Clock-in Early: Integrated Structural Multisource Clock Tree Synthesis to Push PPA in High-speed GPU Designs

Presenter: Francisco Rivera Valverde
Co-authors: Huy Cao, Milind Mahajan, Atul Walimbe
Intel

Agenda

Context on Multisource CTS

What/Why/How?

Early Clock-in Features

Results & Recommendations

Common Acronyms

Abbreviation	Definition
CTS	Clock Tree Synthesis
MSCTS	Multisource CTS Flow
SMSCTS	Structural Multisource CTS
RMSCTS	Regular Multisource CTS
ISMSCTS	Integrated SMSCTS
IRMSCTS	Integrated RMSCTS

Multisource Clock Tree Synthesis

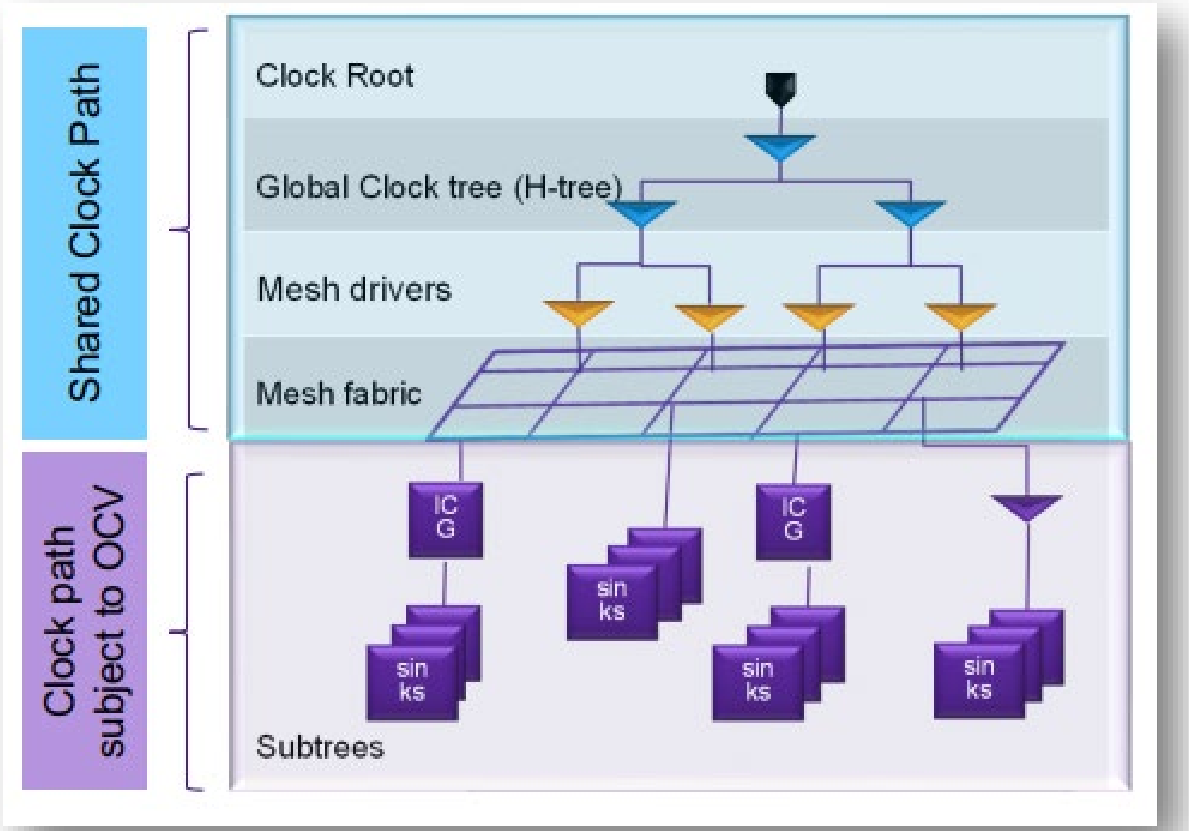
Focusing on Local Clock Tree Distribution

Multisource Clock Tree Synthesis

- On-chip-variation (OCV) motivates advanced clock techniques to achieve lower skew, improved shared clock paths and other PPA targets.

- MSCTS consists of two components:
  - **Global clock distribution:** clock mesh, spine, …
  - **Local clock distribution:** subtree's that connect the global network to the sinks

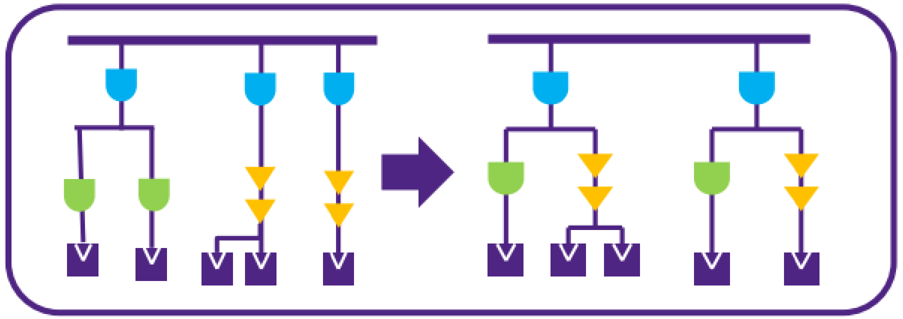
- We focus on **local clock distribution** techniques!



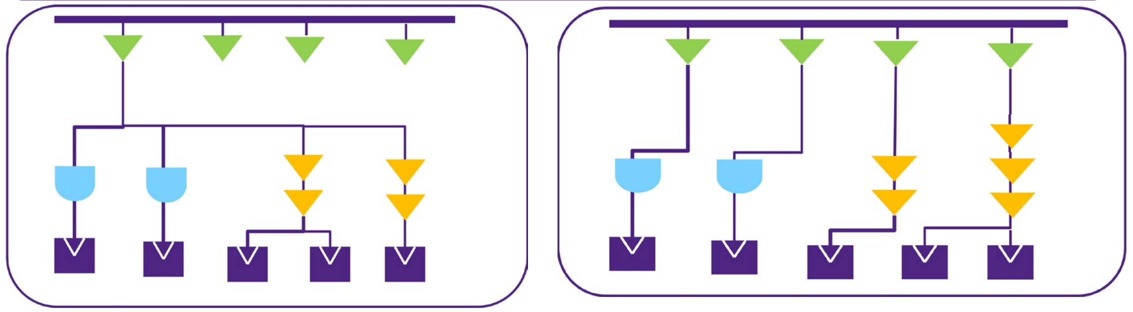
Local Clock Tree Synthesis

- There are 2 main mechanisms in which local clock trees are created in MSCTS:

Structural MSCTS (SMSCTS)

