Automation of Waiver and Design Collateral Generation on Scalable IPs

Gopalakrishnan Sridhar,
Midhun Krishna,
Vadlamuri Venkata Sateesh
Agenda Overview

• Problem Statement
• Perl Template Toolkit
• Template Based Scalability
• Intel Proprietary Integration Tool
• Flow Diagram
• Results and Conclusion
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.

Given a new set of parameters for each SOC configuration the designer has to run the tools and develop waivers from scratch.

Due to multiple SoC requests, we need to make IP’s scalable.

Develop features useful to SoC and parameterize them.
Perl Template Toolkit (PTT)

• 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.
Template based Waiver Changes

- 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 parameters shown on the side is derived from multiple top level parameter with a mux logic.
Intel Proprietary Integration Tool

- 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 summary. Any parameter/local parameter which maybe mathematical expression or arrays is calculated by IPIT.
IPIT Advantages

• IPIT prints the values of parameters and local parameters of the top module in Collage build summary.

• Any parameter/local parameter which maybe mathematical expression or arrays can be calculated by collage.

• The calculated values are printed in this report either in Hex or Dec. For Example:

![Parameter Values Table]

```
Intel Proprietary Integration Tool build summary.rpt parameter values

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>DN_P_STR_CMD_INT</td>
<td>0xe</td>
</tr>
<tr>
<td>BUFFER DN_P_STR_DATA</td>
<td>0xe</td>
</tr>
<tr>
<td>AXI SDWIDTH</td>
<td>0x200000000800000020</td>
</tr>
<tr>
<td>FIRST_VFOFFSET</td>
<td></td>
</tr>
</tbody>
</table>
```

© Accellera Systems Initiative
# Intel Proprietary Integration Tool

<table>
<thead>
<tr>
<th>Sub Routines used</th>
<th>Description fo Sub Routine</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Sub Routine</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>1 DEC_TO_HEX(n)</td>
<td>Convert a Dec number into Hex number.</td>
</tr>
<tr>
<td>2 f_log2(n)</td>
<td>Number of bits required for a Dec number</td>
</tr>
<tr>
<td>3 num_to_array(n,c)</td>
<td>Chop a number “n”, each with “c” number of characters and join by “,”.</td>
</tr>
<tr>
<td>4 DEC_TO_BIN(n)</td>
<td>Convert a Dec number into Hex number and then to Bin number.</td>
</tr>
<tr>
<td>5 HEX_STRING_TO_DEC_NUM(num_to_array(DEC_TO_HEX(n),c)).split(‘,’)</td>
<td>Convert a decimal number “n” into an array, the number of characters in each element is decided by “c”.</td>
</tr>
<tr>
<td>6 size_arr(n)</td>
<td>To get the size of an array.</td>
</tr>
</tbody>
</table>
Flow of param/localparam usage in waivers

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

Collage build summary output
AXI_SDWIDTH       0x200000000800000020

Hexadecimal values converted using Perl script to Decimal value
AXI_SDWIDTH=944732966289046241312

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 %]```
Waiver template file example

```plaintext
[% AXI_SDWIDTH_arr = HEX_STRING_TO_DEC_NUM(num_to_array(DEC_TO_HEX(AXI_SDWIDTH),-8)).split(',\')%]
[% UPSTREAM_SUPPORT_arr = num_to_array(DEC_TO_BIN(UPSTREAM_SUPPORT),-1).split(',\')%]

[% FOREACH i IN [0 .. TNUMCHAN] %]

[% IF (US_AREQ_SPLIT_DISABLE == 1) && (NARROW_BURST_ENABLE == 0) && (TD_WIDTH + 1) == AXI_SDWIDTH_arr.$i%]

[% H = NUM_AXIS_PORTS - 1 %]

[% FOREACH k IN [0 .. H %]

[% IF UPSTREAM_SUPPORT_arr.$k == 1 %]

waive -du { {iosf2axibr_axi_slave_cpl} } -msg {Detected unloaded\(unconnected\)input port iosf_axi_top\np_trk_addr\[.*\]
-rule { {UnloadedInPort-ML} } -regexp -comment {CAM_ENABLE}

waive -du { {iosf2axibr_axi_slave_cpl} } -msg {Detected unloaded\(unconnected\)input port iosf_axi_top\np_trk_arsize\[.*\]
-rule { {UnloadedInPort-ML} } -regexp -comment {CAM_ENABLE}
```

………………..
Flow

- Intel Proprietary Integration tool is run first which generates the build summary containing the values of parameters and local parameters.
- Perl Template Toolkit takes the parameter values from build file and generates the required output file.
- Perl script uses this template file to generate the required outputs needed.
RTL Top File Integration

• 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.
## Result

<table>
<thead>
<tr>
<th>Effort Comparison</th>
<th>Tools</th>
<th>Previous Approach</th>
<th>Template based collateral implementation</th>
<th>Review</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.</td>
<td>Lint</td>
<td>Few Days</td>
<td>10 minutes</td>
<td>1 Day</td>
</tr>
<tr>
<td>2.</td>
<td>CDC</td>
<td>Few Days</td>
<td>10 minutes</td>
<td>1 Day</td>
</tr>
<tr>
<td>3.</td>
<td>Debug Signal File</td>
<td>Few Days</td>
<td>10 minutes</td>
<td>1 Day</td>
</tr>
<tr>
<td>4.</td>
<td>UPF retention list</td>
<td>Few Days</td>
<td>10 minutes</td>
<td>1 Day</td>
</tr>
</tbody>
</table>
Key Takeaways

• No need to recalculate manually or through Perl scripts (which are parsed HDL parameter converted to PERL variables)
• Easy to generate waivers for the modules which are instantiated through the loops.
• This Approach and all sub routines can be easily reused across different IPs.
• The time required for spyglass lint scalability for new designs can be reduced to half using this approach.
References

Questions?