Advanced Compile Features

Outline

Parallel Netlist (PNL) for IXCOM flow

Parallel Invocation of HDL-ICE Analyzer

Parallel Synthesis

Parallel Partition Compiler (PPC)

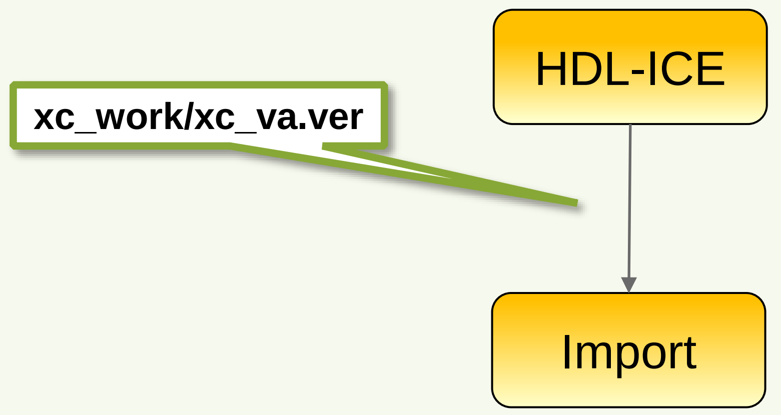
Modular Compilation Flow (MC)

Parallel Netlist

- By default, HDL-ICE only generates a single netlist file for the entire design
- Parallel Netlist was introduced in VXE 19.10
- Compile with -parallelNL switch to the ixcom command:
  - ixcom -parallelNL <other required ixcom options>
- IXCOM then generates multiple netlist files during HDL-ICE
  - Creates a netlist file for each module and a skeleton file for design hierarchy
- When Parallel netlist is detected, designImport is launched with multiple processes in parallel, and each process works on a subset of the netlist files

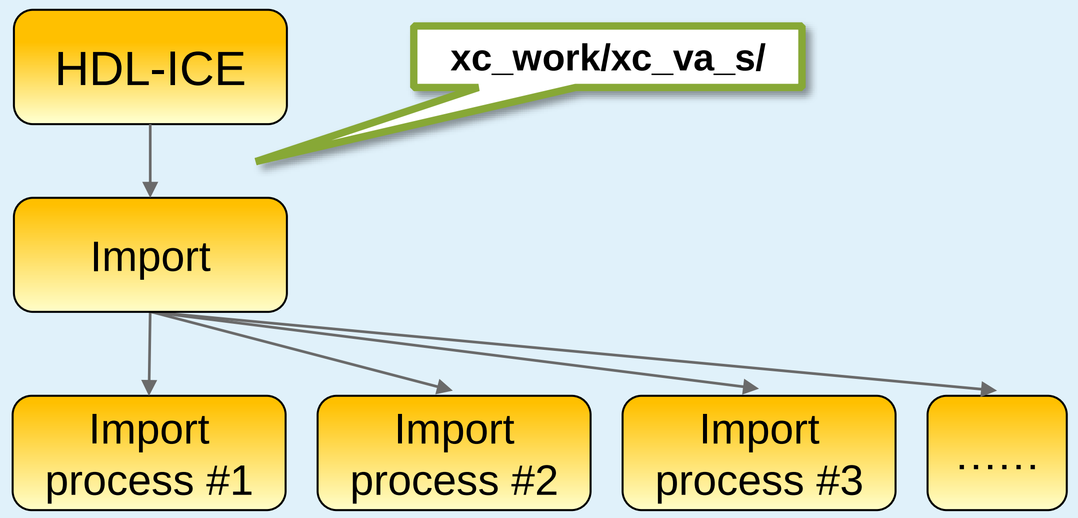
Without Parallel NL v.s. with Parallel NL (PNL)

Without PNL



– Single netlist for the entire design
– Single netlist file is processed sequentially
– Performance bottleneck when netlist grows bigger

With PNL



– xc_va.ver is replaced with a xc_va_s directory which contains:

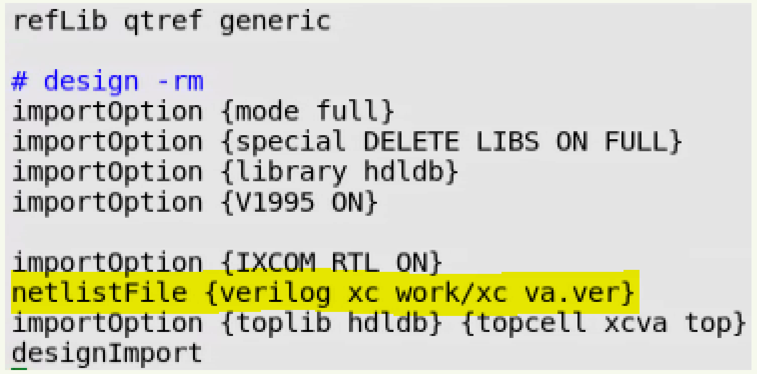
- Netlist files for all design modules: 1.v, 2.v, 3.v, etc.
- A design skeleton file

– Multiple designImport processes are launched and each process works on a subset of the netlist files

Without Parallel NL v.s. with Parallel NL (PNL)

Without PNL

xc_work/qel/ixcomImport.qel contents

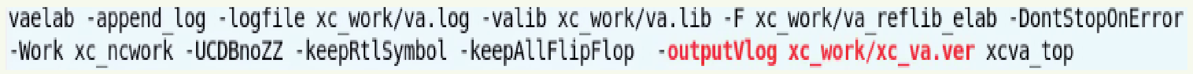


```
refLib qtref generic

# design -rm
importOption {mode full}
importOption {special DELETE_LIBS_ON_FULL}
importOption {library hdldb}
importOption {V1995 ON}

importOption {IXCOM_RTL_ON}
netlistFile {verilog xc_work/xc_va.ver}
importOption {toplib hdldb} {topcell xcva_top}
designImport
```

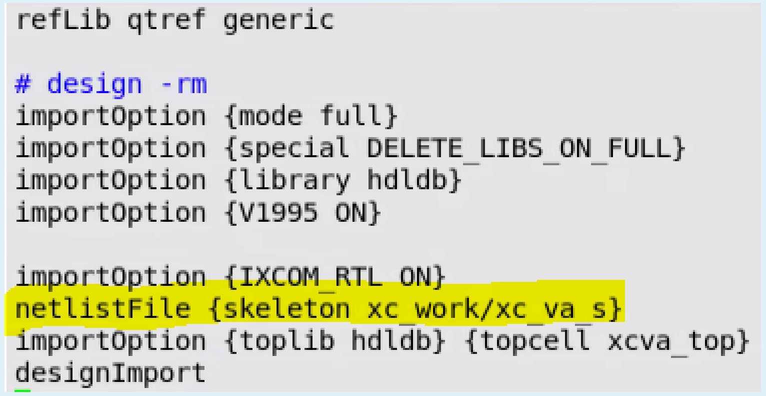
Vaelab command in ixcom.log



```
vaelab -append_log -logfile xc_work/va.log -valib xc_work/va.lib -F xc_work/va_reflib_elab -DontStopOnError
-Work xc_ncwork -UCDBnoZZ -keepRtlSymbol -keepAllFlipFlop  -outputVlog xc_work/xc_va.ver xcva_top
```

With PNL

xc_work/qel/ixcomImport.qel contents

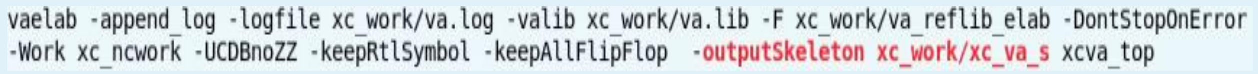


```
refLib qtref generic

# design -rm
importOption {mode full}
importOption {special DELETE_LIBS_ON_FULL}
importOption {library hdldb}
importOption {V1995 ON}

importOption {IXCOM_RTL_ON}
netlistFile {skeleton xc_work/xc_va_s}
importOption {toplib hdldb} {topcell xcva_top}
designImport
```

Vaelab command in ixcom.log

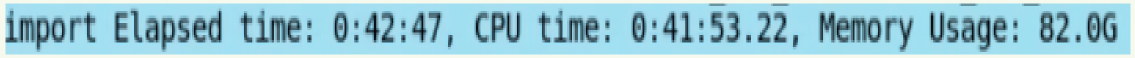


```
vaelab -append_log -logfile xc_work/va.log -valib xc_work/va.lib -F xc_work/va_reflib_elab -DontStopOnError
-Work xc_ncwork -UCDBnoZZ -keepRtlSymbol -keepAllFlipFlop  -outputSkeleton xc_work/xc_va_s xcva_top
```

Without Parallel NL v.s. with Parallel NL (PNL)