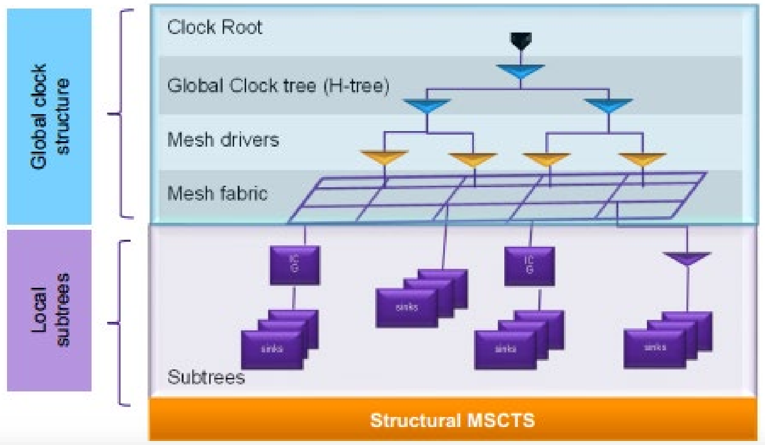


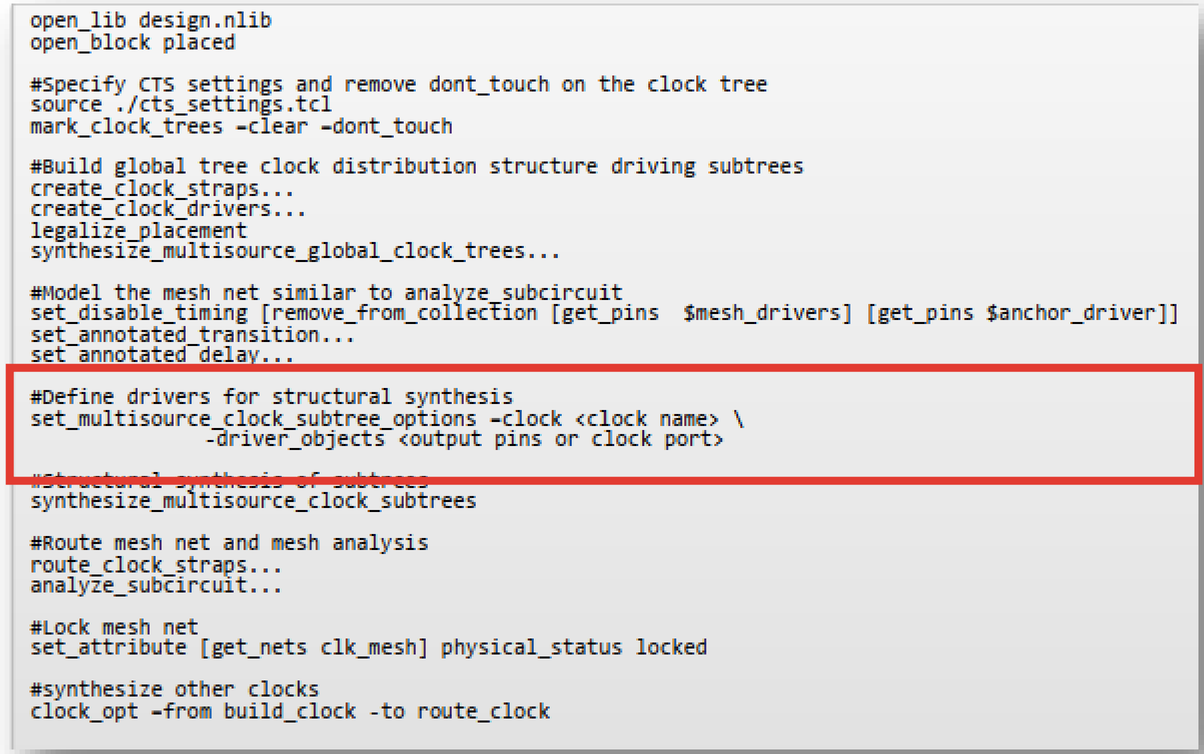
Regular MSCTS (RMSCTS)



Structural MSCTS

- Optimize for latency and skew, while minimizing clock power
- Consists of:
  - Balance: add cells to level subtrees
  - Merge: merges equivalent clock cells
  - Optimize: splitting, sizing, relocation of clock cells
  - Route
  - Refine: sizing and cell relocation
- Typically, less OCV, better skew but slightly more demanding flow/power

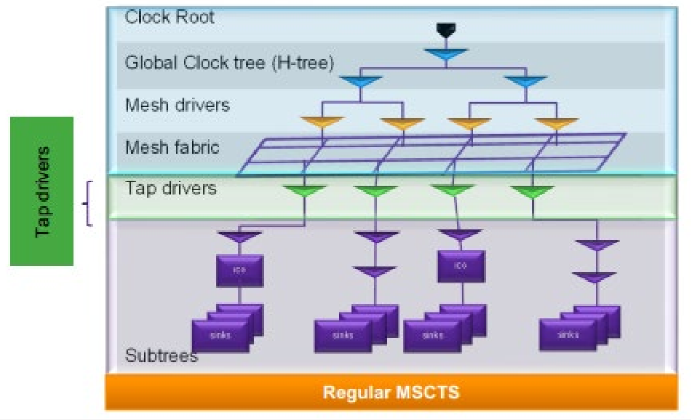


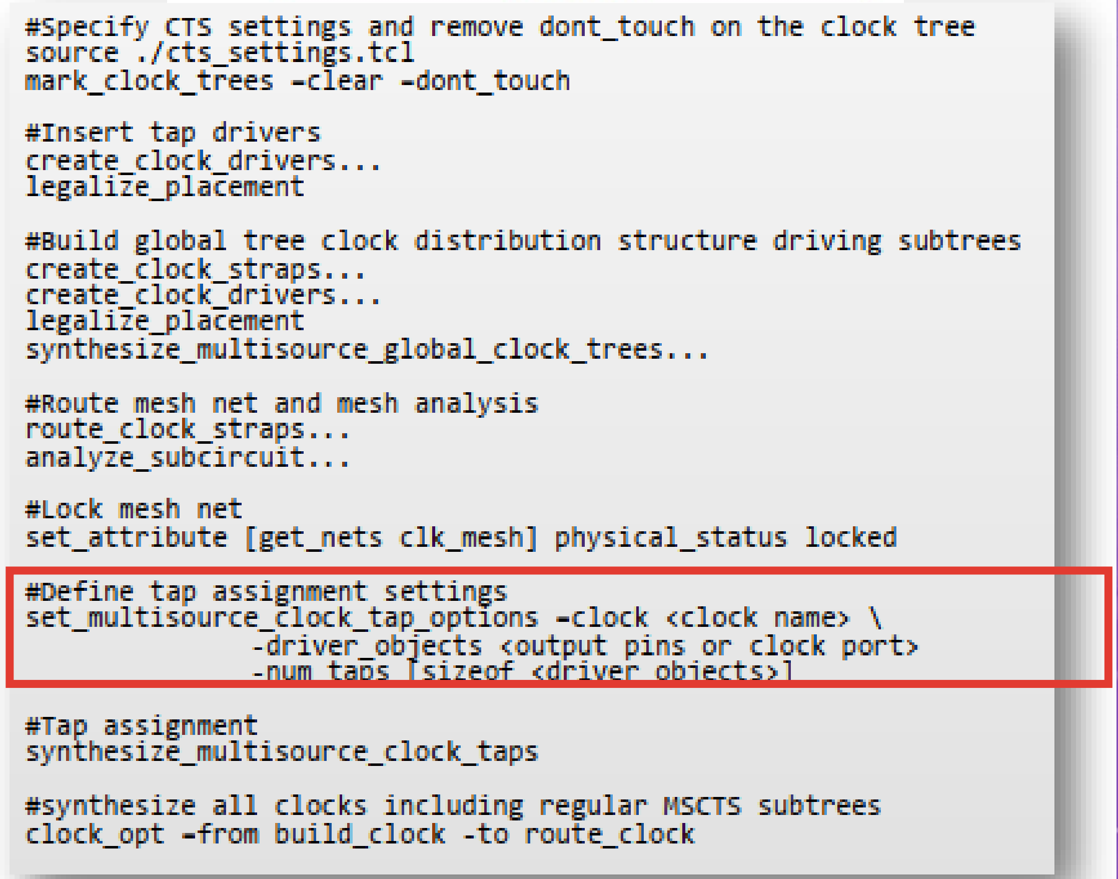


```
open_lib design.nlib
open_block placed

#Specify CTS settings and remove dont_touch on the clock tree
source ./cts_settings.tcl
mark_clock_trees -clear -dont_touch

#Build global tree clock distribution structure driving subtrees
create_clock_straps...
create_clock_drivers...
legalize_placement
synthesize_multisource_global_clock_trees...

#Model the mesh net similar to analyze_subcircuit
set_disable_timing [remove_from_collection [get_pins  $mesh_drivers] [get_pins $anchor_driver]]
set_annotated_transition...
set_annotated_delay...

#Define drivers for structural synthesis
set_multisource_clock_subtree_options -clock <clock name> \
               -driver_objects <output pins or clock port>

#Structural synthesis of subtrees
synthesize_multisource_clock_subtrees

#Route mesh net and mesh analysis
route_clock_straps...
analyze_subcircuit...

#Lock mesh net
set_attribute [get_nets clk_mesh] physical_status locked

#synthesize other clocks
clock_opt -from build_clock -to route_clock
```

Regular MSCTS

- Sink distributed among tap drivers during tap assignment
- Regular CTS is used for building the subtrees
  - No restriction on adding/removing buffers
- Lesser power than SMSCTS, very useful skew friendly

