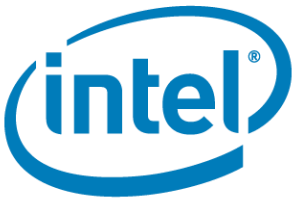


Chassis Clocking Verification Component User Guide

Synopsis:
This component should be used to validate an IPs clocking interface with the Chassis defined CCU and for generating clocks needed by or verification components. It is a System Verilog OVM component. The user is given control over the clock parameters, delays etc using configuarion objects as well as contrainted-random transactions. It also has a checker that checks the Chassis defined clocking protocols.

Contact: **Rajitha Ravindran**

Intel Corporation
CSG
Folsom

August 1, 2012

Legal Notice and Disclaimer

Document Location

Show the URL location of this document

Distribution

To receive component package updates and release notices, please send a request stating your need to:
rajitha.ravindran@intel.com

Acknowledgements

Revision History

Rev.	Date	Editor	Description
0	Aug 1st 2012	Rajitha Ravindran	CCU VC feature review
1	r120907	Rajitha	CCU VC 0.5-0.7 release
2	r121015	Rajitha	Review based changes
3	r121105	Rajitha	Bug fix clkreq/clkack default value should be 0
4	r121108	Rajitha	Bug fixes clkack driving was failing when multiple clkreqs asserted the same time
5	r121109	Rajitha	Added in_phase_with field to resolve the svc-pvc-ccu issue
6	r121113	Rajitha	Added dcg_blk_num to resolve the svc-pvc-ccu issue as a more robust solution. This should make the in_phase with redundant
7	r121115	Rajitha	Added IS_ACTIVE parameter in TI to resolve possible FC integration issues
8	r121120	Rajitha	Added IP_ENV_TO_AGT_PATH parameter in TI to resolve FC integration issues
9	r1301	Alamelu	Bug fixes

Rev.	Date	Editor	Description
10	r130219	Rajitha	Checker release CCU monitor removal CCU OB uniquification to avoid conflict GATE/UNGATE typo bug fix
11	r130315	Rajitha	Features release Added handshake delays that can be programmed From config object and test Udf/hdl update requested for doc reuse added
12	r130411	Rajitha	Features released Added ability for tests to enable random phase between clock slices Added coverage for ccu interface Added ability for FC to disable timeprecision Removed the ovm info printing configuration by default
13	r130419	Rajitha	Freq change delay feature added. This delay can be randomized more details can be found in section
14	r130503	Rajitha	Coverage handshake delay parameters added. Random phase enable made on by default Phase shifting was not working properly for gated clocks, fixed that Sip vintf dependency removed from code CCU OB and CCU CRG
15	r130521	Rajitha	Bug fixes Clk src switching changed the div ratio of the slice,fixed that The random phase enable default on feature was meddling with the phase relationship on clks within the same dcg block fixed this.
16	r130613	Rajitha	Bug fixes In a particular env the sv queues used within the code of the tracker started losing values – swapped them with arrays instead. Not sure if that was a simulator bug but changed it anyway The coverage req1 to clkack was coded in a way that it wasn't getting hit, Fixed this Enhanced the bfm to not try clk src switching if back to back same clk src is attempted from a random test
17	r130706	Rajitha	DCN changes https://vthsd.fm.intel.com/hsd/sunrise_point/#requirement/default.aspx?requirement_id=111045
18	r130830	Rajitha	Bug fixes - Simultaneous global reset and clkreq change handing

Rev.	Date	Editor	Description
			- Clkreq/ack handshake checks made async
19	r131008	Rajitha	DCN changes (usync) https://vthsd.intel.com/hsd/soc_chassis/issue/default.aspx?issue_id=1570784 Bug fixes Made clkack async https://vthsd.intel.com/hsd/seg_softip/bug/default.aspx?bug_id=5191246 clkgate delay issue https://vthsd.intel.com/hsd/seg_softip/bug/default.aspx?bug_id=5190780
20	r131216	Rajitha	Clkgating issue with the release r131008 fixed in this release
21	r140110	Rajitha	Performance updates - Internal always running clk within the checker removed as it was bringing down the performance at FC - Usync related tasks also made efficient so it doesn't remain as heavy on the performance. Enhancements - Randomization of handshake delays now can be done using just 4 config switches more explained in section 3.1.3 - Plusarg checking for maximum allowable time for clk to ungate added. Request was made keeping soc/fc clk trees in mind to make sure the clk ungating doesn't take very long - API added which can be used to return the freq of a given slice. Usage / example of this api given in test15
22	r140429	Rajitha	Release was made for chassis_reset_vc. IT requires visibility to the ccu interface so changed the scope of the interface declaration.
23	r141103	Rajitha	Enhancements - DUTY cycle can be programmed to other values than default 50. - CLK_SRC switching was enhanced to comprehend in_phase_with_slice_num field. Just changing the always running clk which all the other slices are in phase with

Rev.	Date	Editor	Description
			should now change frequencies for all slices that are listed as in phase with it.

Table of Contents

1. Introduction

This component should be used to validate an IPs compliance to clocking interface with the Chassis defined CCU. It is a System Verilog OVM component. The user is given control over the clock parameters, delays etc using configuarion objects as well as contrainted-random transactions. It also has a monitor that sends information about clock state and a checker that checks the Chassis defined clocking protocols (not implemented yet). The initial version of this component is based on SEG SIP teams CRG agent.

1.1 Terminology

Terminology	Meaning
CCU_VC	Chassis Clocking Unit – Verification Component
usync	Universal synchronizer pulse.
g_usync	Global Universal synchronizer pulse

This guide explains code examples delivered with Agents. Example code should be consulted, not the explanation, if they are different. Example code is regression tested. Guide updates are less frequent.

1.2 Tool Support

To file request on new features, report problems, raise issues, please take a minute to fill up the HSD form here :

Issue Reporting: *<HSD Link of this tool>*

- Unit Name : *Chassis – CCU_VC*
- Owner : rravindr

You can call or e-mail a support representative to fill out a ticket for you, but response time may be slower. See Section 21 for a guide to getting quick answers.