Without PNL

xe.msg snippet

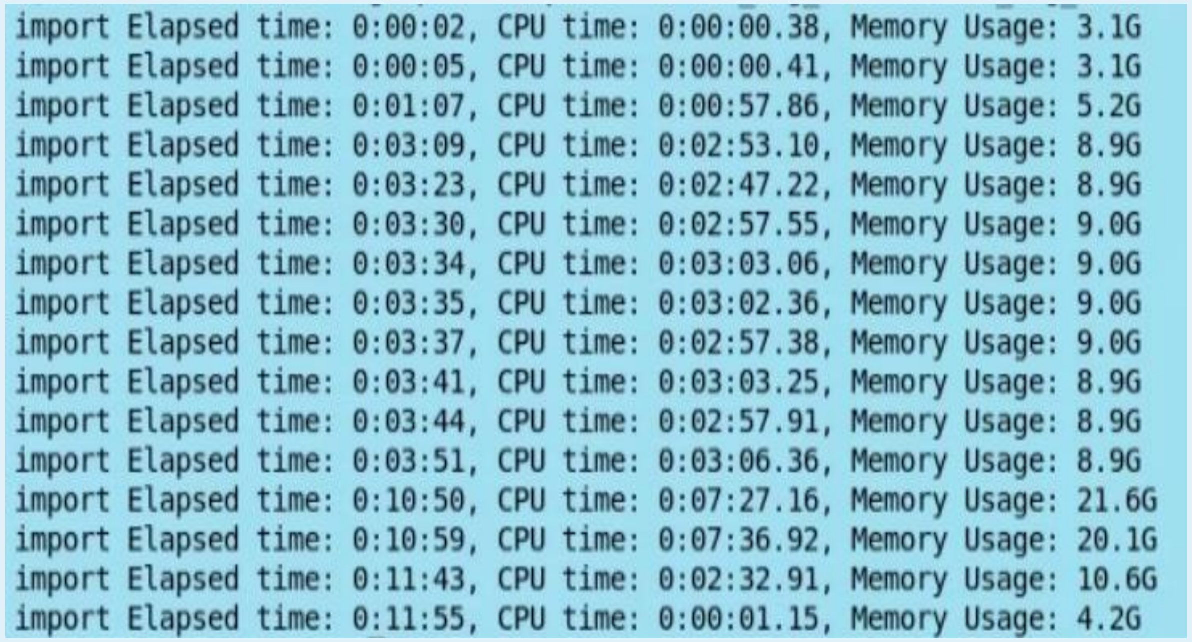


```
import Elapsed time: 0:42:47, CPU time: 0:41:53.22, Memory Usage: 82.0G
```

– DesignImport launches a single process

With PNL

xe.msg snippet

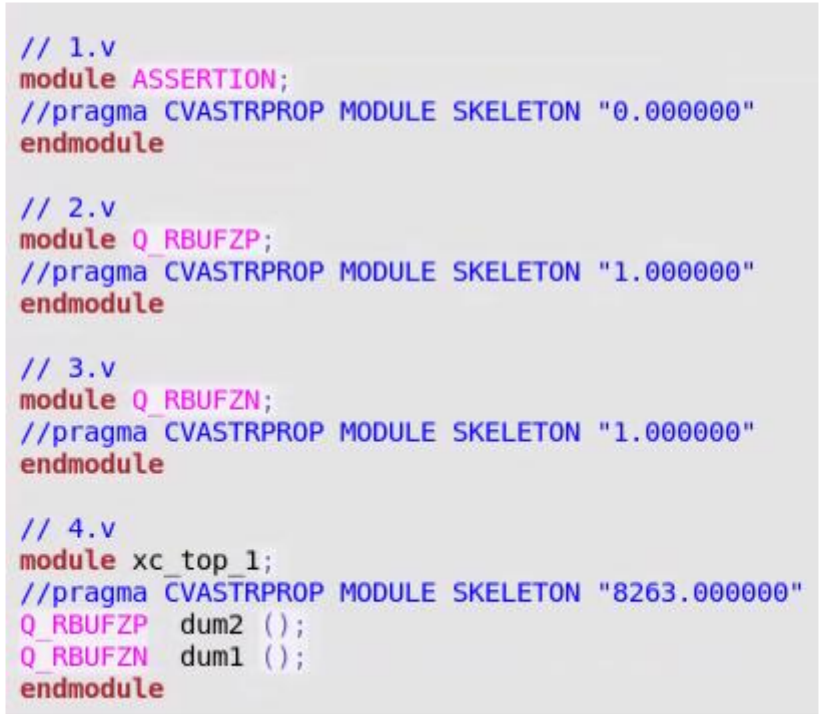


```
import Elapsed time: 0:00:02, CPU time: 0:00:00.38, Memory Usage: 3.1G
import Elapsed time: 0:00:05, CPU time: 0:00:00.41, Memory Usage: 3.1G
import Elapsed time: 0:01:07, CPU time: 0:00:57.86, Memory Usage: 5.2G
import Elapsed time: 0:03:09, CPU time: 0:02:53.10, Memory Usage: 8.9G
import Elapsed time: 0:03:23, CPU time: 0:02:47.22, Memory Usage: 8.9G
import Elapsed time: 0:03:30, CPU time: 0:02:57.55, Memory Usage: 9.0G
import Elapsed time: 0:03:34, CPU time: 0:03:03.06, Memory Usage: 9.0G
import Elapsed time: 0:03:35, CPU time: 0:03:02.36, Memory Usage: 9.0G
import Elapsed time: 0:03:37, CPU time: 0:02:57.38, Memory Usage: 9.0G
import Elapsed time: 0:03:41, CPU time: 0:03:03.25, Memory Usage: 8.9G
import Elapsed time: 0:03:44, CPU time: 0:02:57.91, Memory Usage: 8.9G
import Elapsed time: 0:03:51, CPU time: 0:03:06.36, Memory Usage: 8.9G
import Elapsed time: 0:10:50, CPU time: 0:07:27.16, Memory Usage: 21.6G
import Elapsed time: 0:10:59, CPU time: 0:07:36.92, Memory Usage: 20.1G
import Elapsed time: 0:11:43, CPU time: 0:02:32.91, Memory Usage: 10.6G
import Elapsed time: 0:11:55, CPU time: 0:00:01.15, Memory Usage: 4.2G
```

– DesignImport launches multiple processes in parallel
– Total elapsed time is equal to the largest elapsed time

Parallel Netlist: Module Skeleton File

- The module skeleton file consists of:
  - – Module NL definition without design contents (ports, nets, Q_cells)
  - – Module instantiation with module name and instance name only



```
// 1.v
module ASSERTION;
//pragma CVASTRPROP MODULE SKELETON "0.000000"
endmodule

// 2.v
module Q_RBUFZP;
//pragma CVASTRPROP MODULE SKELETON "1.000000"
endmodule

// 3.v
module Q_RBUFZN;
//pragma CVASTRPROP MODULE SKELETON "1.000000"
endmodule

// 4.v
module xc_top_1;
//pragma CVASTRPROP MODULE SKELETON "8263.000000"
Q_RBUFZP  dum2 ();
Q_RBUFZN  dum1 ();
endmodule
```

Parallel Netlist: Performance

- While using Parallel NL feature, designImport elapsed time is improved by 2x to 3x

#	IXCOM based design	With/Without parallel NL (PNL)	Preoptimized gate count (million gates)	Overall IXCOM compile elapsed time (minutes)	designImport elapsed time (minutes)
1	18-board design	Without PNL	674	64	19
		With PNL		57	9
2	36-board design	Without PNL	1422	142	40
		With PNL		116	12

Parallel Netlist: Guidelines & Limitations in VXE19.10

- Parallel NL doesn’t support below use models:
  - Incremental compilation. User has to do a clean recompile every time
  - Encrypted designs
  - LEC flow
  - External netlist flow (ixcom -externalNetlists)

- VXE19.10 Parallel NL feature supports IXCOM RTL flow only

Outline

Parallel Netlist (PNL) for IXCOM flow

Parallel Invocation of HDL-ICE Analyzer

Parallel Synthesis

Parallel Partition Compiler (PPC)

Modular Compilation Flow (MC)

Parallel Invocation of HDL-ICE Analyzer: Introduction

- The HDL-ICE analysis sessions might take significant time for designs that have multiple libraries. Run these sessions in parallel can optimize the compilation time.
- IXCOM generates the vavlog and vavhdl commands for each library and merges all the sessions of one library in a single session.
  - Use IXCOM option -parallelHdliceAnalyze [<nJobs>]
- When -parallelHdliceAnalyze [<nJobs>] is used, IXCOM automatically invokes the vavlog commands in parallel using the following command:
  - make -j <nJobs> -f xc_work/xc_make vavlog_parallel
  - <nJobs> is the number of vavlog sessions required by user and is optional.
    - If it is not specified, IXCOM compares the number of the available cores on the machine and the number of vavlog sessions that are required, and sets the lesser number as <nJobs>.
    - If it is specified, but its value exceeds the number of available cores on the machine, the number of available cores on machine is set as <nJobs>.