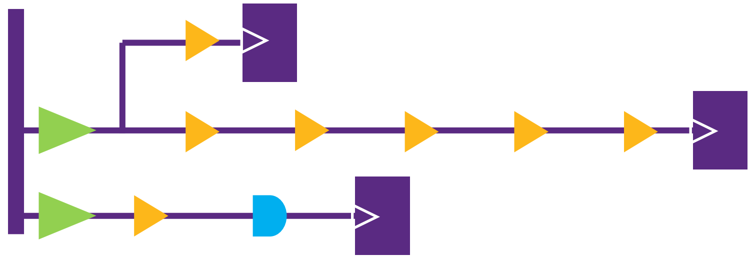




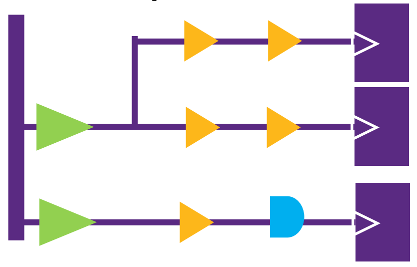
```
#Specify CTS settings and remove dont_touch on the clock tree
source ./cts_settings.tcl
mark_clock_trees -clear -dont_touch

#Insert tap drivers
create_clock_drivers...
legalize_placement

#Build global tree clock distribution structure driving subtrees
create_clock_straps...
create_clock_drivers...
legalize_placement
synthesize_multisource_global_clock_trees...

#Route mesh net and mesh analysis
route_clock_straps...
analyze_subcircuit...

#Lock mesh net
set_attribute [get_nets clk_mesh] physical_status locked

#Define tap assignment settings
set_multisource_clock_tap_options -clock <clock name> \
                -driver_objects <output pins or clock port>
                -num_taps [sizeof <driver objects>]

#Tap assignment
synthesize_multisource_clock_taps

#synthesize all clocks including regular MSCTS subtrees
clock_opt -from build_clock -to route_clock
```

A Better Way to Do Structural MSCTS

- RMSCTS “simplicity” gives the tool a lot of flexibility to:
  - Enable compile_fusion clock-aware optimizations – better clock gate optimization, concurrent clock/data (CCD) opt => Integrated Regular MSCTS
  - Easier useful skew implementation as there is no restriction on adding/removing buffers on subtrees



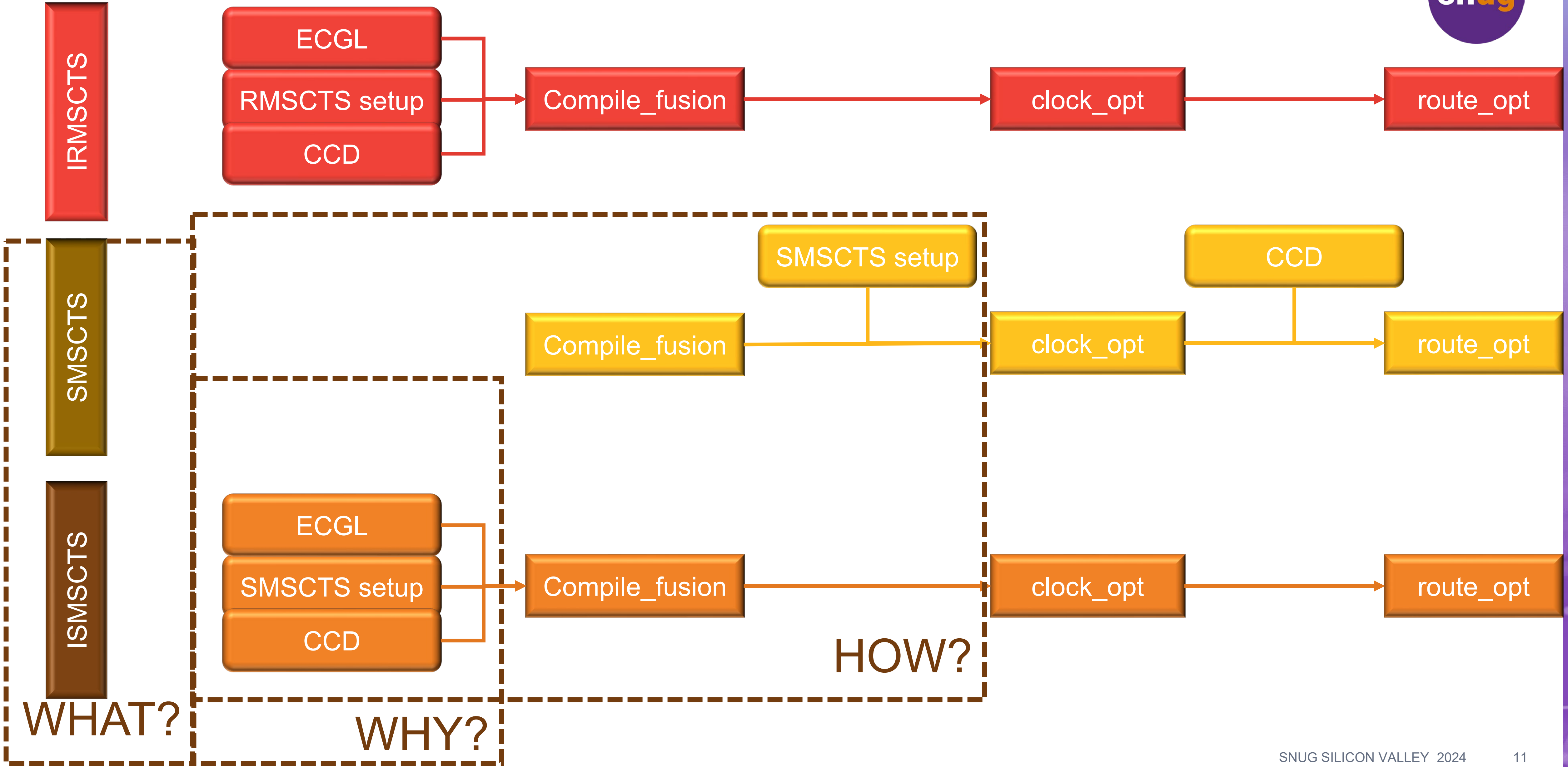
- Traditionally, SMSCTS clock trees are frozen during build clock – CCD mainly used for sizing/relocation of clock cells
  - Push/pull can be done on a post-CTS optimized database also, but limited room for CCD as compared to full flow enabling



Presentation Goal

- Enable **early** SMSCTS clock construction during compile_fusion to take advantage of **clock-aware optimization features**

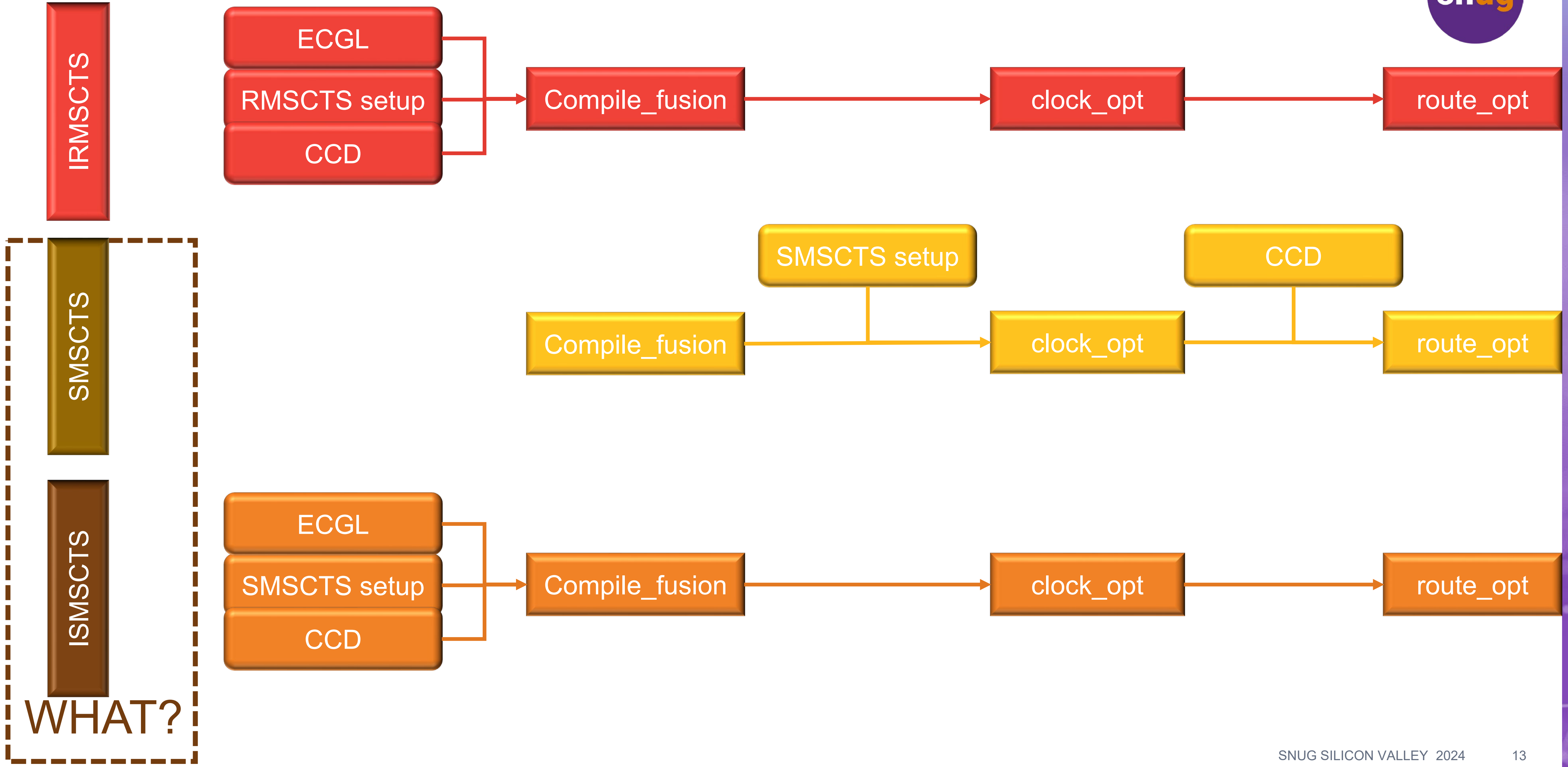
In a Nutshell….



