Leaping Left: Seamless IP to SoC Hand-off

Swetha Thiagarajan, Rashika Madan, Hiran Morar, Sangeivi Sivagnanasundaram
Intel Corporation
1900 Prairie City Rd
Folsom, CA 95630

Abstract*- IP packaging and qualification are an integral part of IP integration into SoC. Higher turn-around times (TAT) to integrate Graphics IPs into SoC has been a constant challenge in the past. Inefficiencies in package generation, undocumented hacks/workarounds and lack of automation are few of the causes resulting in low quality IP drops. In addition, there weren't any efficient methodologies to check IP design quality against Industry and SoC standards. This required substantial manual effort to generate IP package and thorough testing at IPs before making an SoC delivery. In this paper, we introduce a novel Tools, Flows and Methodology (TFM) agnostic approach to automated IP packaging along with a sophisticated Quality Assurance (QA) as well as a Quality Checker (QC) infrastructure aiming to shift-left identification of integration bugs using a Continuous Integration (CI) system. The methodology is based on robust and improved methods which can generate fully qualified drops to help reduce SoC Integration TAT from 2-3 weeks to less than 3 days per milestone drop as shown in Figure 1. Within a Product Life Cycle (PLC), there has been around one quarter savings and 50% reduction in resourcing through these efforts.

I. Introduction

With each IP maintaining a local copy of packaging tool in their environment to maximize customizations, early packaging solutions lacked any form of standardization. Our initial solution was a diversified utility that completely transformed an IP model to SoC-based model. While this provided extensive customization hooks, there was a lack of centralization and heavy ties to TFM/environment. This led to noticeable shortcomings which included the need for IP and SoC's build systems to be aligned as well as requiring the tools versions between both to be converged. This apparent lack of portability eventually resulted in multiple integration bugs and hot fixes. Next came export_to_soc, which addressed some of these limitations. With the improved package structure, fewer issues were seen at Discrete and Integrated SoCs, resulting in better integration TAT. IPs didn't need to maintain individual copies of the flow anymore because of better centralization. However, tool customization across Soft-IPs and Hard-IPs became expensive. Different approaches to CTECHs, Memories, DFT and Validation collateral delivery are some of the areas where IP's customizations come into picture. This solution also assumed SoC to be using certain TFM which rendered unfeasible when SoC moved to a different TFM in the most recent project. Additionally, the lack of a good quality assurance tool led to several front-end integration issues to go unnoticed and being caught only at SoC. While SoC did have a quality assurance tool which can be used to run several static checks on the generated package by parsing through log files, we weren't able to efficiently utilize it due to IP versus SoC environment dissimilarities and non-applicability of several checks to our Graphics IPs.