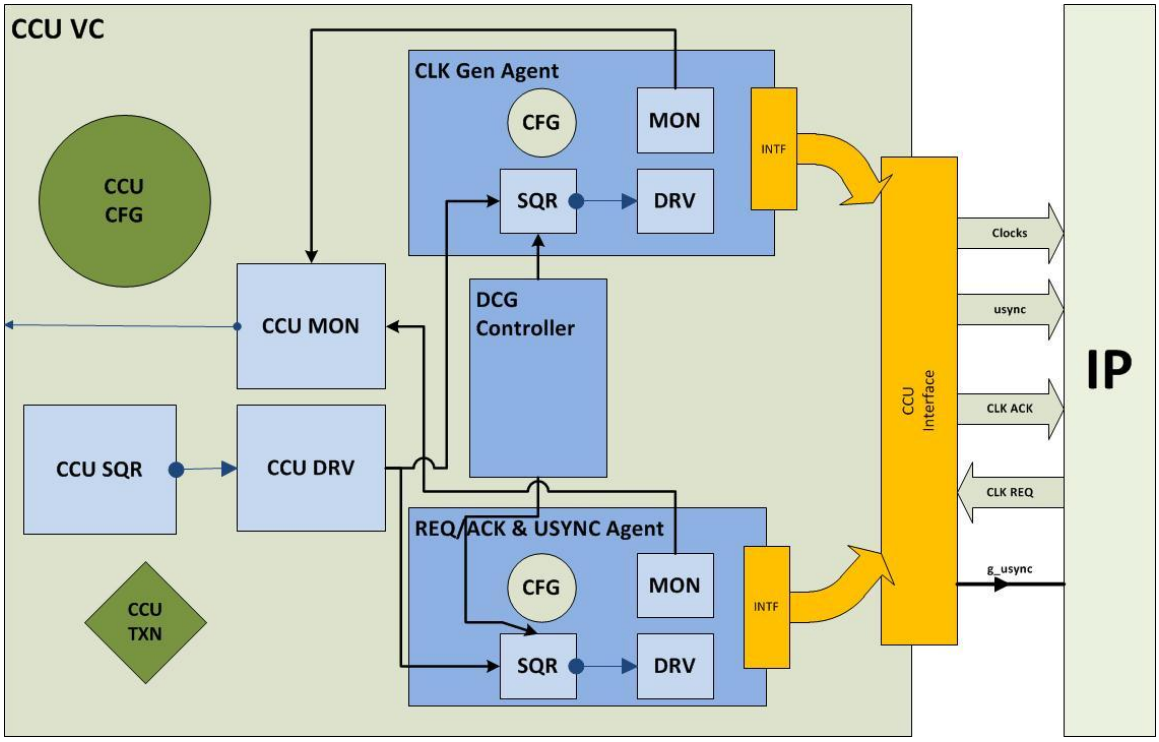
Support Contacts :

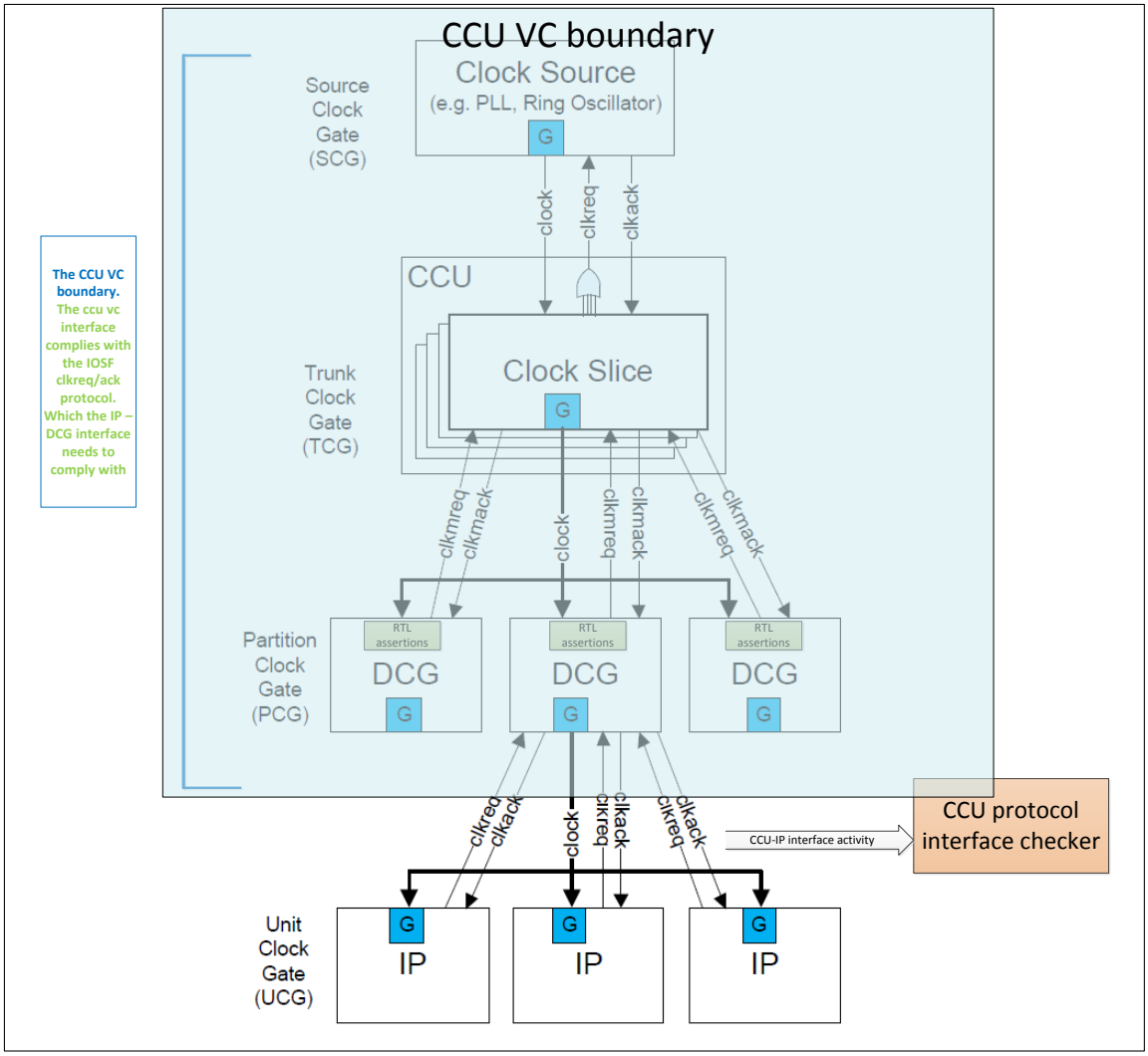
Role	Name	User ID	Location	E-Mail	Telephone Number
Primary Owner	Rajitha Ravindran	rravindr	FM	rajitha.ravindran@intel.com	
Secondary Owner					
Original Developer					
Manager					
Cluster/IP Lead					

2. Overview

CCU_VC generates clocks , usync and corresponding clk-req/ack handshake that can be configured using the controls mentioned in the next section. It is compliant to the Chassis Clocking Unit HAS. The checker document can be found at CCU Checker document

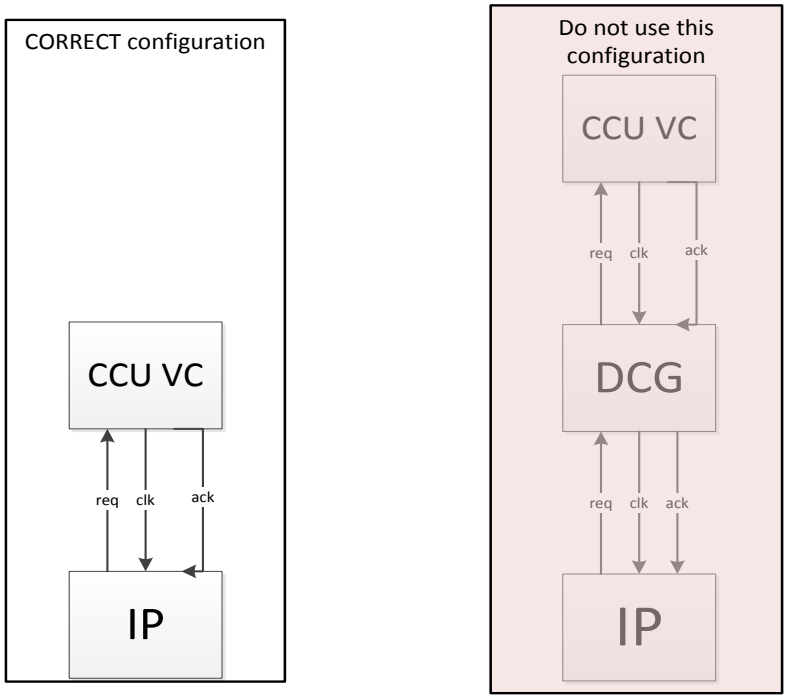
2.1 Architecture





The CCU VC interface complies with the IOSF clkreq/ack handshake rules. Every DCG downstream interface or IP interface is expected to be compliant with the same.
The above diagram also captures where the CCU checker would reside and how the other interfaces between DCG <-> CCU or DCG <-> DCG will be checked by RTL assertions within the DCG block provided by Chai Forng on IRR.