Integrated SMSCTS

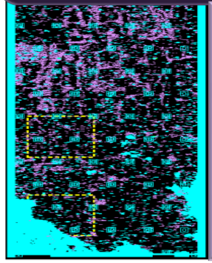
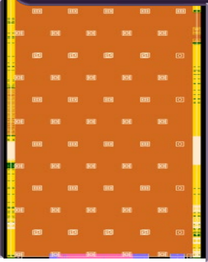
The What, Why and How this Feature Is Enabled

In a Nutshell….



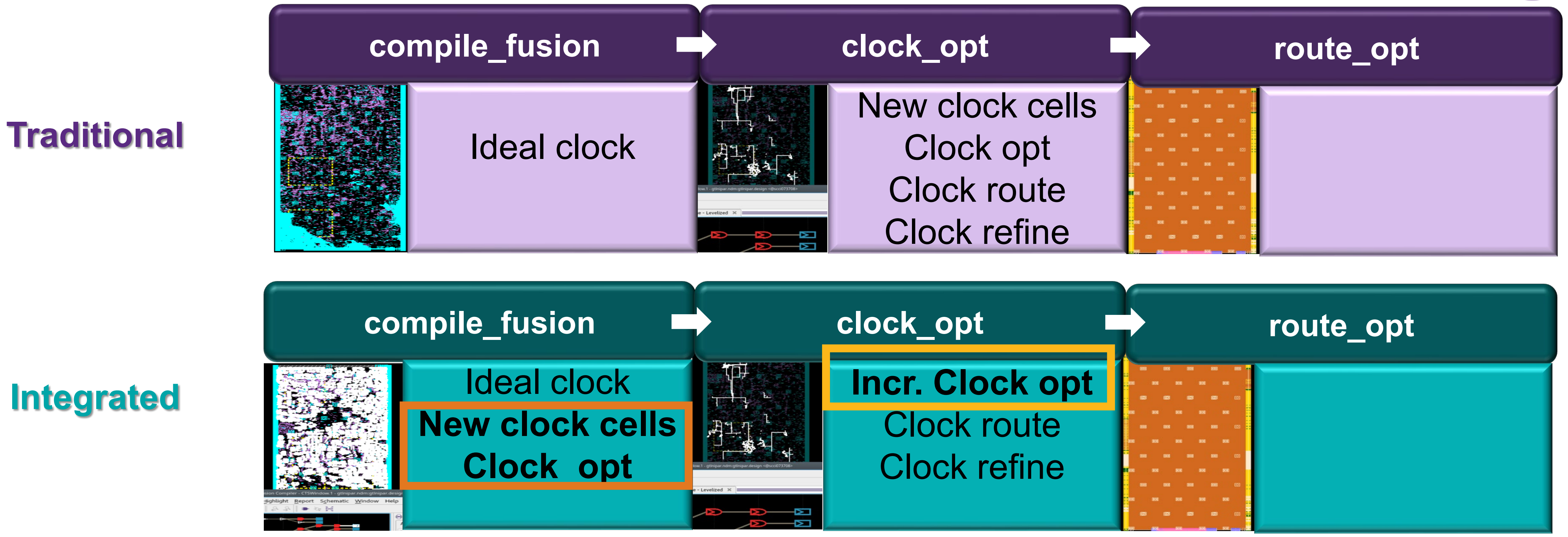
What Is Integrated SMSCTS? (Traditional)

Traditional

compile_fusion		clock_opt		route_opt	
	Ideal clock		New clock cells Clock opt Clock route Clock refine		

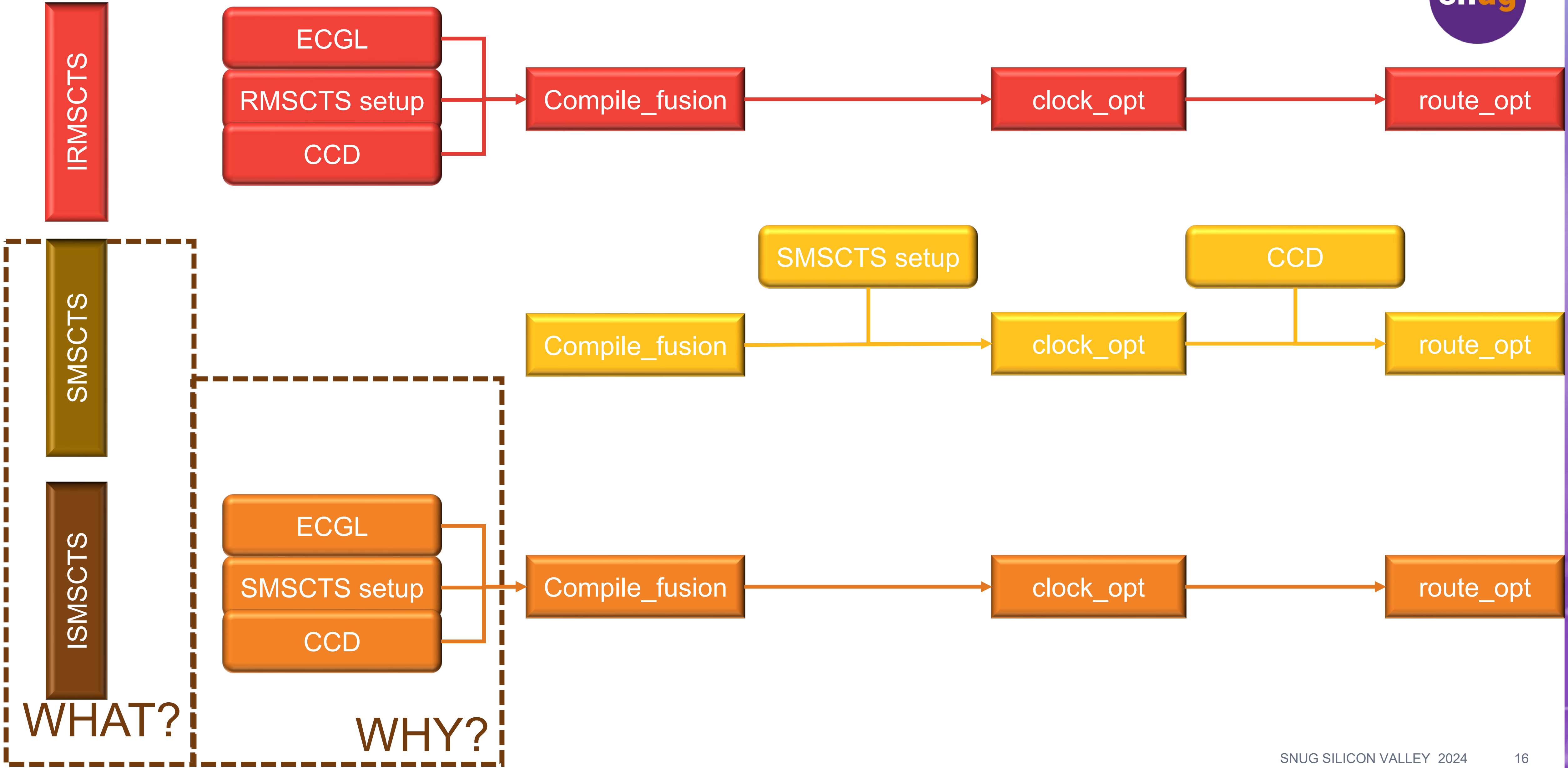
- Clock cell insertion (balancing, splitting, DRC fixing, …) happens until clock_opt
- After new cell insertion, clock network is routed
- Refine optimization are performed, with post clock data optimization as well (clock_opt –from final_opto)

What Is Integrated SMSCTS?