Parallel Invocation of HDL-ICE Analyzer: Example

- Upon executing -parallelHdliceAnalyze, IXCOM generates a target rule, vavlog_parallel, in xc_work/xc_make Makefile. For example:

```
vavlog_parallel: vavlog_1 vavlog_2 vavlog_3 vavlog_4 vavlog_5
```

- It also creates a target rule for each library in xc_work/xc_make as follows:

```
vavlog_1:
vavlog $(VAVLOG_OPTIONS) -append_log $(VALIB) +ixcom -Work IXCOM_TEMP_LIBRARY \
-File xc_work/v/vavlog-IXCOM_TEMP_LIBRARY-verilog.f
vavlog_2:
vavlog $(VAVLOG_OPTIONS) -append_log $(VALIB) +ixcom -Work lib1 \
-File xc_work/v/vavlog-lib1-verilog.f
vavlog_3:
vavlog $(VAVLOG_OPTIONS) -append_log $(VALIB) +ixcom -Work lib2 \
-File xc_work/v/vavlog-lib2-verilog.f
vavlog_4:
vavlog $(VAVLOG_OPTIONS) -append_log $(VALIB) +ixcom -Work lib3 \
-File xc_work/v/vavlog-lib3-verilog.f
vavlog_5:
vavlog $(VAVLOG_OPTIONS) -append_log $(VALIB) +ixcom -Work lib4 \
-File xc_work/v/vavlog-lib4-verilog.f
```

Parallel Invocation of HDL-ICE Analyzer: Example (Con'd)

- Then, IXCOM invokes HDL-ICE Compiler as follows:

```
make -f xc_work/xc_make vlog_serial
make -f xc_work/xc_make vlog_parallel -j 5
make -f xc_work/xc_make vaelab_only
```

- In the ixcom.log file, an INFO message is generated to indicate the number of job slots used for the parallel invocation:

```
Info!      Invoking hdlice analysis steps in parallel using (5) job slots.
```

Outline

Parallel Netlist (PNL) for IXCOM flow

Parallel Invocation of HDL-ICE Analyzer

Parallel Synthesis

Parallel Partition Compiler (PPC)

Modular Compilation Flow (MC)

Parallel Synthesis: Introduction

- Parallel synthesis is useful for synthesizing VHDL or Verilog RTL designs containing tens of million gates.
  - Enables distribution of synthesis into tasks at module level. The tasks are dynamically allocated to balance the load across multiple workstations or processes.
  - Helps to increase the speed during design synthesis.

- Ways to enable parallel synthesis
  - Using parallel.host file
  - Using multiple CPUs of the same workstation
  - Using Load Sharing Facility (LSF) queue
  - Using Network Computer (NC)

Parallel Synthesis: Using parallel.host File

- Hosts and job numbers need to be specified in parallel.host file:

```
HostA -jobs=20
HostB -jobs=30
```

  - 20 HDL-ICE processes are specified to start on HostA and 30 to start on HostB
  - The maximum number of total parallel HDL-ICE processes on all hosts is 248
  - File parallel.host should be located in design directory
  - The workstations can be of mixed platforms
- Execute any of the following commands to start parallelsynthesis using the parallel.host file:
  - vaelab -parallel <top_module>
  - hdlSynthesize -parallel <top_module>
  - vaelab -parallelHostSSH <top_module>
  - hdlSynthesize -parallelHostSSH <top_module>
    - With -parallelHostSSH, HDL-ICE uses SSH to distribute synthesis over hosts listed in parallel.host.

Parallel Synthesis: Using Multiple CPUs of the Same Workstation

- If a workstation has multiple CPUs, tasks can be distributed to run in parallel on those CPUs.

- To distribute a definite number of tasks on the current workstation, use either of the following two commands:
  - vaelab -parallel <number> <top_module>
  - hdlSynthesize -parallel<number> <top_module>
    - The maximum <number> of parallel HDL-ICE Compiler processes on the current workstation is 248.

Parallel Synthesis: Using Load Sharing Facility (LSF) Queue

- Synthesis processes can also be submitted to an LSF queue using any of the following commands:
  - vaelab -parallelLSFSameHost <number> <top_module>
  - hdlSynthesize -parallelLSFSameHost<number> <top_module>
  - vaelab -parallelLSF <number> <top_module>
  - hdlSynthesize -parallelLSF<number> <top_module>
    - <number> is a required argument that specify the number of jobs to submit to the LSF queue. The maximum <number> is 248.
    - Use -parallelLSFSameHost option to assign one workstation for HDL-ICE Compiler and avoid network latency between multi-workstations during parallel synthesis.

Parallel Synthesis: Using Load Sharing Facility (LSF) Queue
Using Multiple bsub Commands

- Starting HDLICE20.12, it supports the distribution of synthesis process over multiple hosts in the LSF network using multiple bsub commands.
  - One bsub command starts one slave process.
- The usage is as follows:
  - vaelab -parallelLSFMultiBsub <number>
  - hdlSynthesize -parallelLSFMultiBsub<number>
    - The maximum <number> is 500.
- Options to the bsub executable can be specified in a file called LSF_options. The compiler does not check for syntax of these options.

Parallel Synthesis: Using Network Computer (NC)

- If NC is used instead of LSF, parallel synthesis processes can also be submitted to NC using either of the following two commands:
  - vaelab -parallelNCSameHost <number> <top_module>
  - hdlSynthesize -parallelNCSameHost<number> <top_module>
    - Use -parallelNCSameHost option to assign one workstation to run parallel synthesis.
    - The maximum <number> is 248.

Parallel Synthesis: Using Network Computer (NC)
Using Multiple nc run Commands

- Starting HDLICE20.12, it supports the distribution of synthesis process over multiple hosts in the NC network using multiple nc run commands.
  - One nc run starts one slave process.
- The usage is as follows:
  - vaelab -parallelMultiNC <number>
  - hdlSynthesize -parallelMultiNC<number>
    - The maximum <number> is 500.
- If a file named NC_options exist under the current working directory, the string will be appended to the NC command when a job is submitted. All the NC options must be defined in one line. The compiler does not check for syntax of these options.

Outline

Parallel Netlist (PNL) for IXCOM flow

Parallel Invocation of HDL-ICE Analyzer

Parallel Synthesis

Parallel Partition Compiler (PPC)

Modular Compilation Flow (MC)

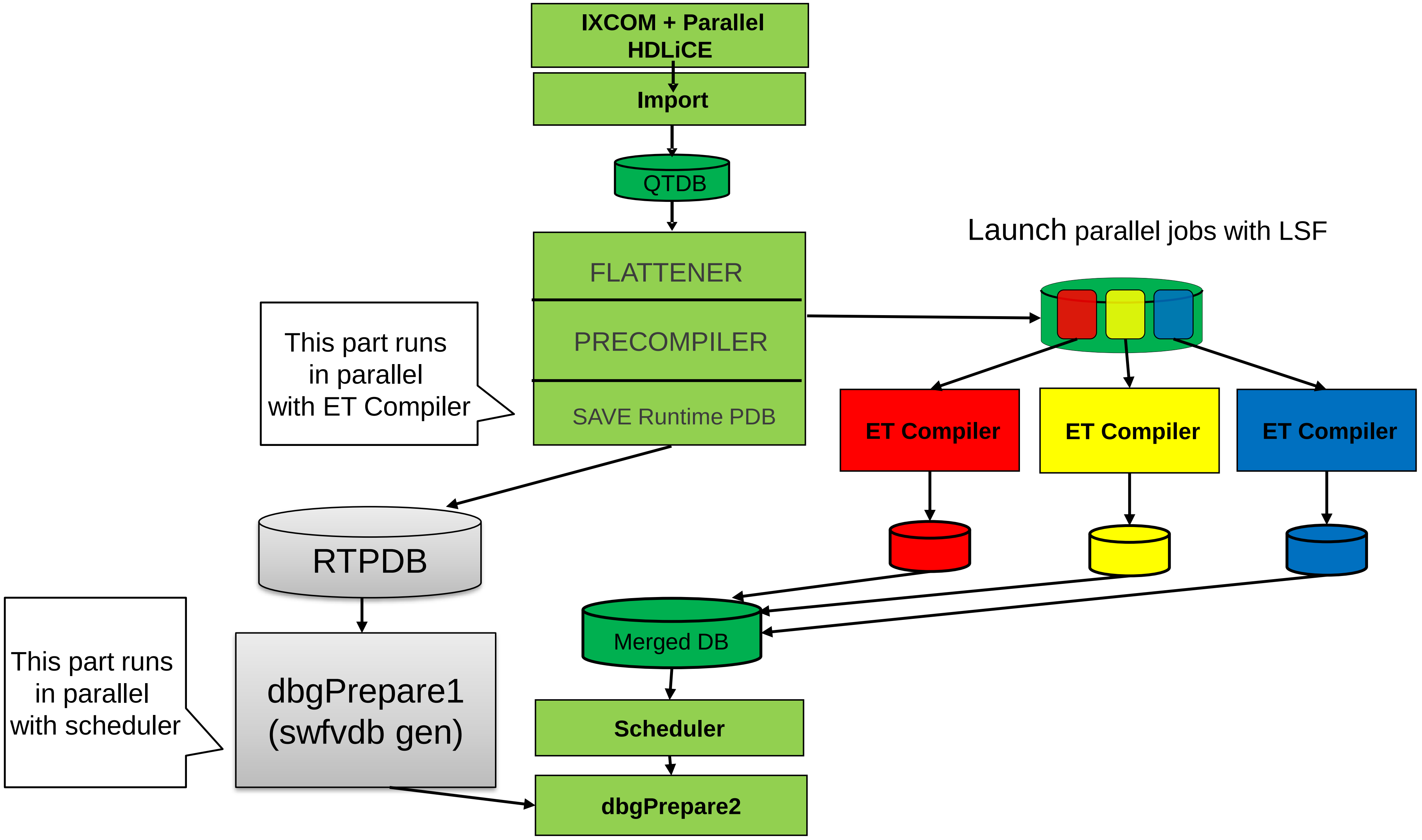
Parallel Partition Compiler (PPC)