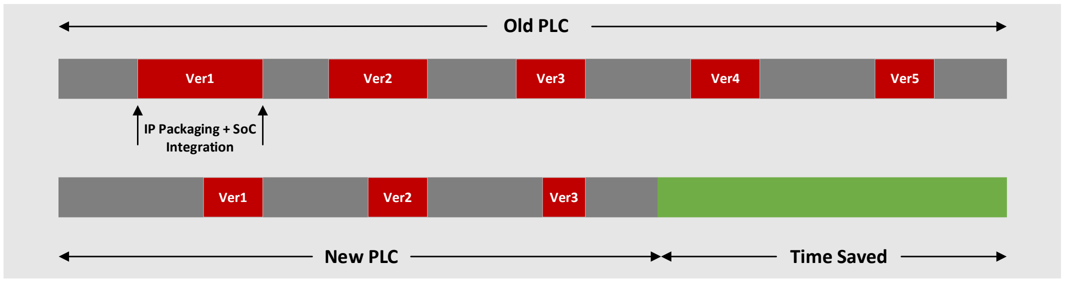


Figure 1. PLC and its dependency on IP to SoC Hand-off

A successful IP delivery is measured by the ease of integration into a SoC along with the Time to Market (TTM). Enter: Export_IP, our TFM-agnostic IP packaging tool that aims to achieve the perfect balance between customization and standardization, without compromising quality. The overall framework is consistent across all Graphics IPs with customization hooks/switches available to help meet unique SoC delivery requirements. Export_IP is environment agnostic, works seamlessly on 2-step and 3-step compile methods, and can be used to deliver to any SoC, irrespective of SoC TFM. We understand that speedy delivery of IP comes with its own limitations. To ensure that the milestone drops meet quality prior to hand-off and is LRM compliant [1], QA is added to ensure that the package is self-contained and compiles standalone on a clean RTL-only and RTL+Val filelist. This qualified IP package then undergoes rigorous checking mechanism through an automated QC tool, Orion, developed internally to support standardized dynamic testing so that it fully meets the IPSOC hand-off exit criteria. Orion architecture and implementation are further discussed in Sections II and III.

II. Architecture

A. Export_IP

While architecting Export_IP, our goal was to develop a robust IP packaging and qualification framework that is TFM-agnostic. This helped in ensuring we don't have to start over each time SoC changed TFMs, as we had to in the past. We also combined the strengths of all our previous solutions, while addressing their shortcomings. As illustrated in Figure 2, Export_IP was developed into three primary stages: create IP data, generate the package, and run standalone QA. To create the IP data, Export_IP leverages built-in APIs from the IP's native build system. This enables us to build the same filelists across IP internal validation + SoC handoff [1]. The second stage takes the data dump, converts them into industry standard filelists that can be plugged in to any downstream compiler. IPs can customize how the collateral gets packaged, depending on SoC requirements. We have also introduced switches to create flattened versions of the package, which can be used for Structural Design (SD) handoff. This allows consistency across all downstream consumers of the package – simulation, emulation, SoC and SD – resulting in fewer bug escapes.

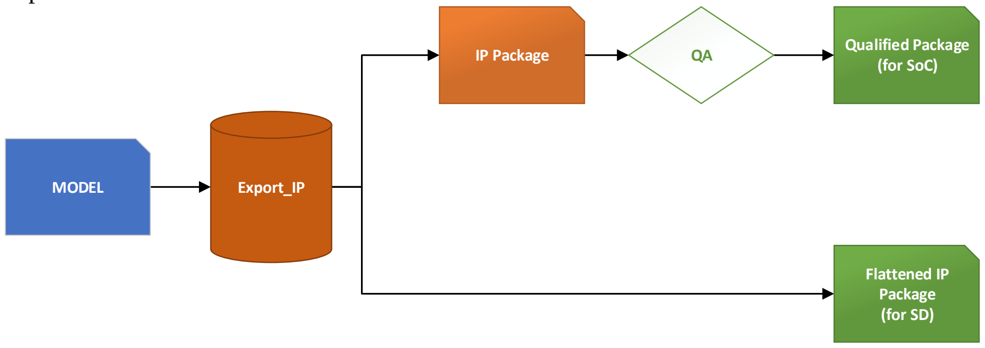


Figure 2. Export_IP Architecture

The final, and perhaps most instrumental stage is having a robust QA system around the generated package. In the past, often issues were only caught at SoC integration, and the turnaround time to get them fixed was several weeks. As a part of Export_IP, we aimed to qualify every filelist that we provided and shifting left identifying integration bugs. VCS, Questa and Spyglass standalone were run, to ensure basic synthesizability of the RTL filelist. VCS sim on the RTL + Val file-list was also implemented to ensure all validation collateral delivered was self-contained. Finally, the resulting package is passed through our automated handoff QC tool, Orion, which guarantees all additional collateral required by SoC (waivers files, configuration files, lots) [1] were present and met the committed PLC milestone requirements as shown in Figure 3. IPs also had the flexibility to easily configure the checks per the same PLC milestone requirements. QC resolves both configurability and environment issues that were faced in the past.