- **Moved** clock cell creation (balancing, merging, splitting,…) to **final_place****
- CTS performs **incremental** skew optimization, routing and post route opt
- Clocks are not propagated**, but cell presence enables clock-aware optimization

In a Nutshell….



Why We Do ISMSCTS?

Goals

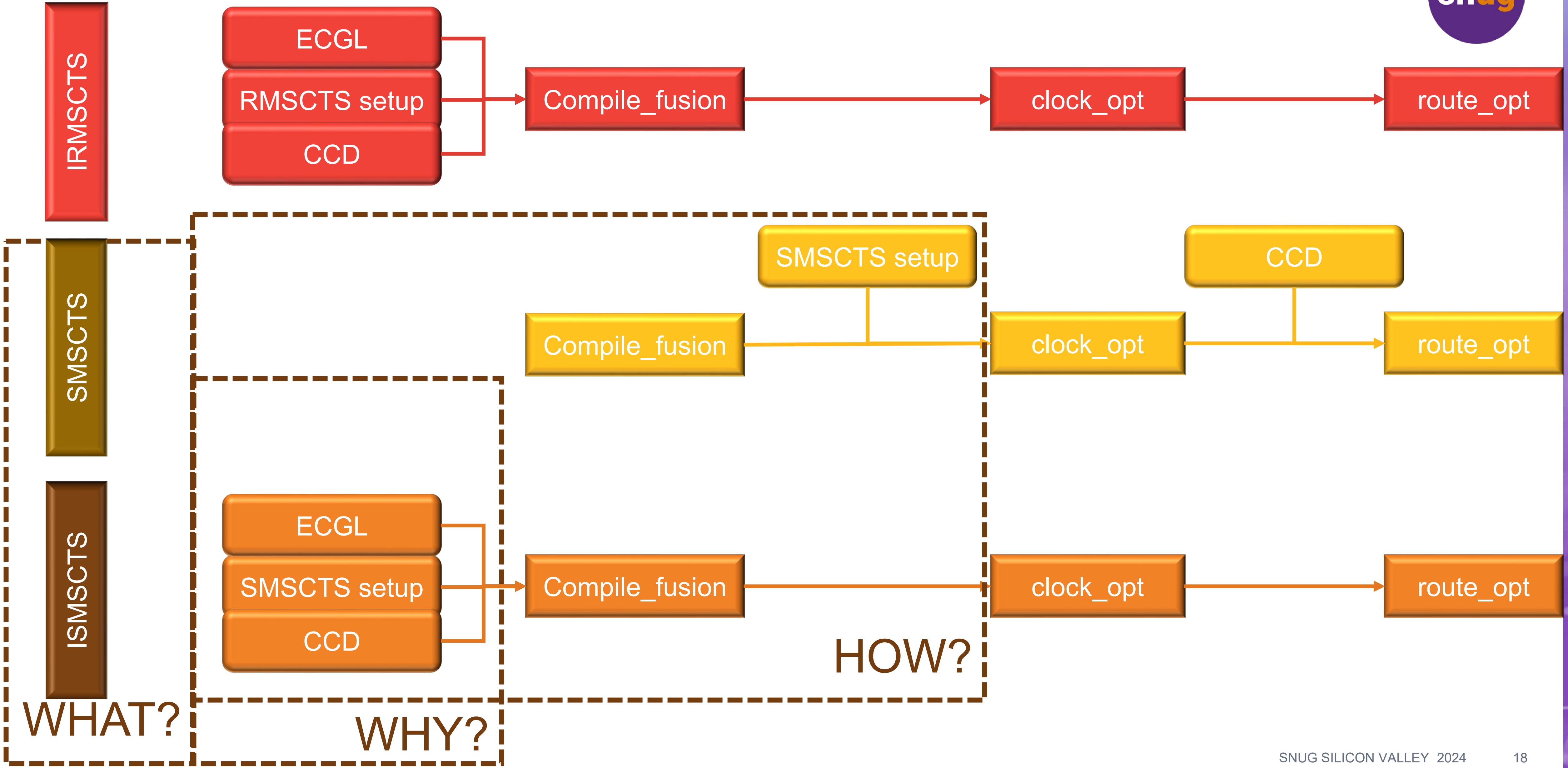
1. (**ISMSCTS**) Leverage the latest fusion_compiler enhancements that enable compile fusion to create the SMSCTS clock subtree structure logically
2. (**CCD**) Enable useful skew computation leveraging the already inserted clock cells
3. (**ECGL**) Enable estimate clock gate latency to have a better optimization perspective on enable paths

Motivation

- Cts network inserted upfront great for clock cell prioritization (placement)
  - Clock delay not propagated, expected in future releases
- Full flow CCD has more opportunity to optimize
- Realistic clock gate picture enables ECGL

How?

In a Nutshell….



Traditional Post Compile SMSCTS

init_design (elab, analyze, constrain,…)

compile_fusion –to initial_opt

compile_fusion –from final_place

source cts_settings.tcl

clock_opt –from build_clock –to route_clock

clock_opt –from final_opt

route_opt

- SMSCTS settings make sense only on a post compile/place_opt context
- Clock network is constructed only after compile_fusion/place_opt is done (build_clock).
- Typically, CCD is enabled only on a post-CTS fashion (from final_opt)

Feature#1: Integrated SMSCTS (Precompile)

init_design (elab, analyze, constrain,…)

source cts_settings.tcl

compile_fusion –to initial_opt

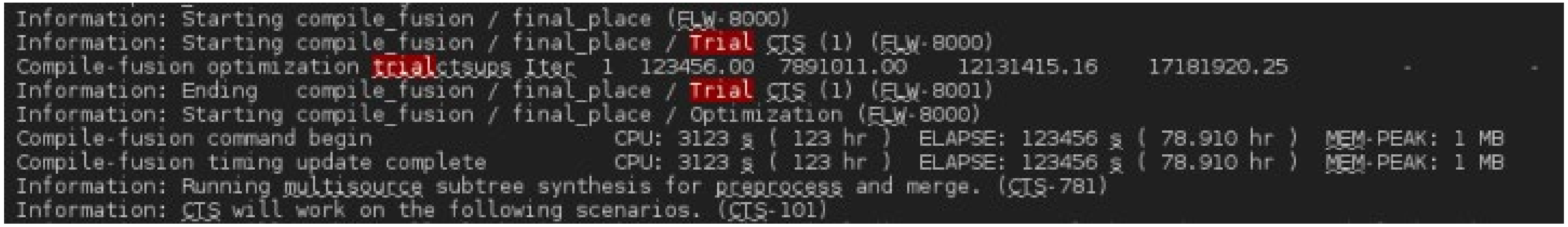
compile_fusion –from final_place

- Provide **CTS settings** before compile_fusion final_place
  - Exceptions, balance points, clock cells to use
  - set_multisource_clock_subtree_options
  - set_app_options -name compile.flow.enable_multisource_clock_trees -value true
- Make sure no user don’t touch on clock network
- Compile_fusion will trigger **Trial CTS** where clock cells are physically inserted in the DB and placed
  - No routing happens on subtree at this stage**

Feature#1: **Integrated** SMSCTS (Trial CTS)

- New Trial CTS stage will be executed during compile fusion final_place
  - Watch for CTS-* messages which traditionally would have been in clock_opt command

compile_fusion –from final_place



```
Information: Starting compile_fusion / final_place (FLW-8000)
Information: Starting compile_fusion / final_place / Trial CTS (1) (FLW-8000)
Compile-fusion optimization trialctsups Iter  1  123456.00  7891011.00    12131415.16    17181920.25         -         -
Information: Ending   compile_fusion / final_place / Trial CTS (1) (FLW-8001)
Information: Starting compile_fusion / final_place / Optimization (FLW-8000)
Compile-fusion command begin                   CPU: 3123 s ( 123 hr )  ELAPSE: 123456 s ( 78.910 hr )  MEM-PEAK: 1 MB
Compile-fusion timing update complete          CPU: 3123 s ( 123 hr )  ELAPSE: 123456 s ( 78.910 hr )  MEM-PEAK: 1 MB
Information: Running multisource subtree synthesis for preprocess and merge. (CTS-781)
Information: CTS will work on the following scenarios. (CTS-101)
```