SPT CCU <-> IP
cfgs



IMP NOTE: DCG implementations may vary from project to project, and the upstream of the DCG block per spec is not required to comply with the 4-phase req-ack handshake. For this reason we would recommend you connect the CCU with the IP interface directly. This also makes sure that the ccu checker has visibility to the IP interface

2.1.1 CCU_VC Agent

It can be configured using configuration object and using transactions of type ccu_xaction sent from the test. The agents'sequencer sends transactions to the driver which converts then into pin level activity. The configuration object gets passed into the driver.

2.2 Applications

- Unit-level Testbench

- All the notes below for Cluster testbenches apply to ULT also.

- Cluster /Full-Chip Testbenches
  - At cluster level, the ccu_cfg can be used to validate an IP's compliance to the Chassis defined CCU interface.
  - At the FC level, the agent would be in passive mode since it would be replaced by the actual RTL. The monitor and checker would continue to operate.
    - Instantiate the CCU_VC in the test island so that it can be promoted.
    - Use the is_active configurable variable in the agent to make it passive.

NOTE: Use the CCU_VC only if the the purpose is to generate /check Chassis CCU compliancy for the clocks in your environment.
Would not suggest using the CCU_VC to generate clocks which have no requirement to be chassis compliant

2.3 Features

The bfm features discussed below are based on the Chassis Clocking Unit HAS.
The checker document can be found at CCU Checker document

- **Clock generation** (clock source,slice) details in section 3.1.3 later in this document

CCU_VC supports basic clock generation using the ccu_cfg.

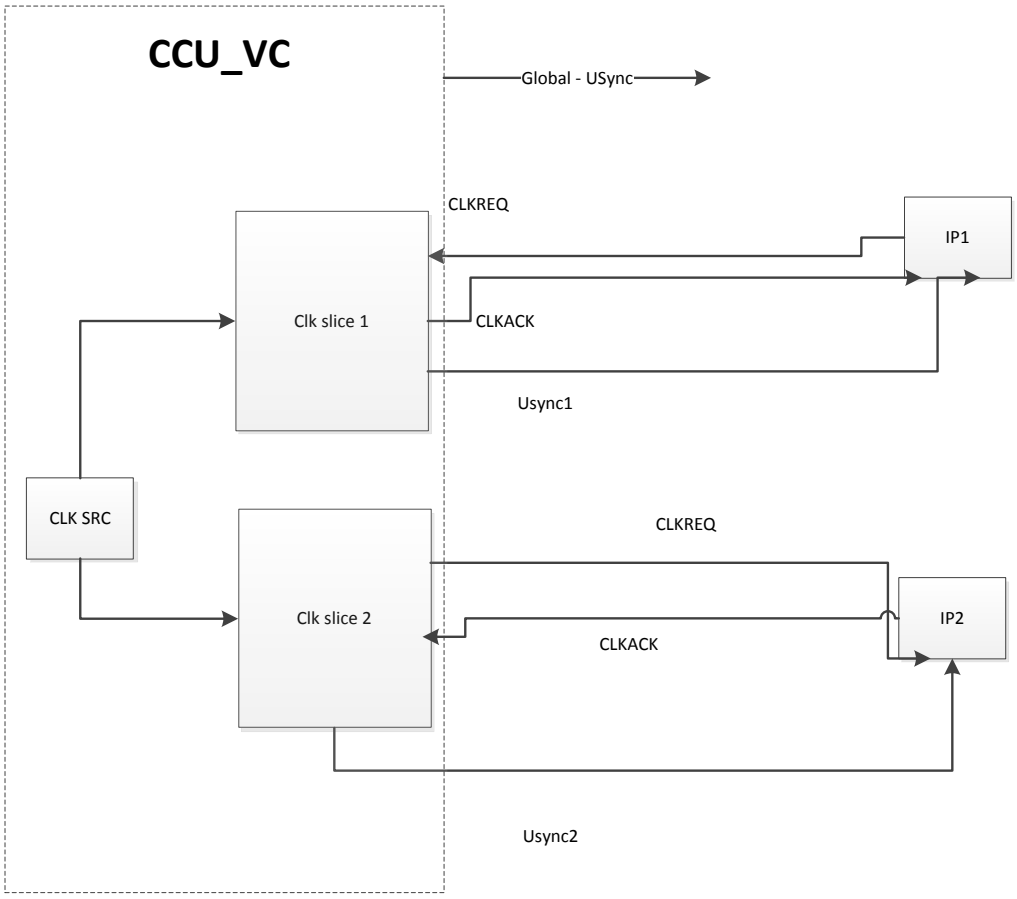
```
cu_vc_cfg_instance. add_clk_source (
                int clk_num ,
                string clk_name,
                time period
               );
```

You can also add clock slices

```
ccu_vc_cfg_instance.add_slice (
                int slice_num ,
                string slice_name,
                int clk_src = 0,
                ccu_types::clk_gate_e clk_status = ccu_types::CLK_UNGATED,
                ccu_types::def_status_e def_status = ccu_types:DEF_OFF,
                ccu_types::div_ratio_e divide_ratio = ccu_types::DIV_1,
                int half_divide_ratio = 0,
                int dcg_blk_num = 1024,
                int in_phase_with_slice_num = 1024,
                time req1_to_clk1 = 0,
                int clk1_to_ack1 = 0,
                int req0_to_ack0 = 0,
                int clkack_delay = 8,
                bit always_running = 0,
                bit set_zero_delay = 0,
                bit enable_random_phase = 0,
                int duty_cycle = 50,
                time freq_change_delay = -1,
                bit randomize_req1_to_clk1 = 0,
                bit randomize_clk1_to_ack1 = 0,
                bit randomize  req0_to_ack0 = 0,
                bit randomize_clkack_delay = 0,
                bit usync_enabled = 0
```

);

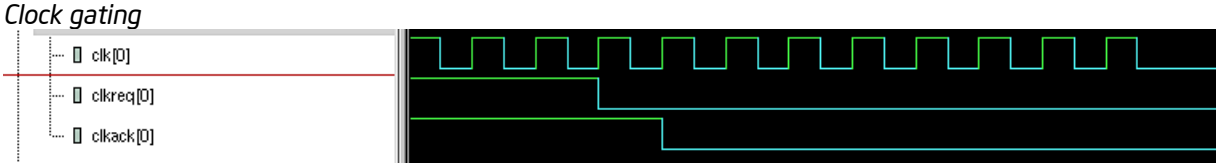
- **Handshake**
  - The CCU_VC also supports the clkreq/clkack handshake so that needs to be hooked up to the IPs clkreq appropriately, as shown in the example below,
    Please note that the bfm will only generate one clkack perl clk-slice, .Each slice should be dedicated to one IP clk only to assure there is a dedicated clk_ack for it.If you have an always running clock in your environment with no clk_req/clk_ack associated with it, you can leave the clk_ack hanging and tie off the clk_req to 1.



