI_SDWIDTH_arr.$i
[% END %]
```

Figure 3. Array Splitting

D. *Flow*

The basic flow for the generation of waivers and collaterals is shown in Figure 4. Intel Proprietary Integration tool is run first which generates the build summary containing the values of parameters and local parameters. The build file is used by the Perl Template Toolkit which takes the parameter values to generate the required output file. A Perl script uses this template file to generate the required outputs needed.

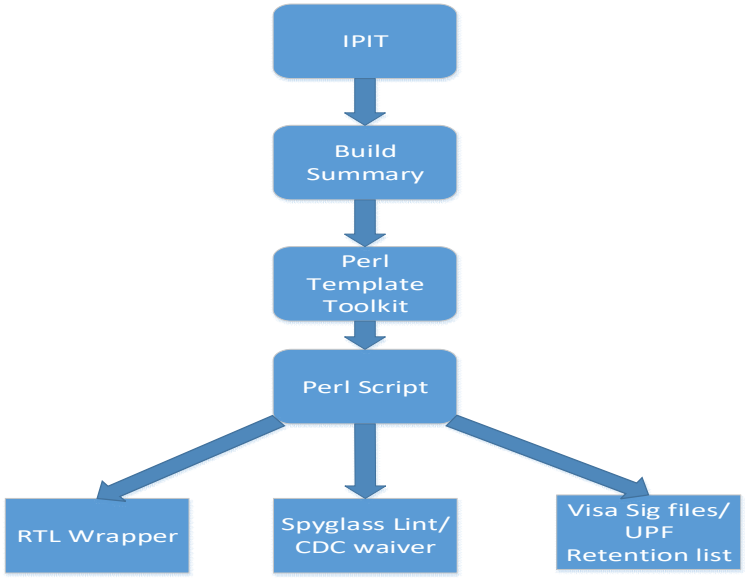

Figure 4. Flow in Creation of waiver collaterals

E. RTL Top File Generation

A significant amount of the IP integration time at the SOC goes to make sure all the input and output ports of the design are connected correctly. Since the IP is generic it has many ports that may not be relevant to the current SOC configuration. The integrator has to review these and make sure he doesn't need to worry about them. By creating the top file as a template RTL ports can be exposed selectively based on the top level parameter. The other inputs can be tied off locally.

In this way the integrator would have to worry about only the ports that would be relevant to his configuration.

III. RESULTS

After enabling scalability waivers and collaterals for multiple customers are generated. The process which takes as much as two weeks, now can be done within a few days. The following table shows a comparison.

The methods discussed in this paper can be reused across different IPs. The template tool kit functions described in this paper can be shared centrally and easily applied for any IP. Perl templates only requires the body and specific conditions for generating the required output file. The body can contain anything from waivers, UPF retention list, Debug signal list or even an RTL code.

Table III. Comparison of time taken with and without scalability

Effort Comparison	**Tools used and effort taken**			
	Tools	***Previous Approach***	***Template based collateral implementation***	***Review***
1.	Lint	Few Days	10 minutes	1 Day
2.	CDC	Few Days	10 minutes	1 Day
3.	Debug Signal File	Few Days	10 minutes	1 Day
4.	UPF retention list	Few Days	10 minutes	1 Day

As mentioned earlier, by collaborating with the Intel Proprietary Integration tool team a standalone script to generate the build summery contents can be created. This would remove the dependence on use of Intel Proprietary Integration tool.

IV. SUMMARY

Perl Template Toolkit allows us to create waivers and collaterals that that makes the IP scalable across multiple customers. This approach can be used in generating waivers for Lint/CDC, RTL top file, Debug signal file generation and UPF retention list generation. More functions for ease of parameter parsing can be developed and be made available for all the IPs to use. Once an IP has been designed based on templates it can reduce a lot of design and integration effort.

ACKNOWLEDGMENT

Yit Pin Lai, Thank you for introducing us to the use of PTT for design scalability.

REFERENCES

[1] David Cross *et.al*, "Perl Template Toolkit," *O'Reilly Media, Inc.,* Dec 2003.