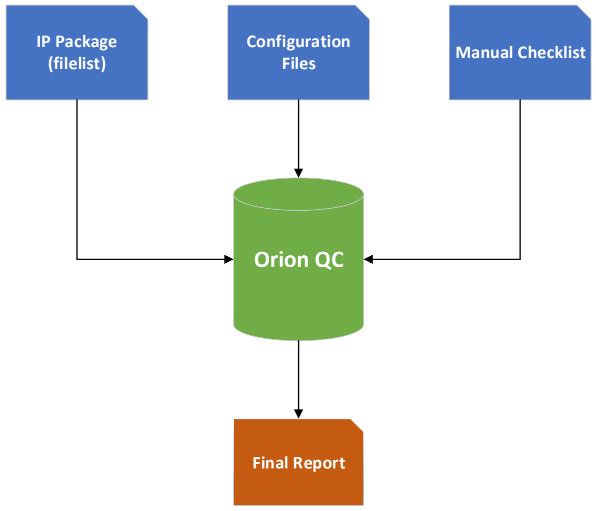
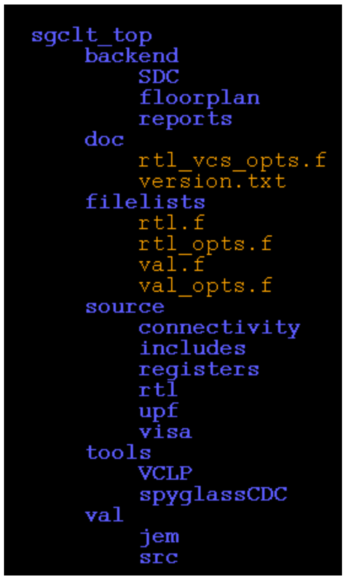


Figure 3. Orion Architecture

III. IMPLEMENTATION

A. Packaging

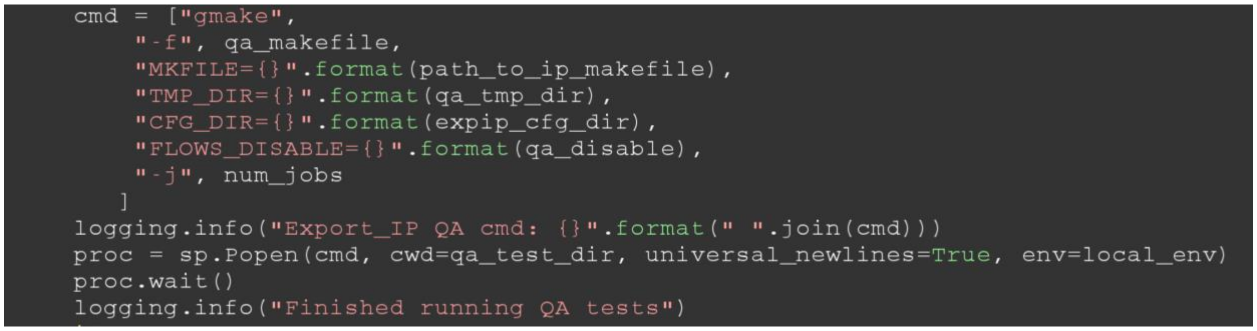
The IP package directory structure is defined by the IP via configuration inputs and is created at the beginning of the flow. Export_IP then uses native build tool APIs to generate a metadata file (libs_data.yaml) which contains all the IP data required to build the package. This data includes source files, Verilog include/lib directories, compile-time options, validation collateral, etc. The APIs used to build this data are the same ones that are used during IP internal model builds, to ensure consistency across IP validation + IP to SOC handoff. Through customization hooks (i.e., callbacks), IPs can define how to transform the metadata into a pre-defined IP package structure. Here, they can also choose to filter certain collateral such as internal BFMs and testbench/coverage files not intended for SoC consumption and/or add additional collateral such as register configurations, checker/trackers, connectivity information and DFT collateral that are needed for SoC validation. The metadata is then translated into industry standard filelists - typically one for RTL (rtl.f) and one for Val (val.f). Figure 4 demonstrates sample directory structure for one of the IPs where tool is customized to add VCLP, upf and spyglass CDC collateral while still maintaining standardized filelists, verilog files and include directories. Since Structural design (SD) TFMs require IP drops to come in a single, flattened directory, we also introduced a method in Export_IP to generate the same content in flat area to help streamline IP2SOC + IP2SD handoffs.



```
sgclt_top
    backend
        SDC
        floorplan
        reports
    doc
        rtl_vcs_opts.f
        version.txt
    filelists
        rtl.f
        rtl_opts.f
        val.f
        val_opts.f
    source
        connectivity
        includes
        registers
        rtl
        upf
        visa
    tools
        VCLP
        spyglassCDC
    val
        jem
        src
```