- By default, et3compile compiles the entire design sequentially
- VXE 181/185 introduced a new feature called parallel partition compiler (PPC)
- Motivations for PPC technology
  - Reduce compile time by focusing on the longest compile stage: et3compile. The et3compile time for larger designs is linear without the parallel process
- With PPC, the design is split into separate partitions at the end of precompile
- Most of et3compile's work is done on the partitions in parallel. Results are then merged and the design is scheduled.

PPC Requirements

- Requires installation of workload management platform
  - Currently support LSF and NC
  - Support for others done with wrappers
    - PPC-invoked LSF commands then execute wrappers which call job scheduler commands
- Workstation memory requirements based on design size
  - 1GB memory per 1 million gates
  - Require multiple smaller memory machines for partition compile with LSF
  - PPC does not reduce host memory footprint compared to without PPC
- Sufficient disk space

PPC Compilation Flow



PPC: What It Is Doing

- Parallel Partition Compiler parallelizes the most time consuming section of the compile process
  - Design needs to be sliced into multi-board sized partitions
  - The partitions should be roughly the same size to improve the compile time
    - Variation in partitions will result in different compile time, affect overall compile times
  - Partitions are assigned boards and domains by user or by compiler
  - After partitions defined and domains assigned, normal compile with compile or compileFind command
  - Compile will spawn jobs for partitions automatically using LSF / NC
  - When spawned jobs finish, performs final steps including scheduling

How to Enable PPC: Partitioning the Design (Step 1)

- Step 1: partition the design by one of the three ways
  - Manual partitioning (VXE186 or later)
    - Original use model
  - Automatic partitioning (VXE186 or later)
    - XEL user data command autoPart
  - Advanced automatic partitioning (VXE19.10 or later)
    - Second generation automatic partitioning
    - Original autoPart is still supported but no new development in VXE19.10
    - compilerOption -add {advancedPart ON}

How to Enable PPC: Manual Partitioning

- First, use partitionGroup command to assign an instance to a selected partition:
  - partitionGroup -add {<instance>[.][<prer>][*]<partition>}
  - <instance> is either a wildcard character (*) or a valid hierarchical instance name. <partition> is one of the partition identifiers part1- part65499.
  - Wildcard used in <instance> means everything not explicitly assigned
  - All instances not covered by specification are placed in lowest number partition

```
partitionGroup -add {dut.a.b part1}
partitionGroup -add {dut.a.b.r* part2}
partitionGroup -add {dut.a.c.* part3}
partitionGroup -add {* part4}
```

- Secondly, use partitionAssign command to control the emulator configuration (boards and domains) assigned to a given partition.
  - partitionAssign -add {<partition> <resource>}

```
partitionAssign -add {part1 0-1}
partitionAssign -add {part2 2-3}
partitionAssign -add {part3 4}
partitionAssign -add {part4 5}
```

How to Enable PPC: autoPart

- XEL command autoPart automatically partitions and assigns resources for PPC
  - Partitioning is done early in precompile stage
  - autoPart overwrites any existing partitionGroup or partitionAssign command
  - It identifies suitable number of partitions of the design based on the resources and design status, and tries to group them into 3 or 4 board partitions
  - Solution generated is written into partitionGroup.qel, partitionAssign.qel, and tmp directory

- The generated files can be modified and used in future compile
  - autoPart generates an initial solution that user can modify with
  - User with design knowledge is key, understanding interactions of design
    - Move hierarchy in groupings around to improve IO
    - Modify the assignments by add / subtract domains to help utilization
    - Modify the assignments by altering the location of the domains on the rack

How to Enable PPC: Advanced Automatic Partitioning

- In VXE 19.10 or later, advanced automatic partitioning can be turned on by
  - compilerOption -add {advancedPart ON | <N>}
  - ON option enables the compiler to create an appropriate number of partitions based on the boards/domains available
  - N is the number of partitions that can be specified to overwrite compiler’s decision
  - Recommend Use Model:
    - *Complete compile first, then set partition number if partitions have high utilization*
    - Fewer, larger partitions are better to help reduce partitions utilization

How to Enable PPC: Advanced Automatic Partitioning

- advancedPart is the second generation automatic partitioning
  - Original autoPart is still supported but no new development
  - Unlike autoPart, partitioning is done after precompile
  - Netlist contains all precompiler instrumentation logic
    - Works better with 1x and other features where the compiler instrumentation is done
  - Generates better fCLK and better design utilization
  - Partitions are generated by gate count
    - No user intervention in partition process
    - Unlike autoPart, partitionGroup.qel is not generated
  - Partition placement on rack resources done by largest partitions first
  - Only creates a partitionAssign file in tmp
    - Can be used in compile -etOnly to move existing partitions
  - The advancedPart option can only be respected in designs that require 32 or more domains
    - Smaller-than-32-domain designs will be scheduled into 1 partition

How to Enable PPC: Enable Parallel Partition Compiler (Step 2)

- Step 2: enable parallel partition compiler
  - compilerOption -add {distributedMultiCore ON | OFF | <path>}
  - If this compiler option is turned ON, compilation runs in PPC mode, and each partition is compiled in its own directory.
  - The directories are created in PARTITIONS subdirectory within the design directory. You can also specify a <path> where you want to create the directories for the partitions.

How to Enable PPC: Optimization (Step 3)

- Step 3 (optional): Determines what optimization are performed after partition databases are merged
  - compilerOption -add {PostMergeOpt <level> }
  - Can increase compile time
  - <value> can be:
    - 0 – no post merge optimizations
    - 1 – netlist optimization only
    - 2 – critical path optimization only
    - 3 – both netlist and critical path optimization
  - Netlist optimizations can reduce design size
  - Critical path optimization can help reduce step count by moving logic across domain boundaries

How to Enable PPC: Compile (Step 4)

- Step 4: Compile as usual, using compile or compileFind command
  - Compile will spawn jobs for partitions automatically using LSF
  - When spawned jobs finish, perform final steps including scheduling

- LSF options can be with LSF_options or with qel command
  - LSF Option in qel file will be used only for PPC jobs

```
set PPC(LSFOptionsForPartitions) {-q linux -R
"rusage[mem=100000] select[mem>=200000 && (OSREL=EE60
|| OSREL=EE70) ]" -M 200000}
```

How to Enable PPC: Use Models

Manual partitioning example

```
emulatorConfiguration -add ……
partitionGroup -add {dut.a.b part1}
partitionGroup -add {dut.a.b.r* part2}
partitionGroup -add {dut.a.c.* part3}
partitionGroup -add {* part4}
partitionAssign -add {part1 0-1}
partitionAssign -add {part2 2-3}
partitionAssign -add {part3 4}
partitionAssign -add {part4 5}
compilerOption -add {PostMergeOpt 3}  #optional
compilerOption -add {distributedMulticore ON}
compileFind ……
```

autoPart example

```
emulatorConfiguration -add ……
autoPart
compilerOption -add {distributedMulticore ON}
compileFind ……
```

advancedPart example

```
emulatorConfiguration -add ……
compilerOption -add {advancedPart ON}
compilerOption -add {distributedMulticore ON}
compileFind ……
```

cādence®

PPC: Successful Compile

- All stages of a compile must complete successfully
- Things to look for in xe.msg file
  - Did the compile successfully schedule and is there a final speed number reported?

```
INFO (db2util-1067): This design is scheduled in 3728 steps.
INFO (qt2dadb-1086): Maximum emulator operating speed is 138 kHz.
```

