Automated and Scalable flow for Multi-Partition RTL Generation

Shruti Narake
Intel Technology India Pvt Ltd
shruti.narake@intel.com

Karthikeya Abhiram Peddiraju
Intel Technology India Pvt Ltd
karthikeya.abhiram.peddiraju@intel.com

Rajkumar Satkuri
Intel Technology India Pvt Ltd
rajkumar.satkuri@intel.com

Alok Anand
Intel Technology India Pvt Ltd
alok.anand@intel.com

***Abstract*—Structural design tools have capacity restrictions that force breaking up large IP subsystems into multiple logical/physical partitions at SOC level. While there is a need for an automated method to create partitioned RTL, it is also highly desirable to keep the content of each partition flexible. To address this problem, an automated and scalable flow has been developed and this push-button flow is agnostic to movement of RTL blocks across partitions. Partitioned RTL generation takes 1 hour, which is 200x reduction in time spent with legacy methods when RTL blocks are moved, added or removed across partitions. This flow also generates the inputs like connectivity Tool command language scripts(TCLs) which provide connectivity information across partitions. This can be used as-is by the customer SoC to decrease IP integration time. This flow has been deployed on one such ARM based subsystem, delivered as 3 partitions to the customer SoC. The flows and methods described in this paper are based on Synopsys Coretools suite and can be used by any IP subsystems which needs to be delivered in partitions.**

***Index Terms*— Partitioned RTL Generation, SoC, IP Partitions, coretools.**

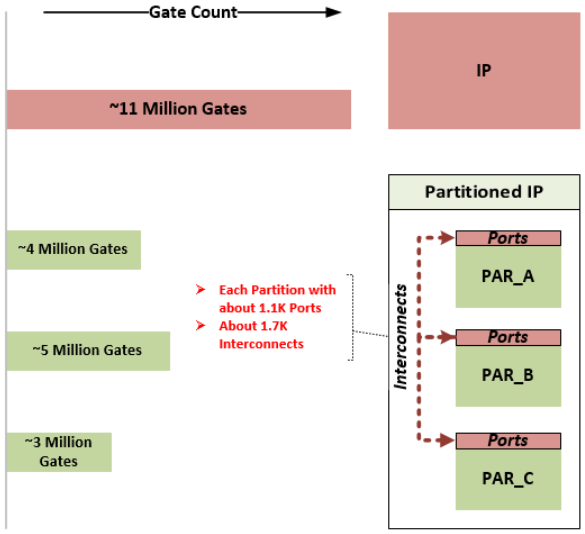

Fig. 1: Sample IP Partition complexity

I. Introduction

A. Background

Complex IPs are being designed to serve various applications like IOTG, ADAS which should provide real time services. The subsystem deployed in this flow is one such complex subsytem with huge gate count. Any such complex IP/Sub-system is preferred to be delivered in partitions to SoC because of following compelling reasons.

- Area and Gate count constraints As a thumb rule, customer SoC usually have a limit on gate count and place-able instances.
- IP Maturity to keep late changing IPs in common partition.
- Clocking and other low power constraints.

Due to the above mentioned reasons, the deployed IP in this flow had to be delivered to the customer SOC in 3 partitions, namely PAR_A PAR_B and PAR_C. These partitions and the complexity involved are shown in the Fig 1.

Partition content cannot be frozen even till the last lap of IP delivery due to the Just-in-time IPs that has to be integrated or sub-IPs that have to be removed or due to the feedback from floor planning, Placement and routing stages. This would require lots of changes which involves creating new partition level ports, removing the unused ones, creating new inter partition connections and removing the unwanted connections as show in the Fig 2.

Manual approach to develop partitioned RTL in fore mentioned scenario is cumbersome, time-consuming and error prone, hence making it a very inefficient choice.

Partition ports are to be frozen early during SoC development during Integration phase, as SoC integration teams expect the consistency to be maintained in the number and the names of the ports among consecutive milestones of IP delivery. This poses a bigger challenge when integrating just-in-time IPs, bug fixes and other feature change requests.