Figure 4. Packaged IP using export_ip

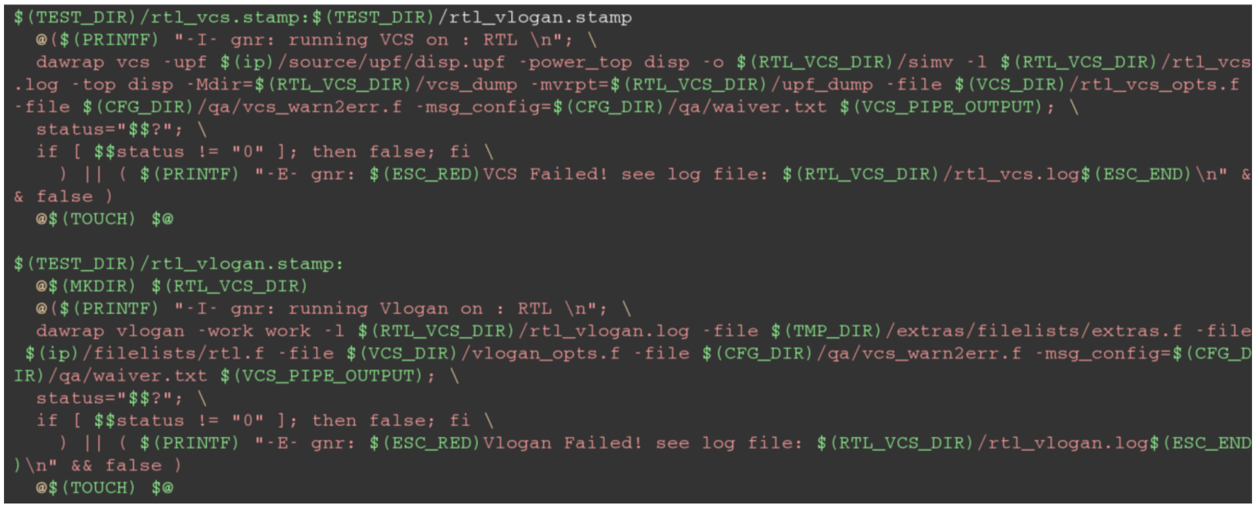
To help ensure correctness of the generated package/filelists, Export_IP comes with a light built-in qualification flow which is maintained through a combination of central and IP specific Makefiles. While central makefile is responsible for overall QA environment and supporting new checks, IP makefile maintains commands specific to IPs to launch these checks. This scalable QA setup allows IPs to effortlessly enable/disable QA tests based on IP readiness and commitment to SoC. Currently, Spyglass DesignRead compilation on the RTL filelist to ensure basic synthesizability and multi-vendor standalone simulation on RTL, RTL+Val filelists to ensure strong LRM compliance [1] are supported. A code snippet provided in Figure 5 depicts how an export_ip QA callback relies on both central and IP makefiles and utilized switches to direct run areas and disable tests.



```
    cmd = ["gmake",
           "-f", qa_makefile,
           "MKFILE={}".format(path_to_ip_makefile),
           "TMP_DIR={}".format(qa_tmp_dir),
           "CFG_DIR={}".format(expip_cfg_dir),
           "FLOWS_DISABLE={}".format(qa_disable),
           "-j", num_jobs
          ]
    logging.info("Export_IP QA cmd: {}".format(" ".join(cmd)))
    proc = sp.Popen(cmd, cwd=qa_test_dir, universal_newlines=True, env=local_env)
    proc.wait()
    logging.info("Finished running QA tests")
```