- **Clock Gating**

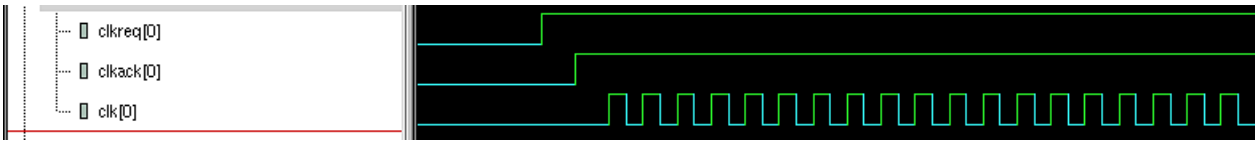
CCU_VC supports clock gating , either in the config or from the test details in 3.1.3

Clock gating



As per the Chassis CCU 0.7 spec, the clk needs to remain alive for atleast 8 clocks after the clk_ack -> 0. That is the hysteresis , it could be more than 8 clks.

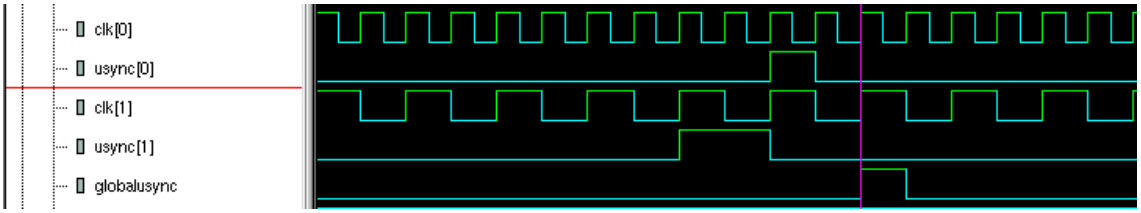
Clock ungating



BFM Issue clk ack needs to deassert after clocks are active

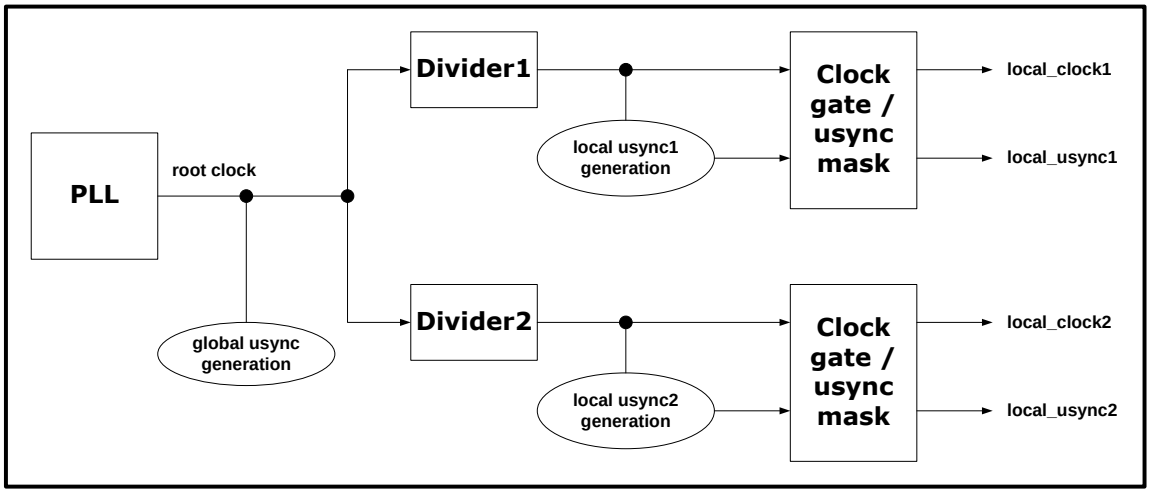
- **Usync/ Global Usync Generation**

You can enable disable usync for each clk slice
If Usync is enabled, a global usync will neet to be generated by the bfm



Usync is associated with each clock slice is the local usync , and global usync is the universal clock edge for all the clock slices that are derived from the same clk source.

High level example of usync generation



Above diagram is from the Chassis Usync ECN
https://vthsd.intel.com/hsd/soc_chassis/default.aspx#issue/default.aspx?issue_id=1570784

Steps to generate Usync from the CCU VC

- Set one of your clock sources as the ref_clk source (root clk) using api ccu_vc_cfg.set_ref_clk_src(0)
- Set global usync counter. Based on lcm calculation of all clk frequencies in your env you should get the number of ref clocks after which the GAL (universal/global usync) edge should be seen. Use this api ccu_vc_cfg.set_global_usync_counter(16'h5F)
- Add slices in your env that are derivates of this ref clk using api ccu_vc_cfg.add_slice(.slice_num(1), .clk_src(0)....)
- Start the gusync counter, usinc api i_ccu_vc_cfg.ctrl_gusync_cnt(1)

You can refer to test02 which generates usync for different freq clks. It uses the frequencies from BXT as an example.

CCU VC Rules/Assumptions on usync

- When usync is enabled all clk slices that are derived from the same clk source should be in sync with each other. There should not be a random phase shift / skew between these clks.

Integrators need to make sure that the random_phase_en is set to 0

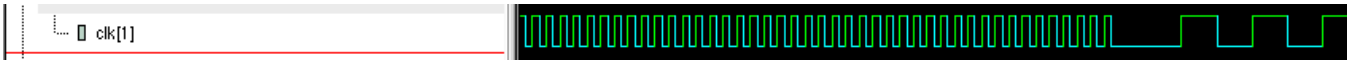
- GAL/Universal clk edge is when all clock slices from same clk source have their posedge aligned
- Usync pulses are always only one associated slice clk wide
- Default value of usync is 1
- Usync will be synchronous to the slice clk it is generated for
- Usync pulses are generated 2 slice clk edges before the GAL(universal clk) edge.
- Usync pulse only gets generated for clks when they are ungated.
- Usync only applies to all clocks which share the same clock source.
- Currently the bfm supports just one global usync , and one root (ref_clk) if you have a project which may require multiple please raise it.

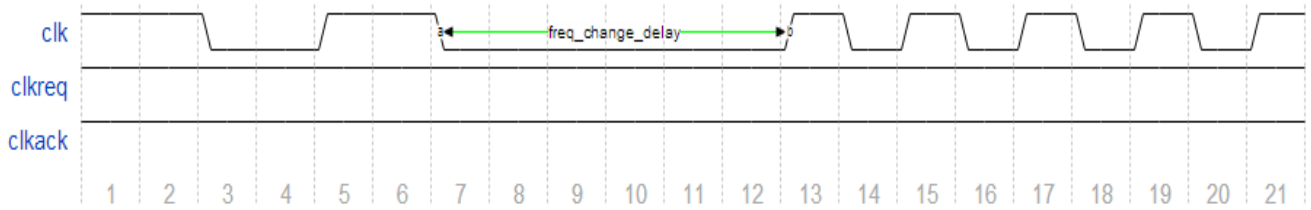
The current support on usync on the ccu vc and the Chassis spec may not be covering determinism. Which means it will not gate/ungate clk or change freq on usync boundaries. There may be a subsequent Chassis DCN which explains the details on determinism and usync.