B. Problem Statement

In essence, the problem statement is to develop a flow which

- can automate the generation of partitioned RTL in a very short time
- is agnostic and flexible to any movement of Sub-IPs across the partitions

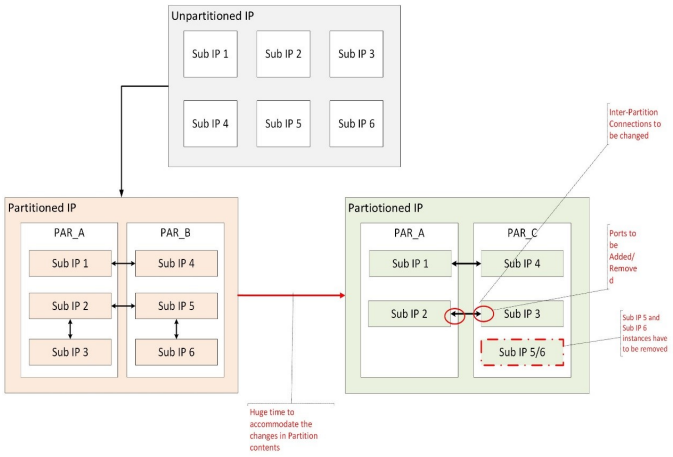

Fig. 2: Efforts needed for a simple change of partition contents

- offers a way to maintain consistency in the number and name of ports after port freeze
- offers control over port name to be readable and conform to proprietary naming convention

C. Solution

A flow, called make flow is designed using Synopsys Coretools, with lots of custom scripting which can solve the issues mentioned in the problem statement. The inputs for the setup and the flow are explained in the next section

II. The 'Make' flow

Tools used in the flow:

- Perl Template Toolkit : Fast, flexible and highly extensible template processing system that increase the configurability and scalability of design.
- Synopsys CoreBuilder : A robust packaging tool that allows designers to capture the knowledge and design intent of the IP in binary format called CoreKits.
- Synopsys CoreAssembler : An IP tool that assembles all the CoreKits and generates the configured RTL.

As shown the Fig 3 the designed make flow can be divided into 3 stages namely -

1) Setup Phase
2) Flat RTL Generation
3) Partitioned RTL generation

A. Setup Phase :

1) Project templates creation:: The following list of the inputs are the only manual and one time setup scripts that are required for the flow.

project_config.pm : This perl module contains all the crucial RTL parameters, Number of instances of a Sub-IP, flags/switches and environment variables defined which are exported to the entire flow.

A snippet of the config.pm setup is shown in Fig 4.

```
NUM_INST_IP_A    =>  "1"
NUM_INST_IP_B    =>  "2"
NUM_INST_IP_C    =>  "2"
```

Fig. 4: Snippet of flow input - config.pm

Instantiate_components.tt : This template contains the TCL commands to instantiate the modules and is an input to CoreAssembler. A snippet of the same is shown in Fig 5

```
for {set x 0} {$x < $NUM_INST_IP_A} {incr x} {
instantiate_component {ThirdParty ThirdParty IP_A_wrapper 1.0} -name i_IP_A_wrapper_$x -noauto
}
for {set x 0} {$x < $NUM_INST_IP_B} {incr x} {
instantiate_component {ThirdParty ThirdParty IP_B_wrapper 1.0} -name i_IP_B_wrapper_$x -noauto
}
```

Fig. 5: Snippet of flow input - instantiate_components.tt

Create_partition.tt : This template groups the Sub-IP instances in the respective partitions. In order move a particular Sub-IP to a desired partition, it must be added to the corresponding partition_list. In the snippet shown in Fig 6 PAR1 and PAR2 are two different partitions of the subsystem.

```
for {set x 0} {$x < $NUM_INST_IP_A} {incr x} {
    lappend PAR1_GATED_LIST "i_IP_A_wrapper_$x"

for {set x 0} {$x < $NUM_INST_IP_B} {incr x} {
    lappend PAR2_UNGATED_LIST "i_IP_B_wrapper_$x"
```

Fig. 6: Snippet of flow input: Illustrating grouping of IPs in different partitions

Ports.tt : This template describes the ports of the unpartitioned/Flat RTL top of the IP. For each port, the direction, width are explicitly mentioned.

```
[%IF clock_gating == "1" -%]
        create_port -log -dir in                [%TOP%]_clkA_[%KEY%]_clk
[%ELSE%]
        create_port -log -dir in                [%TOP%]_clkB_[%KEY%]_clk
[%END%]
```

Fig. 7: Snippet of flow input: Ports.tt

connectivity.tt : These templates provide the Sub-IP to Sub-IP connectivity. It can be observed the flexibility offered in writing the connectivity between two Sub-IPs without any regard to the partition association.

```
[%IP_A_NUM = NUM_IP_A - 1-%]
[%NUM_IP_A = [0..IP_A_NUM]-%]
[%FOREACH x IN NUM_IP_A-%]
create_connection {{i_design_top_0/design_ip_clk_out_IP_A[[%x%]]} {i_IP_A_wrapper_[%x%]/clk}}
create_connection {{i_design_top_0/design_ip_clk_out_IP_A[[%x%]]} {i_IP_A_wrapper_[%x%]/clk_rst}}
```

Fig. 8: Snippet of flow input: Illustrating connectivity between different IPs

It can be observed that all the above mentioned templates are highly scalable.