Figure 5. Code Snippet – Export_IP callback to invoke QA using makefiles

Code snippet in Figure 6 walks us through 3 step VCS compilation to ensure the package is able to compile standalone. Since these commands are IP specific, they are maintained in local makefiles and owned by IP.



```
$(TEST_DIR)/rtl_vcs.stamp:$(TEST_DIR)/rtl_vlogan.stamp
  @($(PRINTF) "-I- gnr: running VCS on : RTL \n"; \
  dawrap vcs -upf $(ip)/source/upf/disp.upf -power_top disp -o $(RTL_VCS_DIR)/simv -l $(RTL_VCS_DIR)/rtl_vcs
.log -top disp -Mdir=$(RTL_VCS_DIR)/vcs_dump -mvrpt=$(RTL_VCS_DIR)/upf_dump -file $(VCS_DIR)/rtl_vcs_opts.f
-file $(CFG_DIR)/qa/vcs_warn2err.f -msg_config=$(CFG_DIR)/qa/waiver.txt $(VCS_PIPE_OUTPUT); \
  status="$$?"; \
  if [ $$status != "0" ]; then false; fi \
    ) || ( $(PRINTF) "-E- gnr: $(ESC_RED)VCS Failed! see log file: $(RTL_VCS_DIR)/rtl_vcs.log$(ESC_END)\n" &
& false )
  @$(TOUCH) $@

$(TEST_DIR)/rtl_vlogan.stamp:
  @$(MKDIR) $(RTL_VCS_DIR)
  @($(PRINTF) "-I- gnr: running Vlogan on : RTL \n"; \
  dawrap vlogan -work work -l $(RTL_VCS_DIR)/rtl_vlogan.log -file $(TMP_DIR)/extras/filelists/extras.f -file
 $(ip)/filelists/rtl.f -file $(VCS_DIR)/vlogan_opts.f -file $(CFG_DIR)/qa/vcs_warn2err.f -msg_config=$(CFG_D
IR)/qa/waiver.txt $(VCS_PIPE_OUTPUT); \
  status="$$?"; \
  if [ $$status != "0" ]; then false; fi \
    ) || ( $(PRINTF) "-E- gnr: $(ESC_RED)Vlogan Failed! see log file: $(RTL_VCS_DIR)/rtl_vlogan.log$(ESC_END
)\n" && false )
  @$(TOUCH) $@
```

Figure 6. Code Snippet – 3 Step VCS to qualify IP Package

B. *Qualification*

Orion has a top-level wrapper that helps IPs toggle the following underlying scripts as shown in Figure 7:

1. Kick off additional checks which are not a part of IP gatekeeper but are required by SoC.
2. File existences check to ensure all files required by SoC (including metadata) are present inclusively. Table I captures the checks and offered by Orion and their intent.
3. Verify the final report of each underlying check to ensure there are no un-waived violations or fatal errors. Each component is written as a module subroutine that can be extended by IP team.
4. Collate all the various logs/reports in the package to create a final aggregated report.
5. Automated email generator to send out release notes and QC results to the IP Integration team.