Feature#1: ISMSCTS (Postcompile)

clock_opt –from build_clock –to route_clock

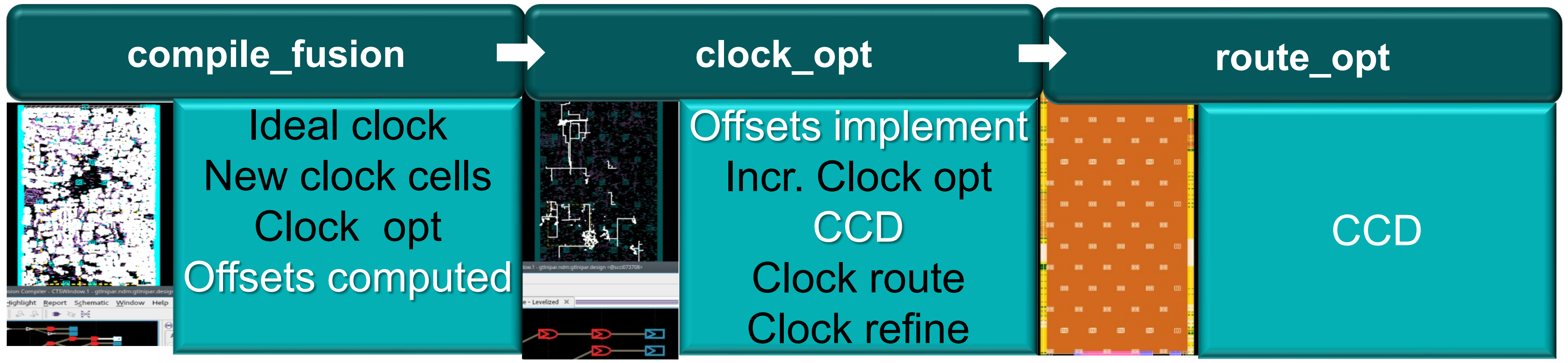
clock_opt –from final_opt

route_opt

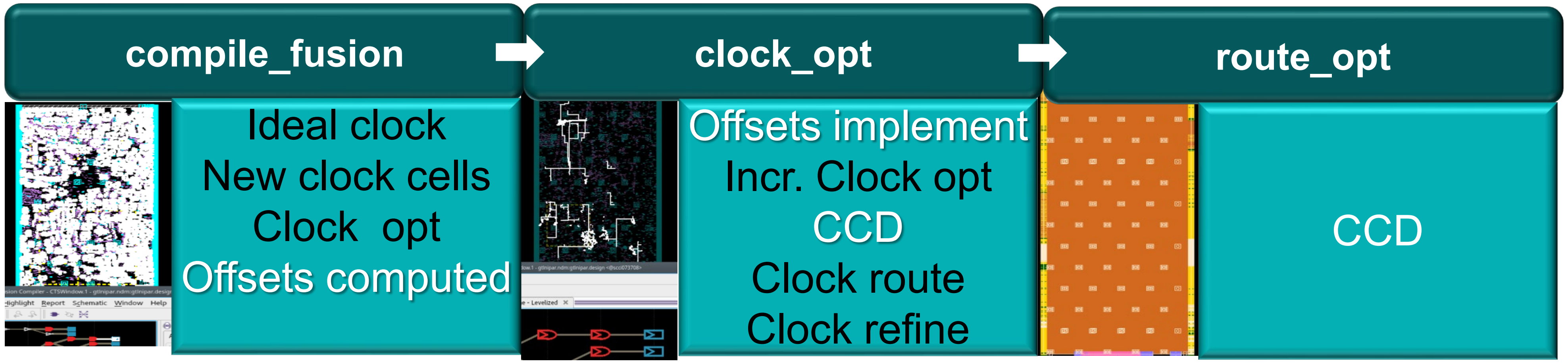
- Keep SMSCTS settings same as the ones used in pre-compile
- **Clock_opt** will understand the already created logical clock network
  - Will only perform incremental optimizations
  - Route and refine clock network
  - This typically allows a better skew as compared to non Integrated flow
- Route-aware optimization on clock network happens on this stage

Feature#2: Full Flow CCD (Enable)

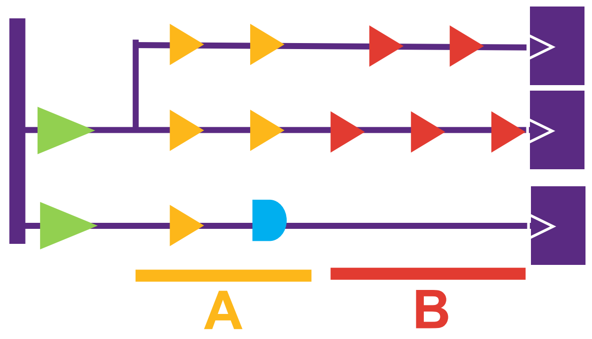
- CCD can be enabled via app options:
  - set_app_options -name compile.flow.enable_ccd -value true
  - set_app_options -name clock_opt.flow.enable_ccd -value true
  - set_app_options -name route_opt.flow.enable_ccd -value true



Feature#2: Full Flow CCD (Level Control)



- SMSCTS tree **target level** (set_multisource_clock_subtree_options –target_level A) can be different that **CCD target levels** (set_clock_tree_options -max_levels B), where B > A.



- Because ISMSCTS preserve subtree level structure, special **handshaking** is required from the tool perspective to physically implement offsets

Feature#2: Full Flow CCD (handshake)

- In ISMSCTS context, following options are required so compile fusion is aware of the subtrees while computing offsets
  - set_app_options -name cts.multisource.enable_subtree_synthesis_aware_ccd -value true
  - set_app_options -name clock_opt.flow.enable_multisource_clock_trees -value true

- The specified offsets must be realistically implementable during clock_opt build tree:
  - set_app_options -name ccd.max_prepone -value <>
  - set_app_options -name ccd.max_postpone -value <>

Feature#2: Full Flow CCD (Hierarchical)

- Hierarchical convergence can be taxing with CCD enabled.

- FC supports different boundary CCD options:
  - No ccd on boundary (ccd.optimize_boundary_timing)
  - Skew targets for boundary paths (recommended value is global skew pre-ccd)
    - set_app_options -name ccd.skew_opt_input_boundary_max_prepone -<>
    - set_app_options -name ccd.skew_opt_input_boundary_max_postpone -value <>
    - set_app_options –name ccd.skew_opt_output_boundary_max_prepone -value <>
    - set_app_options –name ccd.skew_opt_output_boundary_max_postpone -value <>

Feature#3: Native Clock Splitting

- With ISMSCTS, it is advisable to enabled native clock enable logic splitting feature:
  - set_app_options -name cts.multisource.enhanced_enable_logic_splitting –value true
  - set_multisource_clock_subtree_options -split_enable_max_width <> -split_enable_max_height <> -split_enable_max_latch_level <> -split_enable_max_fanout <>
- With realistic clock splitting, it is advisable to enable ECGL feature to help enable path closure on clock gates
  - set_app_options -name opt.common.estimate_clock_gate_latency -value true