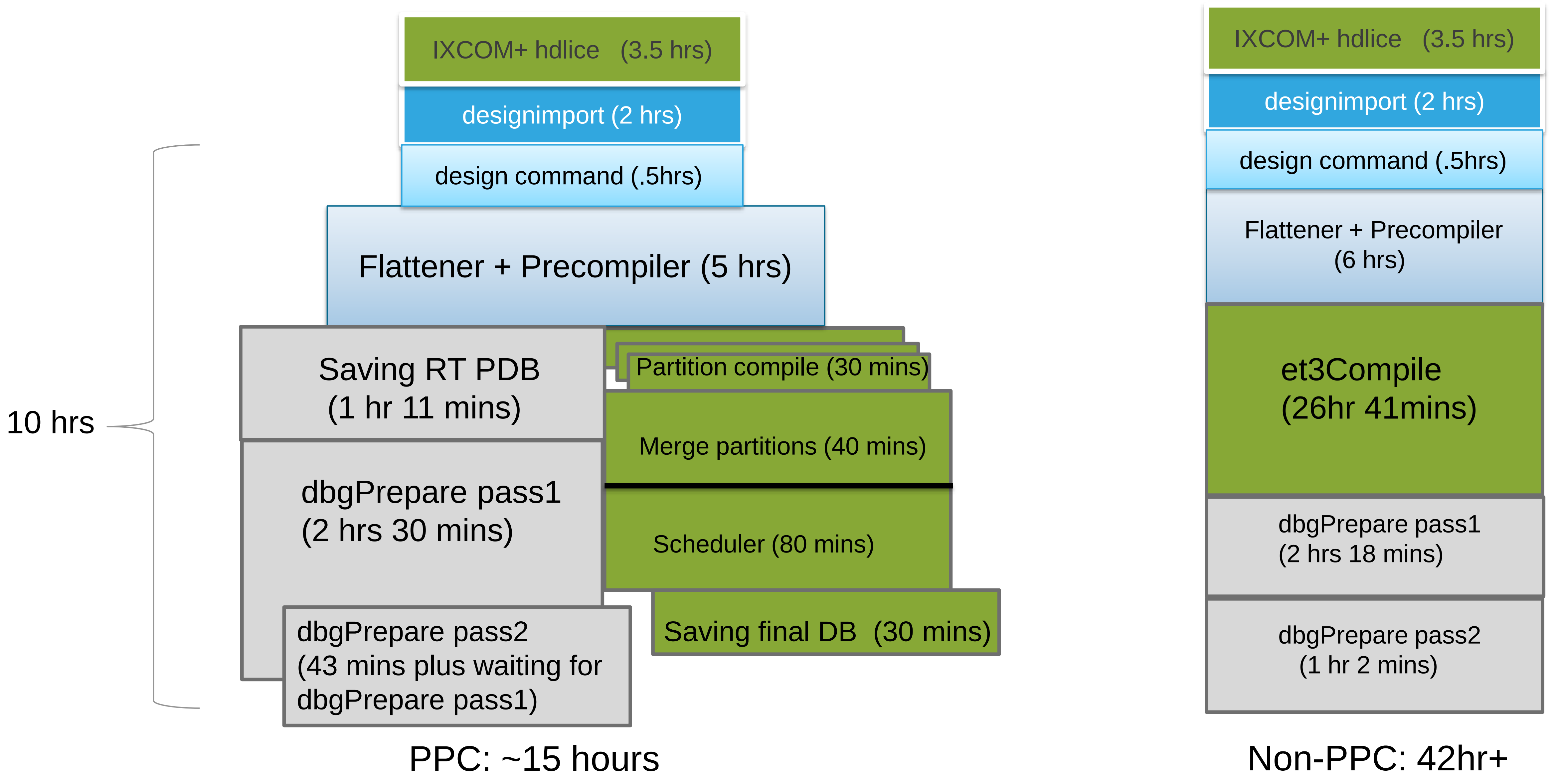
  - Did dbgPrepare pass 1 and 2 complete? These 2 passes run in parallel but overlap manner, and both must complete

```
## dbgPrepare exits at 03/06/2019 03:19:51 Status: 0, PID: 87669
## dbgPrepare Elapsed time: 7:38:25, U time: 33:46:37.56, Memory Usage: 4553.8G

## dbgPrepare exits at 03/06/2019 03:19:57 Status: 0, PID: 132057
## dbgPrepare Elapsed time: 2:11:52, U time: 29:43:54.54, Memory Usage: 833.4G
```

  - Did xeCompile exit with no errors?

Performance Comparison on a 3.78-billion-gate Design


IXCOM+ hdlice (3.5 hrs)
designimport (2 hrs)
design command (.5hrs)
Flattener + Precompiler (5 hrs)
10 hrs
Saving RT PDB (1 hr 11 mins)
Partition compile (30 mins)
Merge partitions (40 mins)
Scheduler (80 mins)
dbgPrepare pass1 (2 hrs 30 mins)
dbgPrepare pass2 (43 mins plus waiting for dbgPrepare pass1)
Saving final DB (30 mins)
PPC: ~15 hours
IXCOM+ hdlice (3.5 hrs)
designimport (2 hrs)
design command (.5hrs)
Flattener + Precompiler (6 hrs)
et3Compile (26hr 41mins)
dbgPrepare pass1 (2 hrs 18 mins)
dbgPrepare pass2 (1 hr 2 mins)
Non-PPC: 42hr+

PPC: Considerations

- Designed for larger designs, in excess of 500M gates

- MUST have LSF or similar load-sharing facility available
  - Machines in LSF queues must have adequate resources - 1GB memory for 1 million gates
  - Limited error recovery for LSF compile is supported, will restart failed partitions
    - compile –recoverPPC

- If using manual partitioning
  - Recommended number of partitions is between 10 – 30.
  - Partition should be of equal size so they complete compile in equal time

- Refer to VXE User Guide on how to convert the process if choosing to use other load-sharing facility

PPC: Considerations

- Optimal performance is achieved by NOT running precompile command
  - Specify all precompile options using precompileOption –add command in qel script
  - Then only use the compile command , this will automatically call precompile
- Optimizations are performed on partitions but they may not result in same optimization performed on the design as a whole
- Expectation is for 20% increase in emulator resources from sequential compiles
- Uneven distribution of memory in partitions can cause compile failure

PPC: Compile Failure

- PPC compile failure will exit the compile
  - Other running partition compile jobs are terminated and xeCompile is aborted

- Logs for LSF / NC jobs for PPC are in tmp directory
  - Naming is reportFromPartition<partNumber>.log

- Partition compile error can be related to LSF issues. For example:
  - LSF queue / machine specification
    - wrong queue, machines resource mismatch, i.e. not enough memory
  - Machine's workload causes timeout

- Can restart compile and recover from PPC errors
  - compile -recoverPPC
    - Will check for PPC job error, recompile only those jobs and continue.

Post PPC Issues

- If final scheduling does fail due to lack of resources
  - Can retry with existing partition data
    - Recompile partitions, and re-do final scheduling
    - New seeds may alter how logic is aligned in partitions
  - Use the xeCompile commands to retry

```
design <DesLib> <topCell>
compilerOption –add {PostMergeOpt 2}  # OPTIONAL
compile -etOnly
```

  - For IXCOM compile, can use -target xecompile:skipimport alternatively
    - Create compile.qel with the above commands
    - IXCOM starts compile at xeCompile stage, skips import and uses compile.qel

Post PPC issues

- dbgPrepare can be a time consuming compile stage in PPC compile
  - dbgPrepare creates databases for debugging features at runtime

- dbgPrepare is done in two stages
  - pass1 takes more time and more memory
  - Pass2 will start concurrently but must wait for pass1 to complete
  - Memory footprint can be sum of both passes:

```
## dbgPrepare exits at 04/29/2019 02:51:34 Status: 0, PID: 168252
## dbgPrepare Elapsed time: 2:19:58, U time: 52:15:15.08, Memory Usage: 3028.4G
## dbgPrepare exits at 04/29/2019 03:38:43 Status: 0, PID: 168156
## dbgPrepare Elapsed time: 1:40:15, U time: 16:12:12.44, Memory Usage: 717.9G
```

  - To compile single dbgPrepare pass
    - dbgPrepare hdldb xcva_top -pass1 -mc
    - dbgPrepare hdldb xcva_top -pass2
  - To compile pass1 & pass2:
    - dbgPrepare hdldb xcva_top