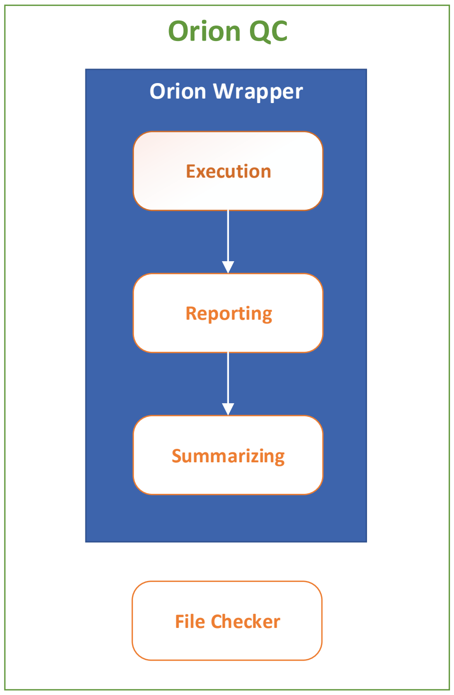
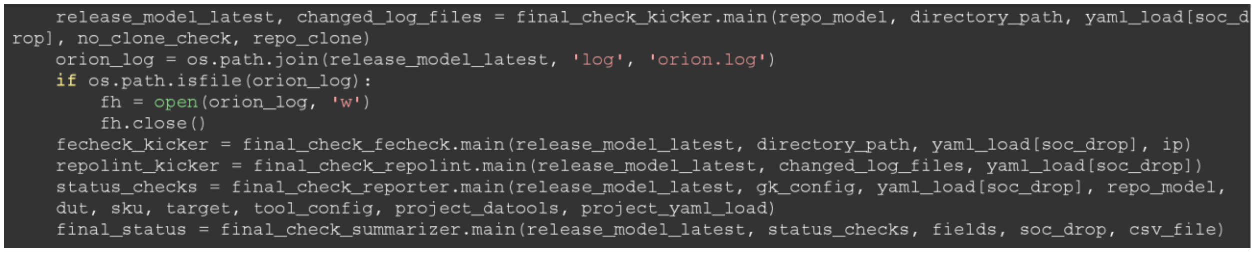


Figure 7. Orion Implementation

TABLE I
ORION QUALITY CHECKER CATEGORIES

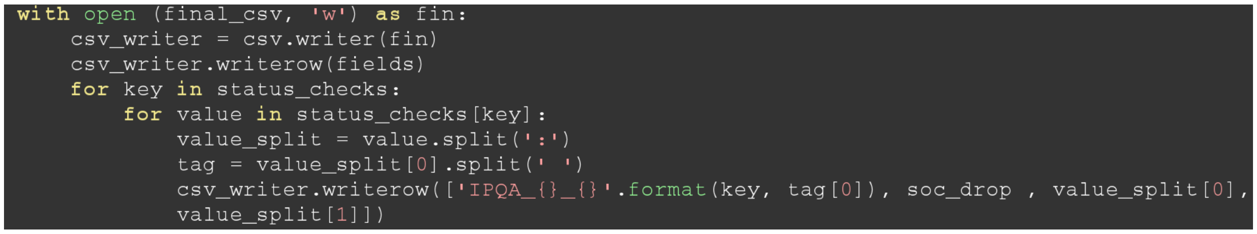
Category	Intent
Front-End	Basic VCS elaboration checks, illegal cells, macro definition, illegal macro usage in the RTL.
Back-End	No latches SoC interface, no undriven pins, CTECH compliance.
SIP compliance	Usage of global macros or vendor-specific macros, LRM compliance.
Emulation	Ensure design is multi-platform compliant (Zebu, Veloce, etc.)
File existence	Ensure any additional/meta files are present in the package.
Manual/custom	Additional custom checks an IP can add based on their unique requirements.

Orion wrapper is a simple script that calls other scripts and functions to generate the data and report the quality of the design as shown in Figure 8. Then the summarizing script will generate the csv file for the final report as shown in Figure 9.



```
    release_model_latest, changed_log_files = final_check_kicker.main(repo_model, directory_path, yaml_load[soc_d
rop], no_clone_check, repo_clone)
    orion_log = os.path.join(release_model_latest, 'log', 'orion.log')
    if os.path.isfile(orion_log):
        fh = open(orion_log, 'w')
        fh.close()
    fecheck_kicker = final_check_fecheck.main(release_model_latest, directory_path, yaml_load[soc_drop], ip)
    repolint_kicker = final_check_repolint.main(release_model_latest, changed_log_files, yaml_load[soc_drop])
    status_checks = final_check_reporter.main(release_model_latest, gk_config, yaml_load[soc_drop], repo_model,
    dut, sku, target, tool_config, project_datools, project_yaml_load)
    final_status = final_check_summarizer.main(release_model_latest, status_checks, fields, soc_drop, csv_file)
```