Results and Conclusions

Context of the Blocks Used

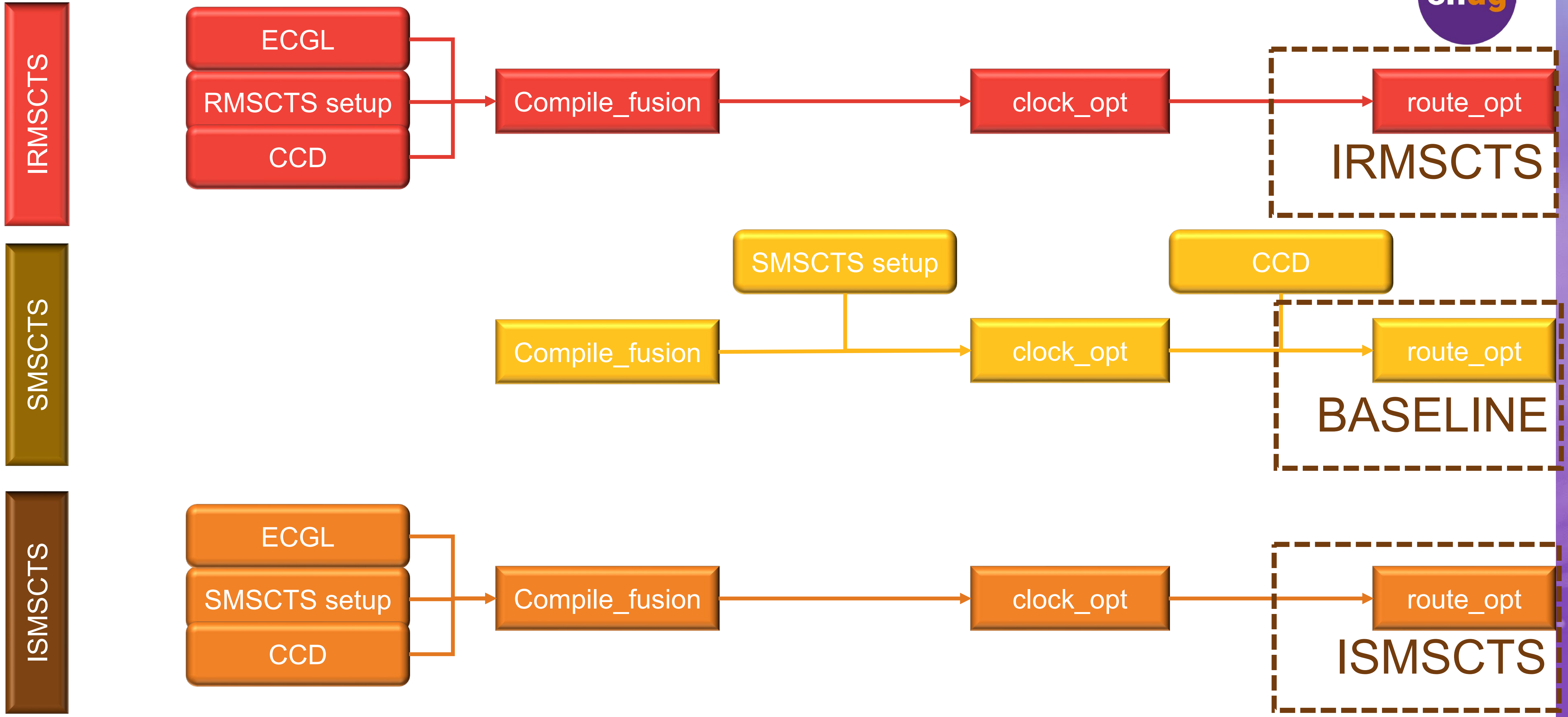
- Used Synopsys Fusion Compiler™ tool version U-2022, highly tuned flow recipe for best PPA

- Blocks details:
  – ~1 million instances blocks
  – highly congested blocks (both wire and placement)
  – 70% > utilization
  – Graphics designs with many datapaths, and muxing
  – Non-macro dominated

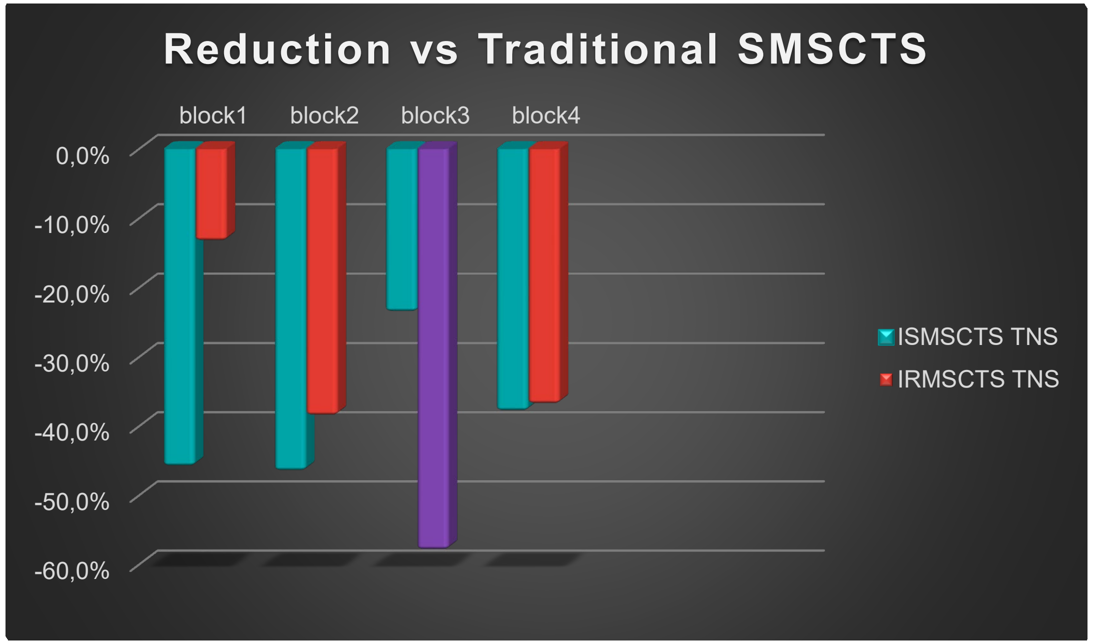
Experiment Setup

- Data is always presented after **route_opt**, with fill-insertion, and apples-to-apples comparison
- **Baseline** for all experiments is a traditional SMSCTS flow, with CCD size only
- **Target** experiments use the best recommendations from before: ECGL, CCD since final_place, ISMSCTS
- We also provide equivalent IRMSCTS results

In a Nutshell….

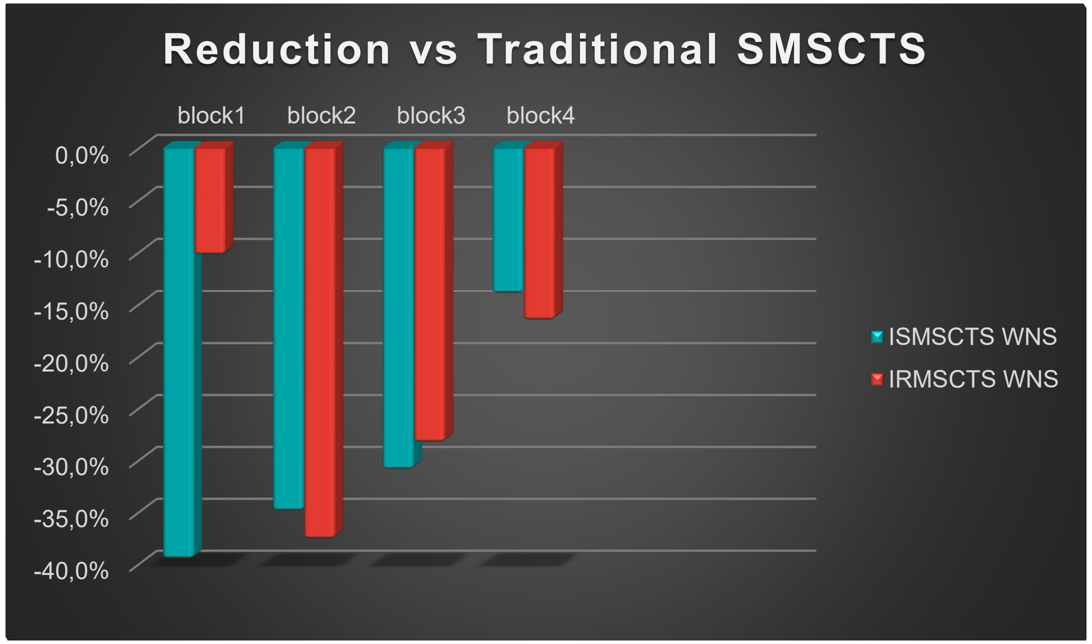


Highlights of the Feature: TNS



- ISMSCTS+features provide a significant **improvement** as compared to baseline
  - Some **blocks** are still better with RSMSCTS
- Min convergence in general is more controlled in Structural flow impacting overall optimization
- TNS reduction can be mapped to ease of convergence:
  - Less congested/buffer polluted designs at route_opt
  - Lesser violations means lesser effort for PR/manual work

Highlights of the Feature: WNS



- Early CCD has been a great enabler of frequency push:
  - For non-RTL bottle-necks, WNS can be reduced by with technique
  - Late CCD (both sizing or incremental post-CTS push/pull) has restricted opportunities to push freq-max

Room for Improvement

- With CCD, min is typically **degraded** (up to 29% degradation on ISMSCTS and 40% on IRMSCTS)
  - CCD hold effort high might be needed for high hold degradation
- With CCD, EGCL and ISMSCTS, runtime can increase up to 10%
- ISMSCTS had ISO power/area as compared to baseline
  - IRMSCTS had slightly better total power (<5%) than ISMSCTS
- We observe maximum 2 extra levels added by CCD on ISMSCTS (5% more clock buffers than baseline) and ISO clock gates