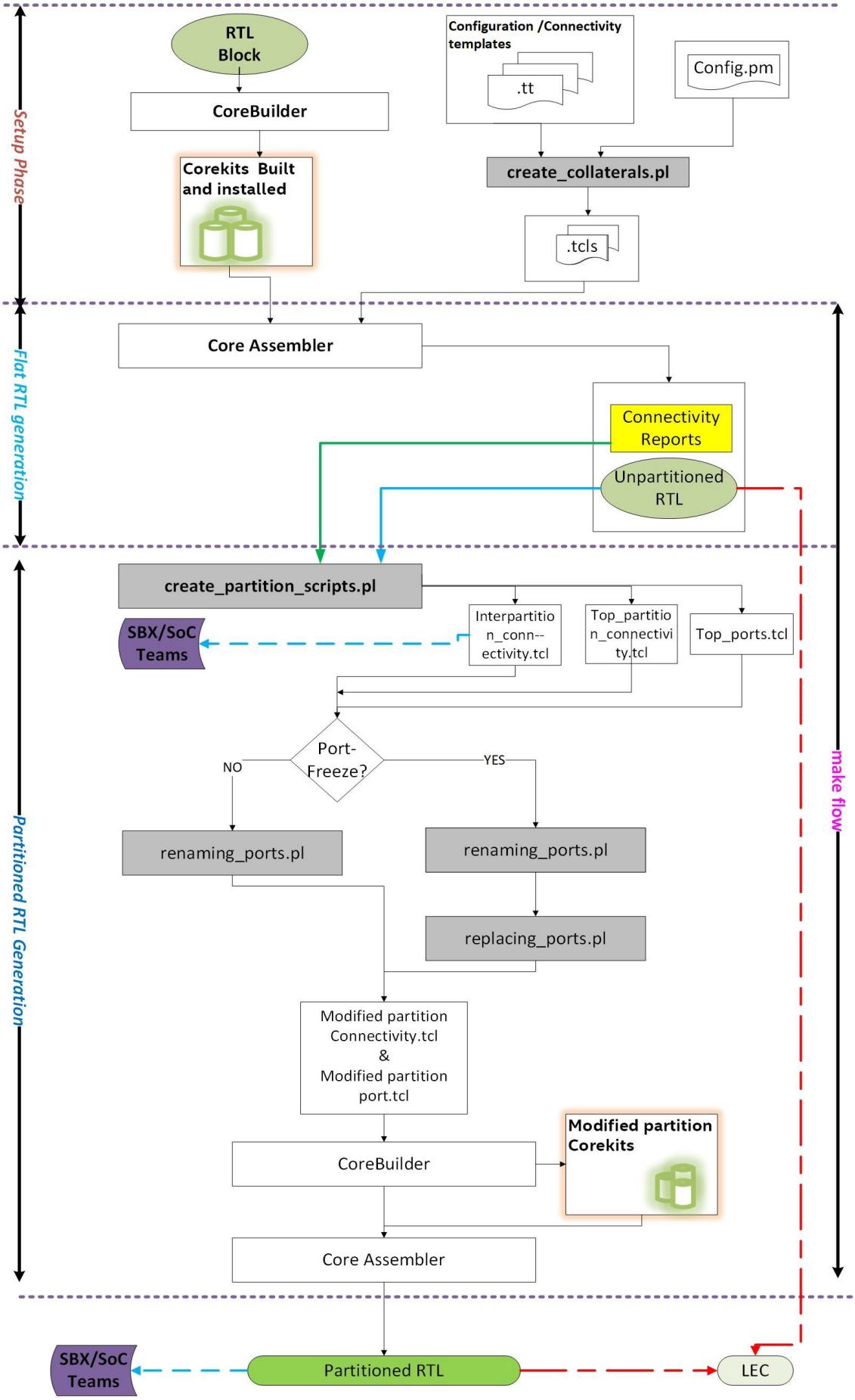

Fig. 3: Different stages of make flow

2) Design Scripts Creation: In this stage, a script called create_collaterals.pl, processes all the template files and generates the TcL scripts that are inputs for the later stages. As shown in the flowchart in Fig 3, the inputs to this stage are the config.pm and the template files.

3) CoreBuilder: This stage invokes the corebuilder shell ,creates and installs the corekits which contain all the design information of the RTL blocks.