Outline

Parallel Netlist (PNL) for IXCOM flow

Parallel Invocation of HDL-ICE Analyzer

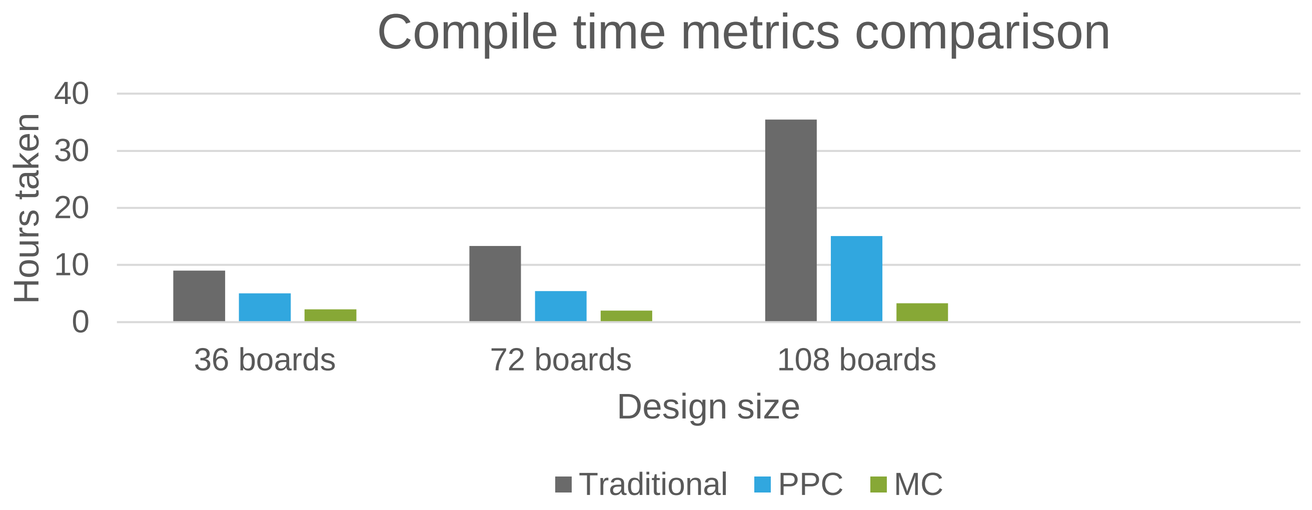
Parallel Synthesis

Parallel Partition Compiler (PPC)

Modular Compilation Flow (MC)

Modular Compilation: Highlights

- Targeted to handle large designs without the need for powerful machines
- Better compilation turnaround time and lower memory footprints
- Distributes most of the compilation steps to compute farms like LSF or NC
  - A new command called diCompile controls distributed compilation.

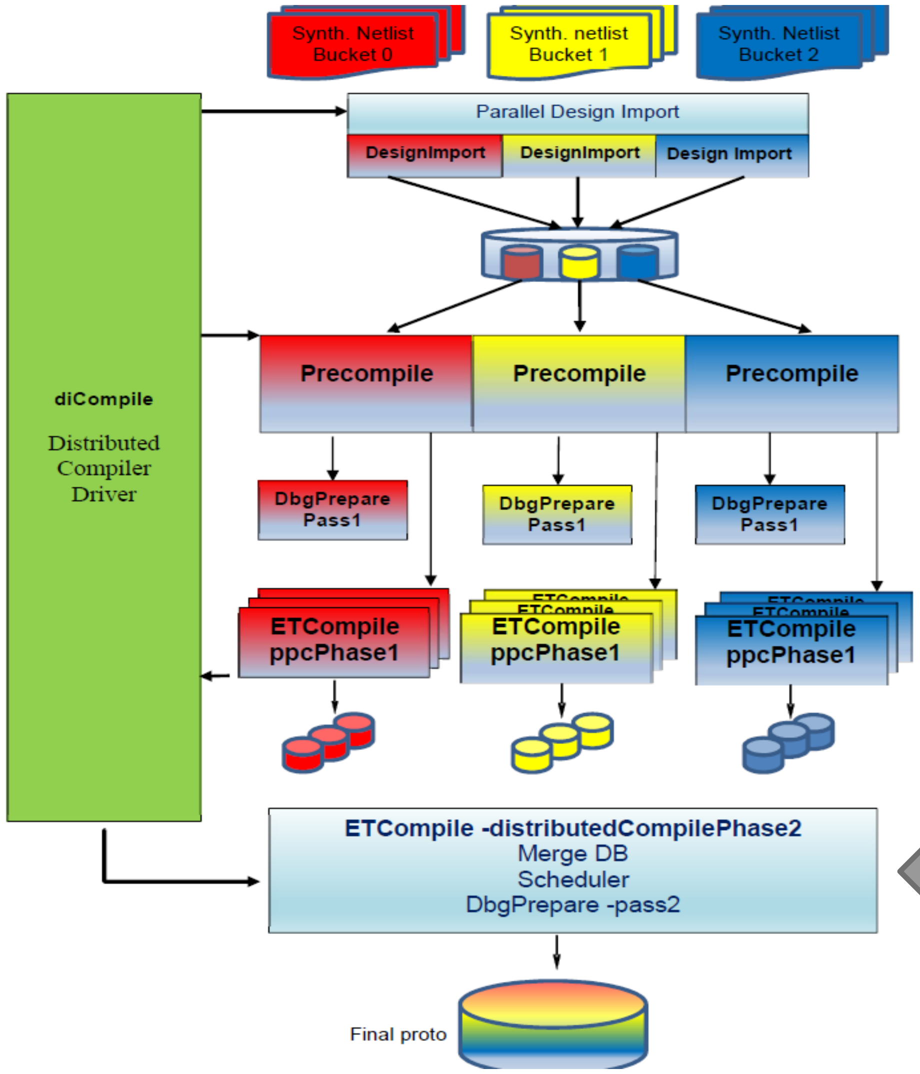


- For a 36-board design, MC is 2x faster than PPC
- For a 72-board design, MC is 2.7x faster than PPC
- For a 108-board design, MC is 4.5x faster than PPC

cādence®

Modular Compilation: Flow Chart

Synchronization points via diCompile
- At the end of parallel designImport
- During Precompile (through handshake)
- At the end of all ETCompile-ppcPhase1 jobs
- At the beginning of dbgPrepare-pass2 (wait for pass1 jobs to finish)



{advancedPart ON} is turned on by default and cannot be turned OFF.

{distributedMulticore ON} has no effect.

ETCompile -distributedCompile phase2 is launched on local host and multi threaded

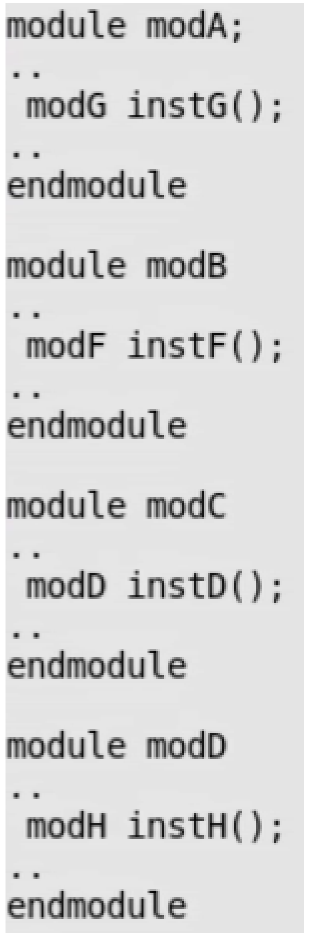
Modular Compilation: Preparation

- Design partitioning
  - User needs to partition the design into buckets
  - Each bucket should be big and logically distinct
  - In simple words, modules from one bucket should not instantiate those from another bucket
  - How big can each bucket be? What's the size of each bucket?
    - Sample DUT size: 10 Billion gates
    - Bucket size can be in the range of 500 Million gates to 5 Billion gates
    - Best MC performance is achieved by balanced distribution of modules into buckets
  - What if a user doesn't have any design knowledge? How to split the design?
    - Use gateCount utility and find the gate count info per module: gateCount <libName> <cellName>

- bucketModules.lst (file name is case sensitive)
  - Needed for providing user defined bucketing info
  - User creates this file with module names only. No instance names allowed
  - Buckets are internally numbered 1 … N
  - Bucket 0 is implicit to contain design top and anything not included in user defined buckets
  - Bucket numbering is not based on the order of the listed modules in bucketModules.lst

Modular Compilation: Puzzle#1

- Sample design top snippet



```
module modA;
..
 modG instG();
..
endmodule

module modB
..
 modF instF();
..
endmodule

module modC
..
 modD instD();
..
endmodule

module modD
..
 modH instH();
..
endmodule
```

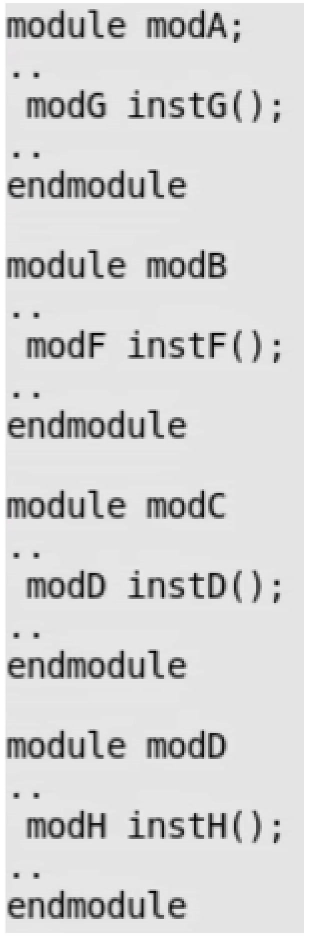
- Sample bucketModules.lst:
  - modA
  - modB
  - modC
  - modD

- Is the above bucketModules.lst appropriate?

cādence®

Modular Compilation: Answer to Puzzle#1

- Sample design top snippet



```
module modA;
..
 modG instG();
..
endmodule

module modB
..
 modF instF();
..
endmodule

module modC
..
 modD instD();
..
endmodule

module modD
..
 modH instH();
..
endmodule
```

- Sample bucketModules.lst:

  modA
  modB
  modC
  modD

- Is the above bucketModules.lst appropriate?
  – No. modD is instantiated inside modC, so it should not be included in the list.