Conclusions and Future Work

- ISMSCTS flow can be considered another tool for designers to converge their block
  - Some blocks converge better with regular MSCTS as compared to structural MSCTS with CCD

- How to improve from here
  - ISMSCTS happens in final_place – discussion in progress to analyze Trial CTS on **initial_opto**
  - **Clock propagation** on compile_fusion could potentially improve total setup cost

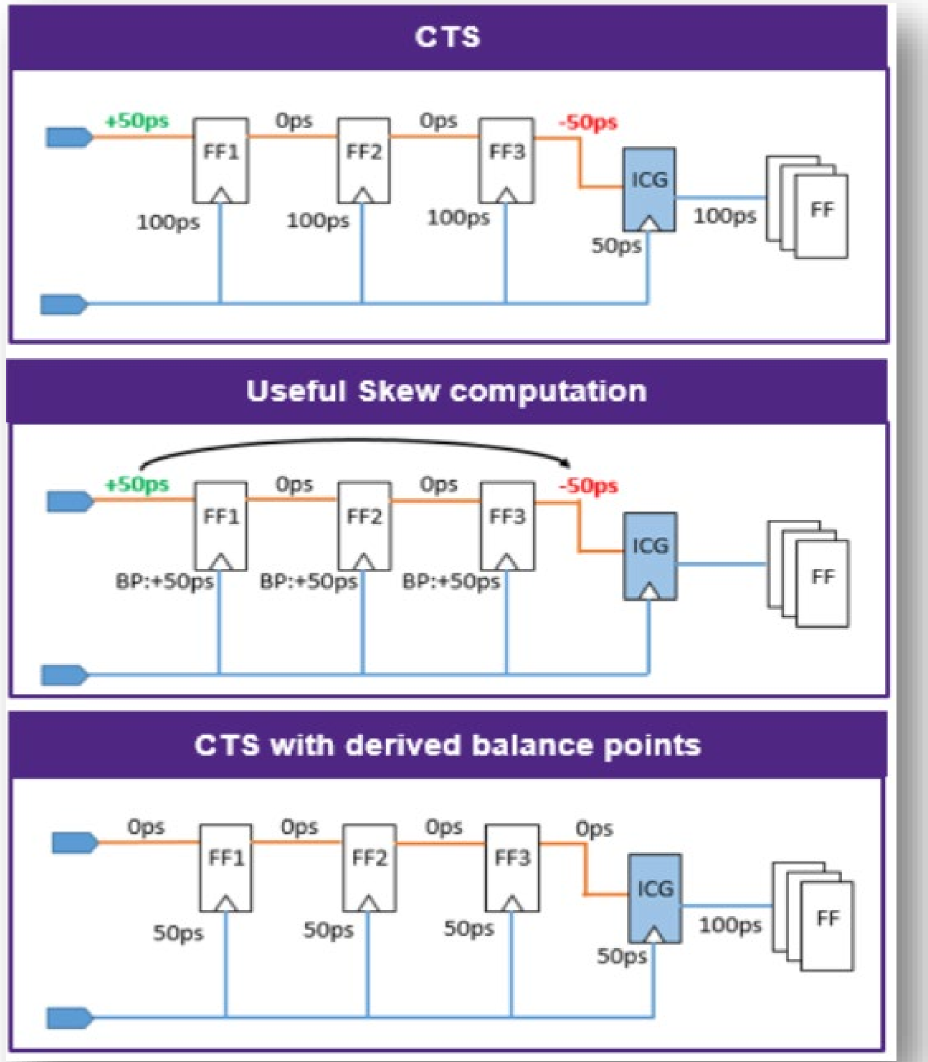
snug

THANK YOU

YOUR
INNOVATION
YOUR
COMMUNITY

Backup

Concurrent Clock and Data Opto

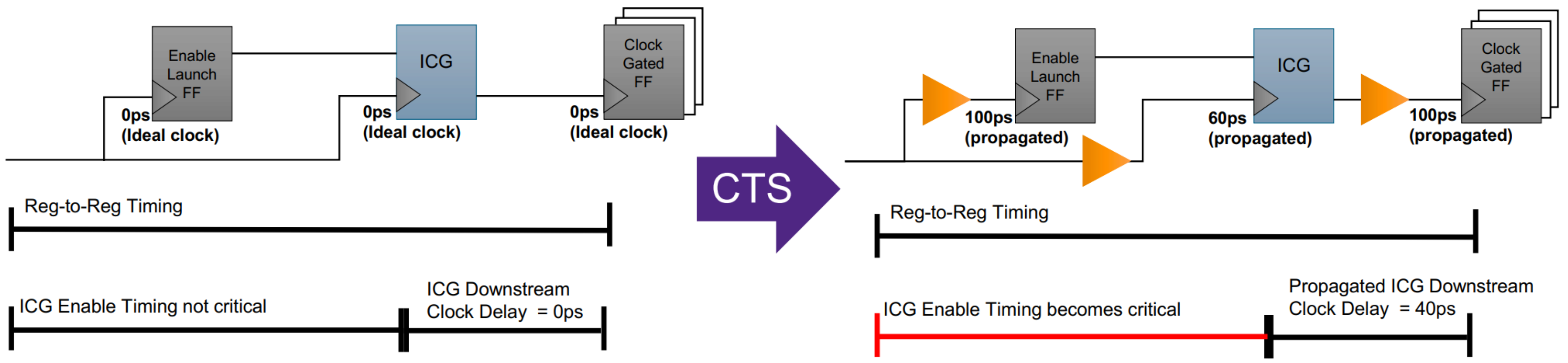


- This feature can improve design power along with timing QoR improvements
- Compile_fusion can estimate offsets saving area/power
- Clock build_clock push/pull clocks for better timing
- Clock final_opto/route_opt perform incremental CCD optimizations

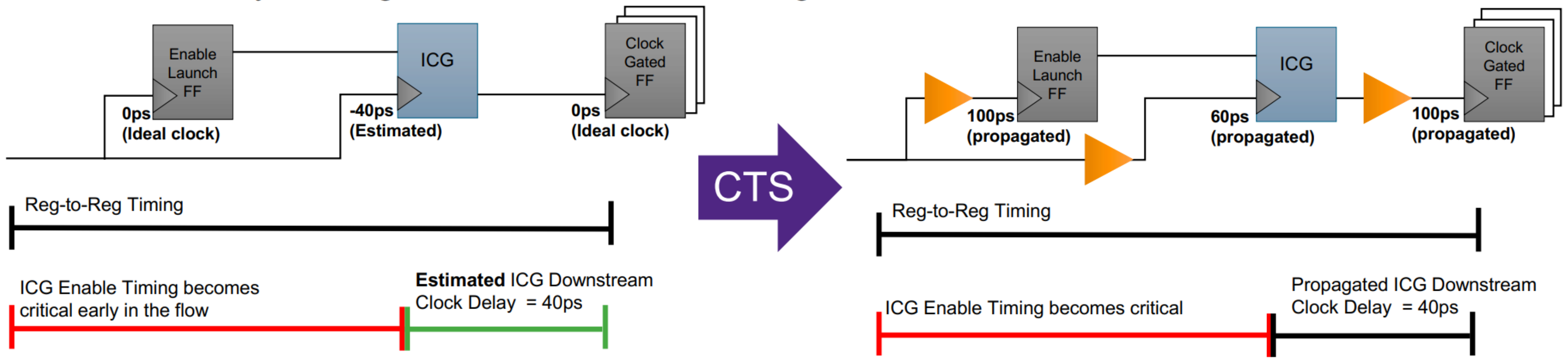
Source: Concurrent Clock and Data, SNUG 2018

ECGL

- At pre-CTS stages we don't have correct clock gating latencies:



- Tool can estimate ICG downstream clock delay after specifying set_multisource_clock_subtree_options and ensuring opt.common.estimate_clock_gate_latency is true (OBD)



Feature#2: Full Flow CCD (Recommendation)

- Many CCD app options to control flow are available
  - report_app_options *ccd*
- For a successful CCD implementation, consider:
  - If setup critical design: set_app_options -name ccd.fmax_optimization_effort -value high
  - CCD is path group oriented, make sure both ccd.skip_path_groups and ccd.targeted_ccd_path_groups are properly set
  - If CCD hold degradation is not acceptable, consider changing value of ccd.hold_control_effort