- **Frequency Change**

CCU_VC supports changing the frequency of the clock , from the test using commands. Check 3.1.3

Examples of Frequency change (assuming usync is disabled for these clocks)





- **Clock Divider**

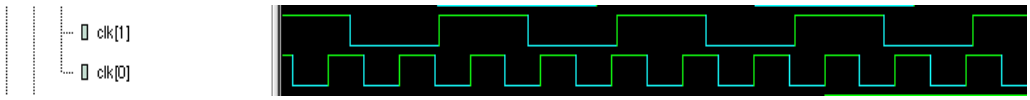
Programming divider – Program each slice to have a dividing ratio anywhere between 1 to 16.
The bfm divides the clock frequency to generate a slower clock.
Below table shows the divide ratio and the corresponding change in the frequency
Post divide ratio – CCU 0.7 (divide upto 256)
If you don't have usync, you can go ahead and program slices directly with DIV raito always 1

divide_ratio	half_divide_ratio	Actual div ratio seen at IP	Resulting clk frequency with an example of 1600Mhz
DIV_1	0	1	1600
DIV_1	1	1.5	1066.666667
DIV_2	0	2	800
DIV_2	1	2.5	640
DIV_3	0	3	533.3333333
DIV_3	1	3.5	457.1428571
DIV_4	0	4	400
DIV_4	1	4.5	355.5555556
DIV_5	0	5	320
DIV_5	1	5.5	290.9090909
DIV_6	0	6	266.6666667
DIV_6	1	6.5	246.1538462
DIV_7	0	7	228.5714286
DIV_7	1	7.5	213.3333333
DIV_8	0	8	200
DIV_8	1	8.5	188.2352941
DIV_9	0	9	177.7777778
DIV_9	1	9.5	168.4210526
DIV_10	0	10	160
DIV_10	1	10.5	152.3809524
DIV_11	0	11	145.4545455
DIV_11	1	11.5	139.1304348
DIV_12	0	12	133.3333333
DIV_12	1	12.5	128
DIV_13	0	13	123.0769231
DIV_13	1	13.5	118.5185185
DIV_14	0	14	114.2857143
DIV_14	1	14.5	110.3448276

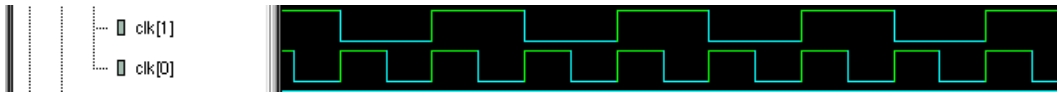
DIV_15	0	15	106.6666667
DIV_15	1	15.5	103.2258065
DIV_16	0	16	100

In both the examples below **clk0** is the source clock frequency and **clk1** is the divided clock from the slice.

Half – clk divider example – clk divider ratio 2.5



CLK divider ratio 2

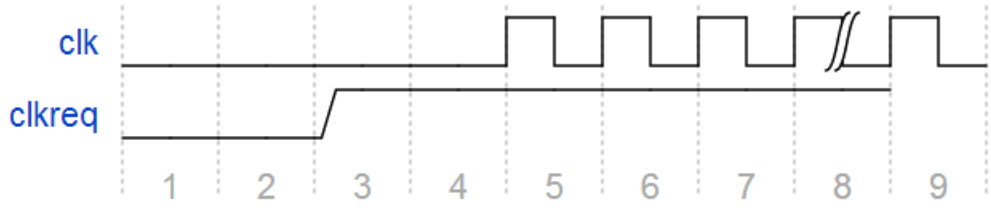


- **Programmable Hysterisis for clock gating**
  Based on the 0.7 spec, the hysteresis of keeping the clock alive for atleast 8 clks requirement will be added to the bfm. But it can be programmed to a value higher than 8, this feature will be added in the bfm, Currently it randomizes the hysteresis to a value above 8.

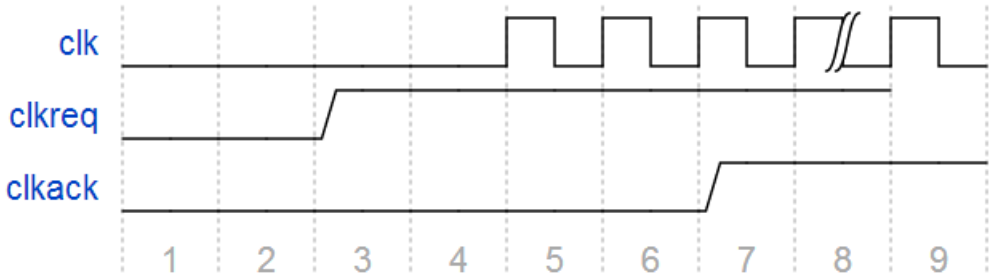
- **Phase shift / Skew**
  Added ability in the config object so you can program random phase between different clock slices. This release r130412 has the default for this feature as 0. But the subsequent release will have this feature enabled by default. Need all Ips to validate if their DUT can handle skewed clocks. Note that the clocks coming from same dcg blocks will not be out of phase.

- **Programmable handshake req-ack-clkrunning delays**
  req1_to_clk1 – clkreq 1 tp clk running in time units
  clk1_to_ack1 – clk running to clkack 1 – min 1 clk to any number max – in clk units
  req0_to_ack0 – clkreq 0 to clkack 0 – min 0 to any number max – in clk units
  clkack_delay – clkack 0 to clock gating – min 8 to max any number – in clk units

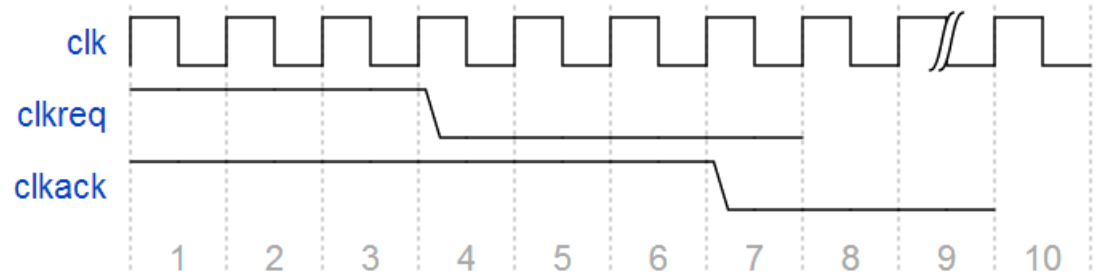
req1_to_clk1



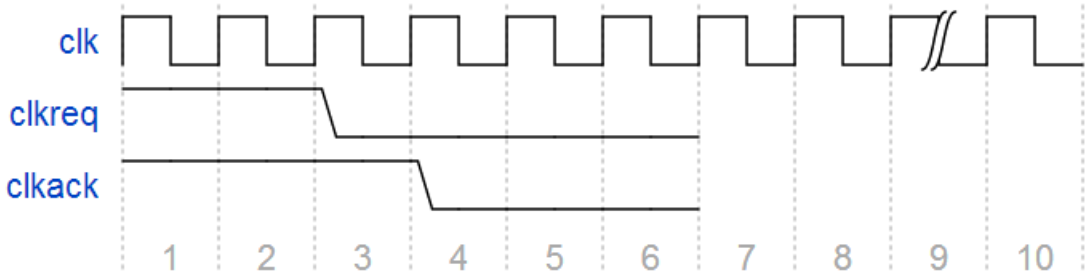
clk1_to_ack1 [Chassis spec restricts this to be atleast 1 clk]



req0_to_ack0



clkack_delay [Chassis spec restricts this to be atleast 8 clks]