B. Bottoms- Up/Flat RTL generation :

The Bottoms-Up/Flat RTL generation is comprised of the following stages:

1) CoreAssembler: This stage invokes the CoreAssembler which uses the Instantiate_components.tcl, create_partition.tcl, ports.tcl, connectivity.tcl and configuration.tcl to generate the flat RTL and the connection information in the form of connectivity reports. It has to be pointed out, at this stage there is no control over the partitioned port names. Having desired ports names can increase the readability of the RTL and helps in maintaining consistency of the port names. A custom script, renaming_ports.pl was designed, which is used in the later stage, for this particular purpose which would allow the user to specify the required name for any port. Also if there is very hard constraint on the ports, unused ports can be utilized using replace_ports.pl.

C. Partitioned RTL generation :

This is final stage of the flow which generates the Partitioned RTL. It is to be noted that the advantages of both Bottoms-top and top-bottom design is being exploited here, making the flow, a hybrid one. The Partitioned RTL generation is comprised of the following stages:

1) Partition TCL scripts Generation: create_partition_scripts.pl takes the outputs of the Flat RTL generation flow, namely the connectivity_reports, flat RTL and generates the inter_partition_connectivity.tcl, tops_partition_connectivity.tcl, create_partition_ports.tcl. The port names generated are of IP-NAME_PARTITION-NAME_SUB-IP_SIGNAL format.

2) Renaming ports: As pointed out in earlier sections, one downside of using the tool based approach is the lack of freedom in choosing the desired port name. This provision is given to the user using a script called renaming_ports.pl, which takes a mapping file as input.

3) Reusing ports: Due to the constraints placed on the changes in the number of ports by SoC to maintain consistency and easier integration time, IP team has to decide on the number of ports at earliest time frame about the port freeze. One good practice which proved to be helpful is to estimate the number of spare ports required and adding them beforehand. These unused ports and the ports left dangling after connectivity changes due to bug fixes or design changes can be reused using the script replacing_ports.pl.

D. CoreBuilder :

Depending on the modified partition ports, partition CoreKits are generated and installed.

E. CoreAssembler :

Using the modified connectivity/partition_port TCLs and the modified partition CoreKits generated in the previous stages, CoreAssembler generates partitioned RTL.

III. RESULTS:

Using the above mentioned automated flow the ARM based Subsystem is successfully delivered to the customer SOC in 3 partitions. The flow is agnostic and flexible enough to handle the movement of Sub-IPs across different partitions. Using this flow, the design scalability and configurability has been increased drastically, so that this subsystem IP can be delivered to other SoCs for any future needs in an efficient way in terms of efforts and the delivery time. The RTL generated by the flow is correct by construction as it is lint and Logical equivalence check (Flat Unpartitioned RTL vs Partitioned RTL) clean. For a SoC request to move a very huge Sub-IP from PAR_A to PAR_B, the updated bug free , partitioned RTL was generated in less than 1 hour.

A brief comparison of Partitioning the IP with and without these changes (Manual) is shown in Table 1 below

TABLE I: Comparison between the make flow and legacy method

Metric	**With automated make flow mentioned in this paper**	**Manual RTL Partitioning**	**Comments**
First-cut Partitioned RTL bring-up time	1 week	5 weeks	**5x** Increase in efficiency
Partitioned RTL generation in the case of movement of Sub-IPS across the partitions	1 hr Independent of the complexity of the module since it is agnostic and robust flow	2 weeks depending on the complexity of Sub-IP	**200x** increase in efficiency
Soc Integration support	Provides the inter _partition_ connectivity.tcl which serves as ad-hoc connectivity input for SOC Integration	Cumbersome and lots of feedback needed to Soc for non-standard port connections	Decrease in the IP integration time at SoC level
RTL Quality	No Syntax and Lint issues possible as it is tool generated. It is totally synthesizable	Can be numerous errors leading to Syntax and lint issues. May not be synthesizable	Quality of the Soft IP is increased drastically

IV. SUMMARY :

A novel approach for partitioning an IP effectively which helped in delivery of a very complex ARM based subsystem in

3 partitions amid of stringent timelines, to SoC is discussed. The flow that can help in increasing the scalability and configurability is thoroughly explained which can be ported to any IP/SS(Subsystem) which uses coretools flow.

V. Future Work :

Future work would include to make the flow more matured and robust. Generating the partition constraints and waivers for the Design check Tools and validation inputs whenever there is an addition, removal and movement of a sub-IP across partitions would definitely make the partitioned subsystem delivery to SoC more agile.

References

[1] Coretools user guide and command reference