Figure 8. Code Snippet - Wrapper Function Call



```
with open (final_csv, 'w') as fin:
    csv_writer = csv.writer(fin)
    csv_writer.writerow(fields)
    for key in status_checks:
        for value in status_checks[key]:
            value_split = value.split(':')
            tag = value_split[0].split(' ')
            csv_writer.writerow(['IPQA_{}_{}'.format(key, tag[0]), soc_drop , value_split[0],
            value_split[1]])
```

Figure 9. Code Snippet - Summarizer CSV Generation

C. *Challenges*

Deployment of Export_IP and Orion QA came with its fair share of challenges. We categorized them in three main categories:

1) *Compute efficiency:* Due to the large size of Graphics IPs e.g. 150,000+ files in Graphics Processing Unit (GPU), scalability of the Export_IP flow was crucial. Multiple features have been implemented to enhance the flow's compute efficiency. We added a --use-tmp feature to build and validate the package on tmp disk to limit file I/O on NFS and avoid thrashing. Improvements are tabulated in Table II.

TABLE II
IP PACKAGING TAT IMPROVEMENT WITH TMP DIRECTORY USAGE AND PARALLELIZING QA

IP	Previous Program	Latest Program	% Improvement
Display	58 min	24 min	59%
Media	164 min	117 min	29%
GPU	363 min	280 min	22%

2) *VCS Elaboration dependency:* To keep package generation time as minimal as possible, a --dryrun feature was implemented to smartly build the necessary dependencies while skipping actual VCS compiles when not applicable. Parallelizing various independent QA checks also helped in substantially reducing

packaging TAT. Table III summarizes the TAT improvements observed by eliminating unnecessary compile dependencies whereas Table IV gives overall percentage improvements resulted by consolidating the beforementioned enhancements. It is noteworthy that runtime improvements were limited in GPU IP since elaboration dependency remains for VCS QA to have an updated header file. Decoupling the dependency of the checklist for Orion was also a challenge and this restricted the QC tool to be completely parallelized.

TABLE III
IP PACKAGING TAT IMPROVEMENT WITH --DRYRUN USAGE

IP	Previous Program	Latest Program	% Improvement
Display	86 min	29 min	66%
Media	104 min	60 min	42%

3) *Lack of clean RTL and VAL separation:* Another major challenge faced, was the lack of a clear separation between RTL and VAL collateral at IPs. This prevented us from enabling full spectrum of QA checks (such as Spyglass Designread) in the beginning of the deployment.

TABLE IV
IP PACKAGING TAT IMPROVEMENT WITH CONSOLIDATED EXPORT_IP ENHANCEMENTS

IP	Previous Program	Latest Program	% Improvement
Display	137 min	53 min	61%
Media	268 min	104 min	61%
GPU	490 min	407 min	17%

4) *Streamlining features across IPs*: Having an extensible and configurable architecture gives IPs the flexibility they need, but it doesn't come without cost. One of the resulting challenges is ensuring key features are deployed across all IPs, which becomes difficult if each IP maintains their own custom logic/hooks. Changes made in one IP's hooks won't automatically propagate to other IPs, and these gaps are often times only noticed during manual reviews. In an effort to alleviate this overhead, we identified all the active features which are common across IPs and consolidated them into a central area where IPs simply can import them. By default, all IPs get all features, but knobs were added around each key feature in case it needs to be disabled in any particular IP. In the end, common features would get streamlined and IPs should be left maintaining only the features which are truly custom/unique to their IP.