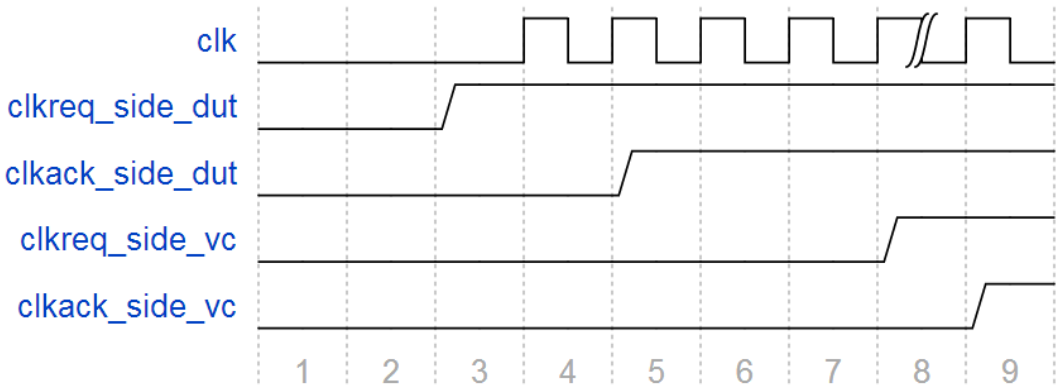


These delays are statically programmable from the config object and you can also change these at run time using commands described in section 3.1.3.2 CCU transaction

IMPORTANT NOTE on handshake delays :

- When you program these delays to slices which belong to the same dcg block number, please note that you will see the delay associated with the slice whose handshake event happens first.

  Example



In this case, whatever the req1_to_clk1 delay value was programmed on the clkreq_side_dut slice will be the one you will see, the second clkreq from the side vc asserts later and although you have programmed a delay value programmed there, you will not see it on the waves.
The same thing will happen when you see the clkgate, the clkack -> 0 happening later of the 2 will be the one that decides the number of clocks the clk needs to stay active for.
You could program the slices belonging to the same dcg block with the same handshake delay if you find this confusing.

- Constraints on delays, there are some default constraints on the delays which will take effect if you do not specify a value for the handshake delays. Please test09 for examples on that
- You can look at test10 for examples on specifying your own range for a particular delay value from the test.

- **DCG block number**
  - Any number of slices can be added to the same dcg_blk_num and their output clock would behave the same way
  - Clocks belonging to the same dcg blk will only gate if all the clkreqs in that dcg blk are 0
  - Will ungate as long as atleast one clkreq is 1
  - Freq change issued on one will imply freq change on all the slices in the same dcg block
  - Clksrc change on one will imply clksrc change on all in the same dcg block
  - The agent constraints the slices belonging to the same dcg block to have the same clk source and the same frequency,
  - Test writers will get a solver error, if they try to configure otherwise

- **Early boot side rst [Global reset] impact**

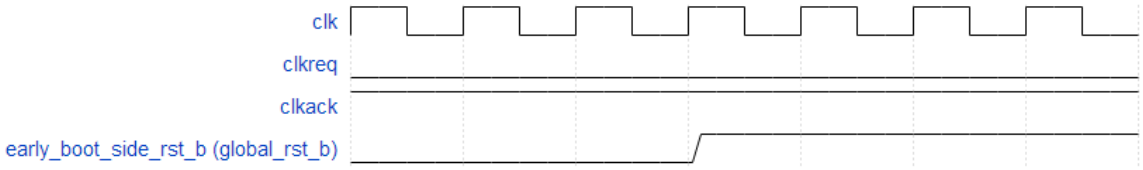
The ccu bfm will sample clkreq only after reset deasserts, So for a default off slice the clock will not start toggling till reset deasserts even if clkreq is asserted before the reset. And similarly for a default on slice the clock will start toggling everytime the reset is asserted and clkack will be driven to high. Explaining the expected behavior from the ccu vc for DEF_ON and DEF_OFF cases. The expected clkreq during reset behavior is also explained later in this section.

Default ON

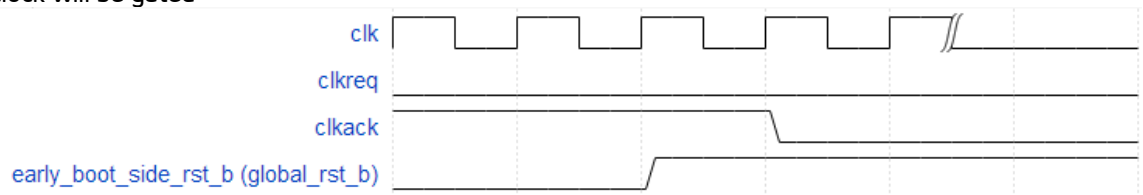
Clkack asserted and clock running when reset is asserted irrespective of clkreq

Coming out of reset

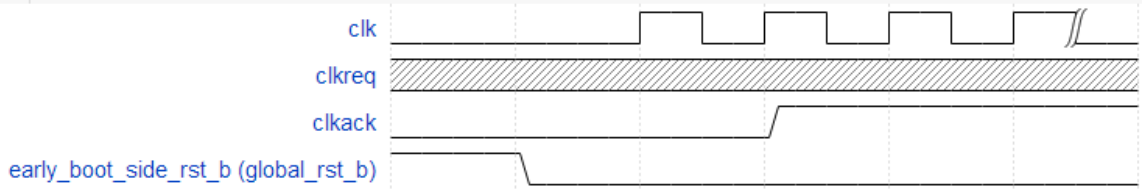
Showing clkreq as 0, but the clock will be running irrespective of clkreq value



Clkreq will be sampled only after reset deasserts, so if it is 0 the clkack will be deasserted and clock will be gated



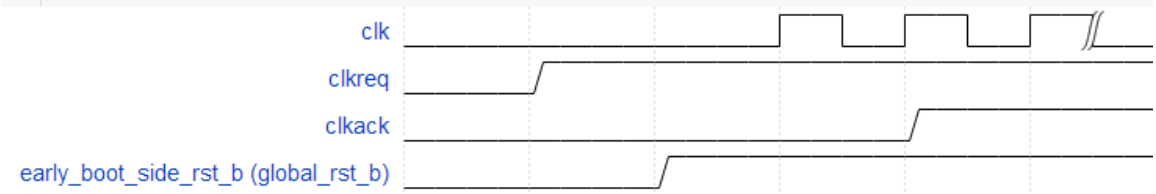
Going into reset



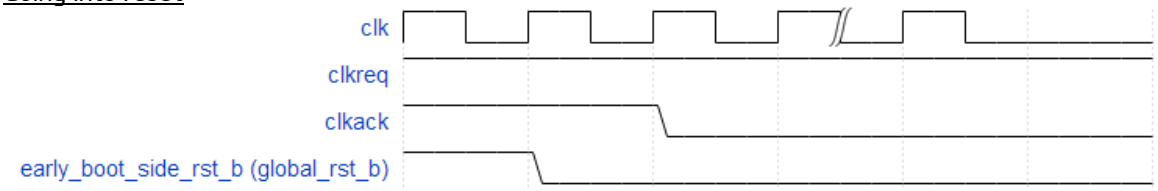
Default OFF

Clkack deasserted and clock gated when reset is asserted irrespective of clkreq.
Clkreq changes when reset is asserted will be ignored