Modular Compilation: Preparation (Continued)

- Ensure compute farm job management SW is installed (like LSF or NC)
- Provide required compute farm job options
  - Instead of using LSF_options file as is done in non-modular compilation, modular compilation relies on a JSON configuration file named computeFarm.json that must be accessible in the design directory.
    - Unlike LSF_options, computeFarm.json is mandatory for MC flow
    - Add “set MC(computeFarm) auto” in compile-time qel file
    - Precompile (aka bucket) jobs are launched with a single job ID
    - All precompile jobs are launched together when compute farm resources are available

Modular Compilation: Preparation (Continued)

- Sample computeFarm.json for LSF

```
{
  "farmName" : "LSF",
    "farmOptions" : {
      "Partitions" : "-M 150000 -R \" select[OSNAME==Linux &&
(OSTYPE==CL&&(OSMJR==6))||(OSTYPE==RL&&(OSMJR==5||OSMJR==6||OSMJR==7))||(OSTYPE==
SL&&(OSMJR==11))] rusage[mem=40000]\" -q secondary",
      "Buckets" : "-M 150000 -R \" select[OSNAME==Linux &&
(OSTYPE==CL&&(OSMJR==6))||(OSTYPE==RL&&(OSMJR==5||OSMJR==6||OSMJR==7))||(OSTYPE==
SL&&(OSMJR==11))] rusage[mem=40000]\" -q secondary",
      "Import" : "-M 200000 -R \" select[OSNAME==Linux &&
(OSTYPE==CL&&(OSMJR==6))||(OSTYPE==RL&&(OSMJR==5||OSMJR==6||OSMJR==7))||(OSTYPE==
SL&&(OSMJR==11))] rusage[mem=50000]\" -q secondary" ,
      "dbgPrepare" : "-M 150000 -R \" select[OSNAME==Linux &&
(OSTYPE==CL&&(OSMJR==6))||(OSTYPE==RL&&(OSMJR==5||OSMJR==6||OSMJR==7))||(OSTYPE==
SL&&(OSMJR==11))] rusage[mem=40000]\" -q secondary"
    }
}
```

- Sample computeFarm.json for NC

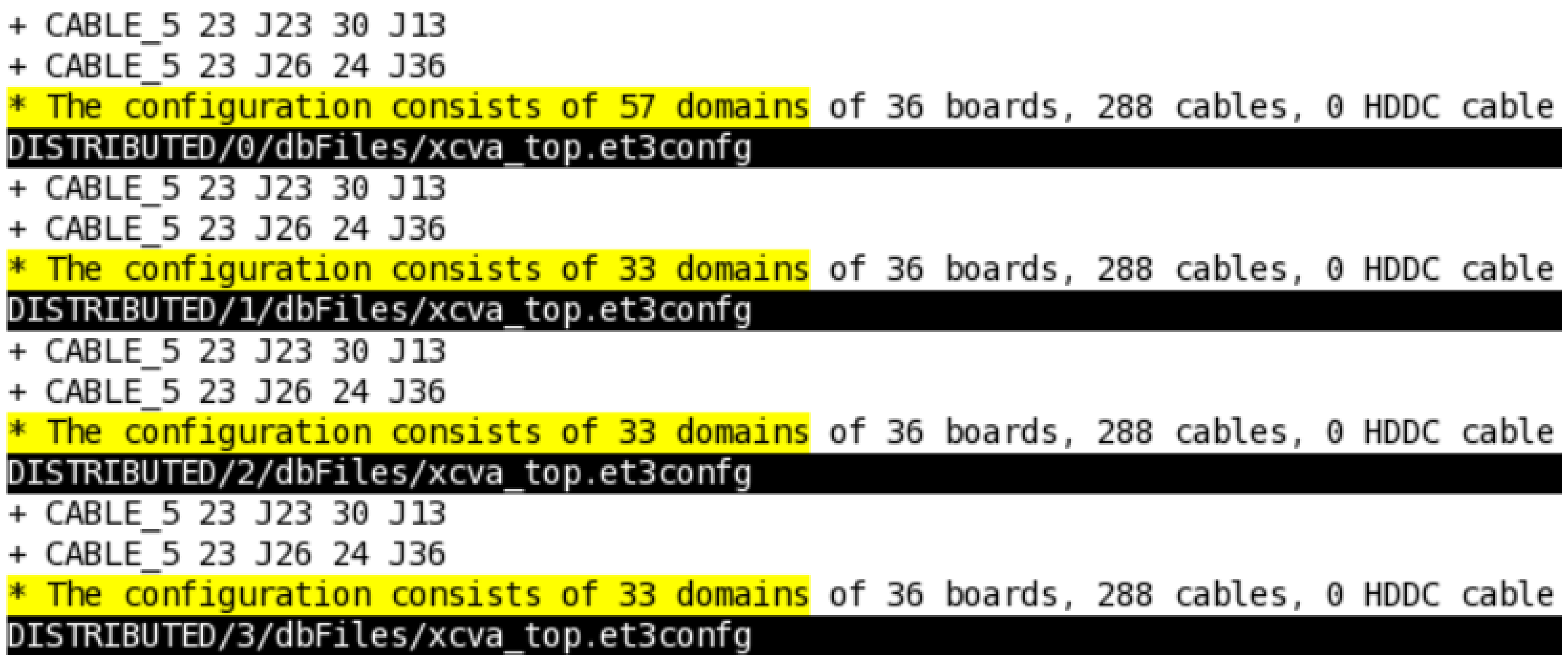
```
{
  "farmName" : "NC",
    "farmOptions" : {
      "Partitions" : "-D",
      "Buckets" : "-D",
      "Import" : "-D" ,
      "dbgPrepare" : "-D"
   }
}
```

Modular Compilation: Puzzle#2

- Where can I find the number of emulator domains used for each MC buckets?

Modular Compilation: Answer to Puzzle#2

- Where can I find the number of emulator domains used for each MC buckets?
  - This information is in DISTRIBUTED/*/dbFiles/<design>.et3confg files:



```
+ CABLE_5 23 J23 30 J13
+ CABLE_5 23 J26 24 J36
* The configuration consists of 57 domains of 36 boards, 288 cables, 0 HDDC cable
DISTRIBUTED/0/dbFiles/xcva_top.et3confg
+ CABLE_5 23 J23 30 J13
+ CABLE_5 23 J26 24 J36
* The configuration consists of 33 domains of 36 boards, 288 cables, 0 HDDC cable
DISTRIBUTED/1/dbFiles/xcva_top.et3confg
+ CABLE_5 23 J23 30 J13
+ CABLE_5 23 J26 24 J36
* The configuration consists of 33 domains of 36 boards, 288 cables, 0 HDDC cable
DISTRIBUTED/2/dbFiles/xcva_top.et3confg
+ CABLE_5 23 J23 30 J13
+ CABLE_5 23 J26 24 J36
* The configuration consists of 33 domains of 36 boards, 288 cables, 0 HDDC cable
DISTRIBUTED/3/dbFiles/xcva_top.et3confg
```

Modular Compilation: IXCOM Mode Mandatory Switches

- Enable parallel design import using IXCOM option -parallelNL
  - ixcom -parallelNL <other required ixcom options>

- Enable MC flow by invoking IXCOM using “-xeCompileArgs -mcBucket”
  - ixcom -parallelNL -xeCompileArgs “-mcBucket”
  - bucketModules.lst should be in the compile directory
  - bucketModules.lst cannot be left empty

Modular Compilation: ICE Mode Commands

- MC flow in ICE mode starts from using generated QTDB libraries
  - User needs to generate the QTDB libraries
- User is expected to update compile.qel with below MC commands
  - *compilerOption -add {distributedLibraries ON}*
    - *Enables* Modular Compilation for ICE flow
  - *setModularCompile*
    - *Checks for user errors before distributed compilation starts*
  - *set MC(computeFarm) auto*
    - Required to use the json based compute farm framework
  - *diCompile*
    - Launches the distributed compilation