IV. RESULTS

Due to its robust and configurable architecture, Export_IP was seamlessly adopted by all Intel Graphics IPs, regardless of what compilation strategy the IP was using and what TFM their SoCs were on. Export_IP has been largely perceived within Graphics as a "one size fits all" IP packaging solution. The entire solution took our team 3.5 man-months to develop, test & deploy - 1.5 months to develop the Export_IP framework, 1 month to develop Orion QC, and 1 month of exhaustive testing and integrating the solution into the Display IP.

For the Display IP, we saw 80% faster TPT, 50% headcount reduction from integration team, and 90% fewer SoC integration bugs just from the first three SoC drops post-enablement. For reference, two projects prior, Display delivered more than 30 drops (including hot fixes). In the most recent project with Export_IP enabled, this reduced to only 6 milestone drops understating the high quality and ease of integration to SoC. In addition to the higher quality drops, Display was able to use the same infrastructure to deliver to different SoCs with varying requirements. In prior programs, SoC integrations of the Display IP would typically take 2-3 weeks. With Export_IP and Orion QC, this was reduced to just 3 days - 1 day to build + deliver the package, and 2 days to successfully integrate at SoC. This also allowed the Graphics IP to auto-integrate into their SoC Die through a Continuous Integration framework. Tables V and VI provide a comparison in IP Packaging TAT and Front-End SoC Integration TAT between previous program and Export-IP implementation for GPU, Media and Display IPs.

With Orion, QC was completed in ~3 hours as opposed to 1 week in prior programs. The number of unnecessary revisions due to poor quality drops was reduced to nearly zero, and the integration bugs filed by SoC was under 5 throughout the entire program. This extensible framework was seamlessly adopted by additional IPs for the next generation programs.

TABLE V
IP PACKAGING TAT PRIOR TO SOC HAND-OFF

IP	Previous Program	Latest Program (with Export_IP)
Display	3-4 Days	**1-2 hours**
Media	1 week	**1-2 hours**
GPU	1 week	**2-3 hours**

TABLE VI
FRONT-END SOC INTEGRATION TAT

IP	Previous Program	Latest Program (with Export_IP)
Display	2-3 weeks	**1-2 Days**
Media	3-4 weeks	**1-2 Days**
GPU	3-4 weeks	**2 Days**

Furthermore, adaptation of Export_IP QA framework by other teams outside Intel Graphics helped in improving the quality of their IP drops by eliminating any issues such as incorrect filelists.

V. SUMMARY

Export_IP and Orion QC have both delivered results immediately upon deployment. From an IP delivery perspective, both Front-end Integration TAT as well as integration bugs significantly decreased. This results in a major left-shift as it lessens the burden on IP/SoC integration and allows IPs to own the responsibility for delivering quality milestone drops. This approach resolved past issues on TFM dependencies, environment, and configurability, The QA approach has even been utilized across other IPs outside Graphics. For future projects, we have a road map identified to enhance the features to continuously surpass established industry standards. This includes a reference test bench delivery for boot sequences, addition of a Mock SoC to ensure the transformed IP package is comprehensive and meets PLC standards. For Orion QC, support for an online reporting dashboard and indicators are being implemented. Advanced configurability is also being added to allow scaling across IPs. As we work through the challenges of the IPs complexity, improving the performance to reduce TAT and identifying areas for convergence across IPs continue to have a strong hold on our roadmap.

ACKNOWLEDGMENT

To Nikhil Krishna Gopalakrishna, for driving an Intel-wide initiative to streamline IP to SoC handoff requirements and methodologies, and for providing valuable feedback as we architected the “Export_IP” solution.

To Siva Reddybathula, who wrote the first implementation of “Export_IP”, the first TFM-agnostic IP to SoC handoff tool in the Graphics RTL environment.

To several members of the Graphics Front-End TFM team, in particular Aditi Nigam, Narasimhan Iyengar, Akshat Saxena and Angela Hantelmann, who helped define and implement many of the underlying SIP compliance and QA checks.

REFERENCES

[1] IEEE Standard for SystemVerilog--Unified Hardware Design, Specification, and Verification Language," in *IEEE Std 1800-2017 (Revision of IEEE Std 1800-2012)*, 22 Feb. 2018