Coming out of reset



Going into reset



Expected behavior on clkreq based on the SPT Chassis Clocking DCN

The changes made to the ccu vc are drive based on this DCN
https://vthsd.fm.intel.com/hsd/sunrise_point/#requirement/default.aspx?requirement_id=111045

Figure 1: DCG Defaulting to ON, Connected to Early Boot IP Block or PMC

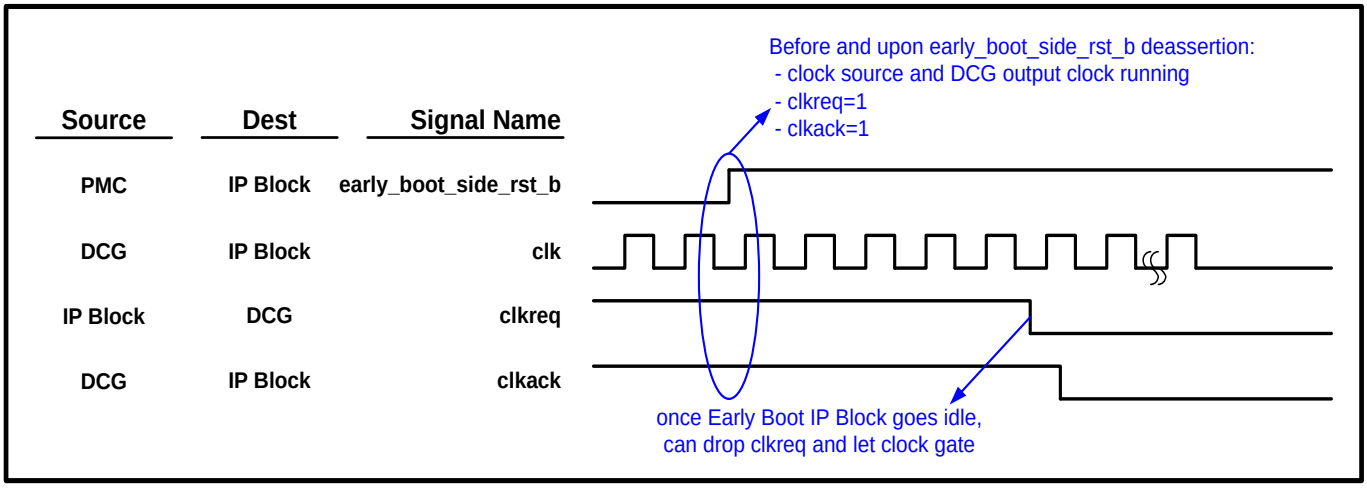


Figure 2: DCG Defaulting to ON, Connected to <group> IP Block

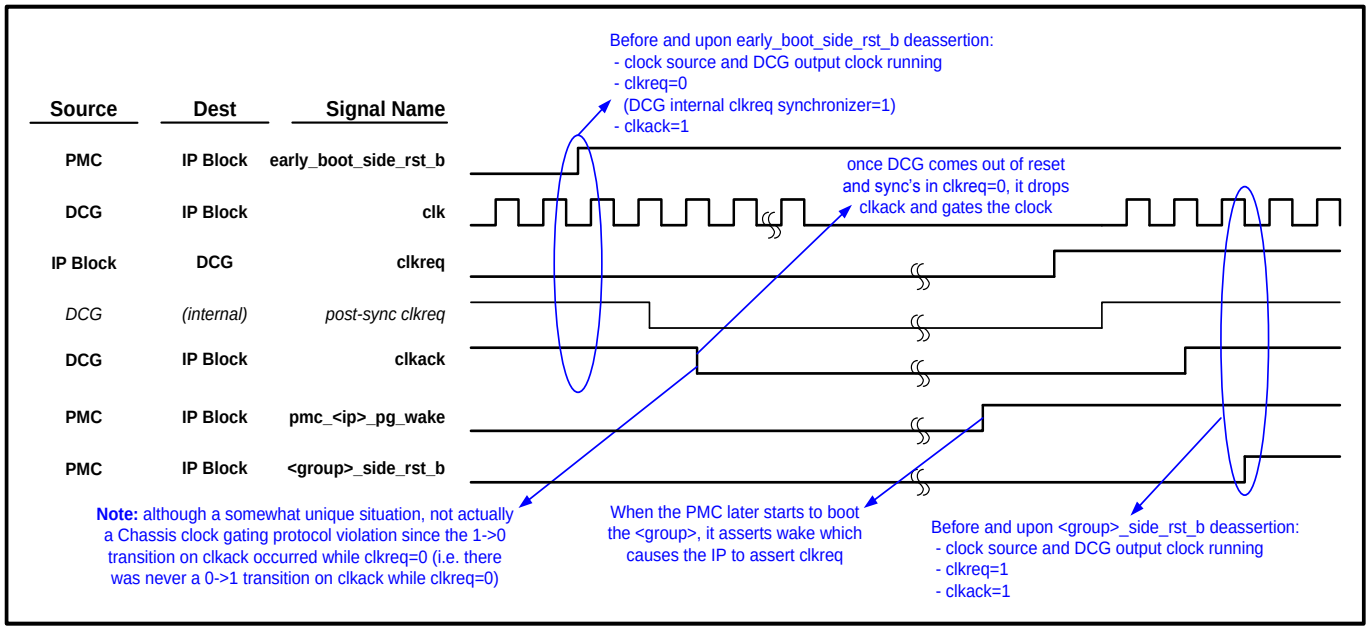


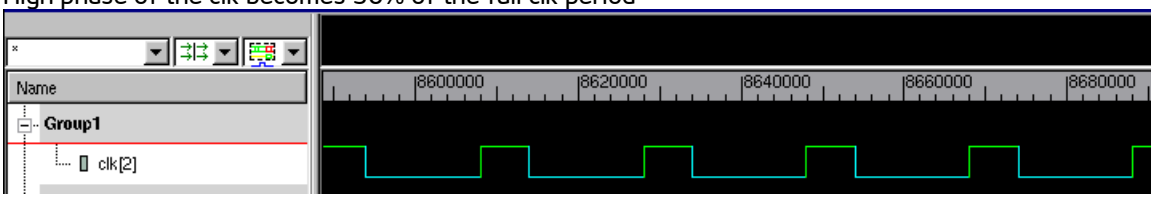
Table: Showing the default configuration for different clocks in SPT

Clock	Default Staus
RTC clock	DEF_ON
Rosc <speed> clock	DEF_ON
Side clock	DEF_ON

Prim Clock	DEF_ON
Any ROSC based clock	DEF_ON
Xosc clock	DEF_OFF
Any PLL based clock	DEF_OFF

- **Configurable DUTY cycle**
  Added capability in the config object to understand a different duty cycle than 50. Till now CCU VC was restricting the clk duty cycle to always be 50 now that is not true anymore. However we still don't support clk duty cycle changing on the fly so the config object which can be changed at the build phase only has the capability to specify a different duty cycle.