Modular Compilation: Puzzle #3

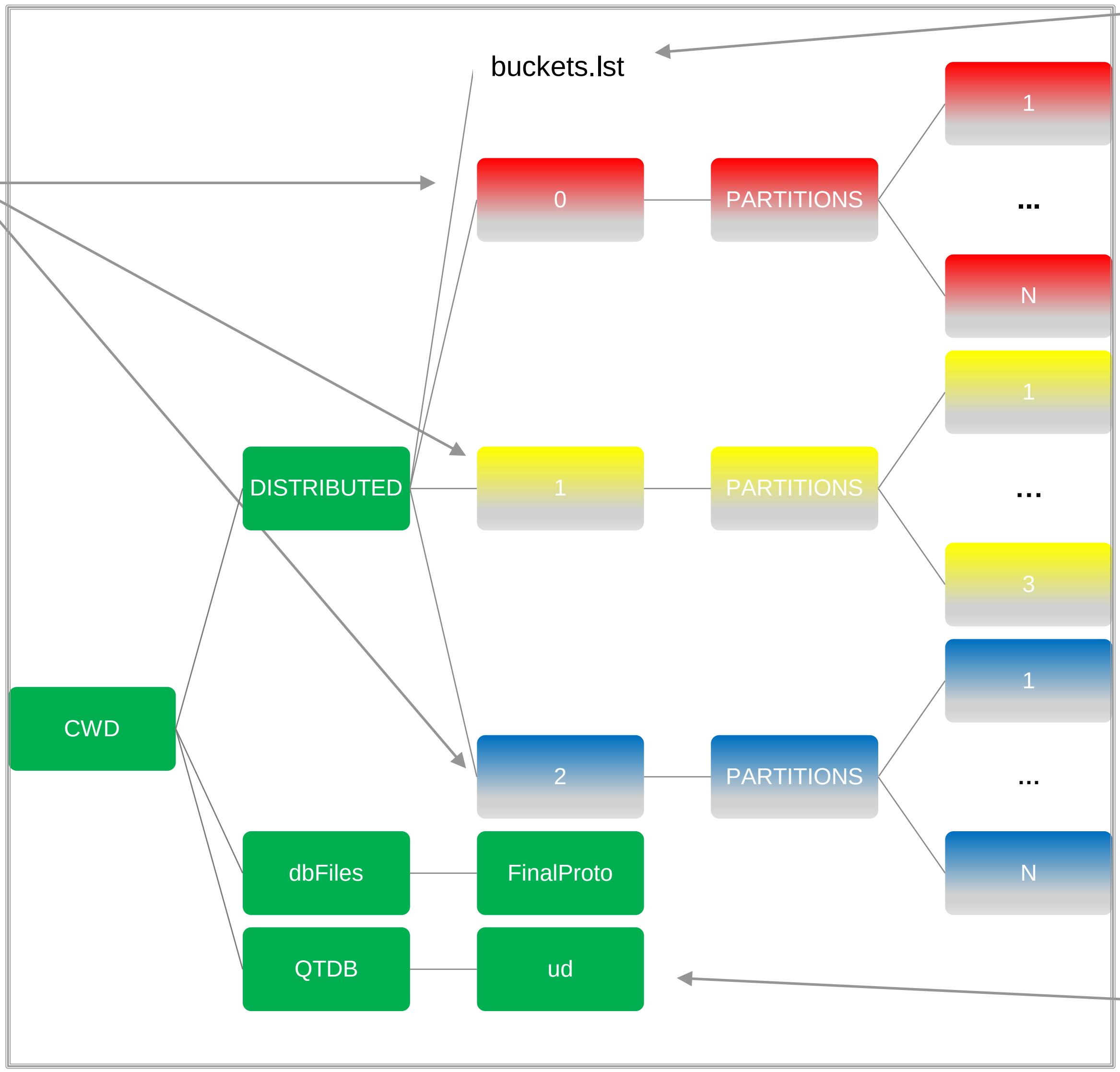
- Is the number of PPC autopartitions under each MC bucket fixed?

Modular Compilation: Answer to Puzzle #3

- Is the number of PPC partitions under each MC bucket fixed?
  - No. They'll vary based on the size of the MC bucket if tool does the partitioning
    - PPC is turned on by default with {advancedPart ON}
  - Yes, if user creates the partitions manually by using below command
    - compilerOption -add {advancedPart N} ; N = 1,2…

Modular Compilation: Generated Directories

In this scenario, user created 2 bkts + bkt0
Sub dir for each bkt is under DISTRIBUTED dir



This file has hierarchical instance names for each bucket module listed in bucketModules.lst

Has user-defined precompile and compile information

Modular Compilation: Puzzle #4

- How does a user know what is inside each bucket?

Modular Compilation: Answer to Puzzle #4

- How does a user know what is inside each bucket?
  - From DISTRIBUTED/buckets.lst
  - Or, use the run-time debug command
    - debug . -listbuckets
    - Bkt0 is special and will be displayed as {}

Modular Compilation: Using Compile Find Capability

- MC supports compile find capability provided by diCompileFind command
  - Multiple compile trials for larger designs to improve success rate
  - Faster compile time and reduced memory footprint compared to PPC
  - Use same compute resources as normal MC compile

- Main diCompileFind Options
  - -parallel: run all trials in parallel, and keep the solution with best step count
  - -serial: run all trials sequentially, and keep the solution with best step count
  - -serial_best: run trials sequentially and aborting any trial which is not an improvement
  - -serial_first: run all trials sequentially and stops at the first successful compilation
  - Options above are MUTUALLY EXCLUSIVE, in which the default is -serial_best
  - Running all diCompileFind trials across compute farm can cause issues: disk usage for multiple trials can be large

Modular Compilation: Full Debug Visibility

- The default option offers full debug visibility during a run-time session
  - Gives xeDebug access to the entire design PDB
  - Each bucket’s PDB is opened by a PDB remote server
  - PDB server can run on local host, list of specified machines or across LSF farm
  - Complete design waveform can be viewed

- Use existing debug command
  - debug <path>

Modular Compilation: Full Debug Visibility (Continued)

- Parameters or environment variable used to distribute across multiple machines
  - Default is to run all PDB server on local host
  - Bucket 0 always runs on the local host
- Two new parameters to control the launch of the PDB servers
  - xeset pdb_server local | lsf | {<host1> <host2> ...}
    - Specify the hardware to run remove servers
  - xeset pdb_remote rsh| ssh
    - Choose connection type to remote server hardware when using hosts
  - Must be done before the host command
- Values can also be set with environment variables
  - setenv XE_PDB_SERVER local | lsf | “host1 host2”
  - setenv PDB_REMOTE ssh | rsh

Modular Compilation: Bucket-wise Visibility

- You also have the option to debug your design according to the defined buckets
- User needs to select which bucket to access at run time
  - By default, NO bucket is selected
  - Once selected the bucket cannot be changed in the same debug session
- Debug capabilities are limited to current bucket scope
  - KeepNet is required to access non-current bucket signals for commands like value, drtl, sdl, etc.
  - User can use memory commands on non-current bucket memory arrays without any keepNets

Modular Compilation: Bucket-wise Visibility (Continued)

- Ways to choose bucket at run time
  - 1. Using debug command in run script
    - debug . -bucket <leaf/bkt module inst name | bucket_number | none >
    - Recommended to use leaf module instances rather than bucket number:
      - debug . -bucket 2
      - debug . -bucket designTop.Inst2 *# Best approach*
      - debug . -bucket none *# Regression mode*
    - Bucket number may not remain the same when design or VXE build changes
  - 2. Using XE_DEBUG_ARGS env variable
    - best suitable for xrun mode
    - setenv XE_DEBUG_ARGS “. -bucket 2”
    - env XE_DEBUG_ARGS=". -bucket 0"
  - 3. Using xeDebug command
    - xeDebug -input "@debug . -bucket …" <other xeDebug options>

- Regression mode
  - Ideal for quick validation with limited debug capabilities

Modular Compilation: Bucket-wise Visibility (Continued)

- Online bucket mode
  - Only signals of current bucket can be seen in the waveform when generated online
  - Non-current bucket signals that are preserved by keepNets can be seen in waveform only when they are streaming probed
  - A single PHY directory is dumped
    - All the buckets 1 to N sub directories will be seen underneath PHY directory
- Regression mode
  - Regular probe command is not supported; thus, *shm/fsdb waveform cannot be generated during this online mode*
  - Dumping of the PHY file is the same as online bucket mode
- Offline mode
  - Whether PHY is generated using regression mode or bucket mode, the offline mode flow remains same
  - User can generate the waveform of any bucket (1 to N) by starting an offline debug session with –bucket <N>, add probes from this bucket, and then dump the waveform

Modular Compilation: Restrictions

- Modular compilation does not support:
  - ATB constructs
  - UPF, CPF, and Hardware-Assisted Weighted Toggle Count (HW-WTC) commands.
    - Note that Modular Compilation supports Fast Toggle Count (FTC).
  - DPA and TCA operations
  - External netlists
  - RTL database
  - Executing precompile separately from compile
  - Following database commands:
    - db
    - moduleNet
    - getDatabaseInfo
  - partitionGroup and partitionAssign commands
  - Combinational loops involving more than one bucket, if detected before scheduling.
    - Such cases are reported in advance, so that the loop can be broken manually or the bucketing strategy modified as required.

Modular Compilation: Restrictions (Continued)

- In addition to the above limitations, at times, when the simulator time is reset to 0, DCC waveform mismatch can be observed between InfiniTrace Prepare and Observe sessions.
  - This issue appears intermittently for modular-compiled designs. To workaround the issue, specify the time without the trigger keyword
    - Example: infiniTrace -goto 1225.

cādence®