Example of clk with duty cycle set to – 30
High phase of the clk becomes 30% of the full clk period



- **Programmable usync assert/deassert times**
  0.7 spec mentions that the usync assert/deassert times can be programmable, some opens are still there related to that. So this feature is not yet added will add it once we get more clarity.Will be a part of 0.8 bfm release

- **Freq Change, clock gating at usync boundaries**
  Currently the bfm doesn't wait for a usync boundary to change the clock frequency or to do clock gating. This feature needs some clarification, will be implemented by the bfm 0.8 release

2.4 Control

This verification component has three basic levels of control:

- Parameters
  - Parameters are used to set the number of clocks.
  - Set the agent to passive
  - IP_ENV_TO_AGT_PATH to scope all the set_configs within the agent instance
- Configuration objects
  - Configuration object ccu_vc is used to configure the clock source/slice behavior.
- Transactions
  - Transactions are sent by test during runtime to change frequency, gate/ungate/divide the clock etc using constrained random xactions.

In general, parameters and configuarion objects are set once per testbench, configuration methods are set once per test, and transactions are set many times during a test

2.5 Requirements

2.5.1 Specifications and Reference

Please go through the following before you work with this BFM

Chassis Clocking HAS – this is the Chassis spec that details the CCU 0.7 HAS

System Verilog (SV) Language Reference Manual – This document contains System Verilog concepts, syntax and example of System Verilog code usage. This user guide will have examples of System Verilog code and it's useful for the user to understand the System Verilog code. The document location is:
//avc/models/SPXBPkg/DOC/SystemVerilog_3.1a.pd

2.5.2 Compute Environment

In order to use the package the following tools, software and operating systems are required.

- Simulators the package can be used with.
  - vcsmx F-2011.12-3
  - vggnu 2011.12
  - ace 2.01.02
  - ovm 2.1.1_1p1
  - saola v20121203
  - Verdi -3 2012.10
- The user needs access to AVC Model Disk where the BFM is located. The BFM is located for download at: /avc/vip/CCU_VC
- It also uploaded in IRR and can be downloaded from there –
- Test plan to check the bfm

2.5.3 System Verilog Packages

ovm_pkg – System Verilog base framework
saola

3. Getting Started

3.1 Setting Up a Testbench Environment

This section describes how to setup and configure OVM envirnonments for this Agent. The previous section explained Agent package installation prerequisite tools.This section focuses on how to connect Agents to a testbench, with example codes.

The section after this one describes how configure Agents and implement test scenarios.

3.1.1 Test Island

Following are details of interfaces that are to be connected at the SoC level. There is only one interface named ccu_intf located in $IP_ROOT/verif/ccu-vc/ccu_intf.sv. The user is expected to instantiate this interface, connect the various signals to his IP and pass the interface to the ccu_ti (located in $IP_ROOT/ccu-vc/ccu_ti.sv) instance.

Signal	Connect to	Description
Clk[]	Clk wires that should be driven by CCU	Each bit of this bus corresponds to a slice. The clk bus is indexed by the slice numbers. For example, clk[0] should be connected to the clk wire in the IP that is to be connected to slice 0 of CCU
Clkreq[]	Clkreq of the different Ips/fabrics	This is a bus of clkreqs coming into the CCU. It is indexed by slice number too
Clkack[]	Clkack of the different IP/fabrics	Clkacks driven by the CCU in response to clkreq. Indexed by slice number
Usync[]	Usync of each IP	Usync driven by the CCU to each IP that needs usync. Indexed by the slice number
Globalusync	Globalusync of the IP	It is a 1-bit signal that is driven by the CCU and has the global usync pulse generated as per configuration
Reset[]	Reset associated with each clock	This is an input to the ccu vc and should be connected to whatever is reset is associated with that clk. There will be assertions written around it soon

Signal	Connect to	Description
Pwell_pok[]	Power well OK , which contains the clkreq/clkack driving logic	Whatever power well houses the clkreq driving logic needs to be connected here. When the simulations will run with UPF, the clkreqs/clkack would become X causing false failures. Will be disabling the assertions based on this power well ok signal. For SPT all the driving logic should be in the primary well so int_pwell_pok needs to be connected to them all.
Global_rst_b[]	Global reset	There is only one global reset in SPT , kept an array since there could be collage related interface splitting changes coming soon, and other projects which may have a different reset.

3.1.2 IP Environment

The CCU VC environment consists of a few components as described in the following table.

Component Name	Description	File location
ccu_vc	Top level component that is to be instantiated by the user in his environment	verif/ccu-vc/ccu_vc.svh
ccu_vc_cfg	VC configuration class that is used to configure the VC to function according to user requiements	verif/ccu-vc/ccu_vc_cfg.svh
ccu_seqr	VC Sequencer that the user will use to run sequences and transactions	verif/ccu-vc/ccu_seqr.svh
ccu_driver	VC driver that is used to drive and translate CCU transactions to the appropriate components	verif/ccu-vc/ccu_driver.svh
clk_gen	Clock generation agent used to generate clocks to the different slices according to configuration	verif/lib/CRG
req_ack_usync_gen	Generator that handles clkreq/clkack in addition to all slice usync signals and the global usync but when told to	verif/lib/CCU_OOB
dcg_controller	Controller that is responsible for managing all clkreq/clkack protocol for all slices	verif/ccu-vc/dcg_controller.svh
ccu_xaction	Main transaction that allows user to control CCU functionality during run-time	verif/ccu-vc/ccu_xaction.svh

Component Name	Description	File location
ccu_intf	SV Interface that holds all the signals to be connected to the IP	verif/ccu-vc/ccu_intf.sv
ccu_vc_ti	Test Island that should be instantiated in user's top level module and passed a ccu_intf instance	verif/ccu-vc/ccu_vc_ti.sv
ccu_vc_checker	The ccu checker contains assertions to verify all clocking spec rules	verif/ccu-vc/ccu_vc_checker.svh

3.1.3 Configuring the IP Environment

3.1.3.1.1 Compile-time parameters

Parameter	Default Value	Allowed Values	Description
NUM_SLICES	1	1 to 128	Number of slices that the CCU VC is supposed to instantiate
IS_ACTIVE	1	0 or 1	1 is the default value for this parameter which means active, to set the agent to passive this should be set to 0
IP_ENV_TO_AGT_PATH	""	"*IP_ENV.Agent_name"	This parameter is to ensure you don't have collision when using multiple ccu instances in your env or when promoting to FC. **PLEASE DO NOT make this parameter just "*". For a single instance your IP val wont flag, but FC will break**
REQ1_CLK1_MIN	100ps	Time unit value in ps	Min time in ps from clkreq-> 1 to clk ungating delay that should be covered by Ips in clt validation
REQ1_CLK1_MAX	1000ps	Time unit valye in ps	Max time in ps from clkreq-> 1 to clk ungating delay that should be covered by Ips in clt validation
CLK1_ACK1_MIN	2	Int value (Number of clocks)	Min clocks from clk ungating to clkack asserting that should be covered by Ips
CLK1_ACK1_MAX	20	Int value (Number of clocks)	Max clocks from clk ungating to clkack asserting that should be covered by Ips
REQ0_ACK0_MIN	2	Int value (Number of clocks)	Min clocks from clkreq -> 0 to clkack ->0 that should be covered by Ips in clt validation
REQ0_ACK0_MAX	20	Int value (Number of clocks)	Max clocks from clkreq -> 0 to clkack ->0 that should be covered by Ips in clt validation
ACK0_CLK0_MIN	8	Int value (Number of clocks)	CP will use this as the min number of clocks that the clocks should stay active after clkack -> 0.

Parameter	Default Value	Allowed Values	Description
ACK0_CLK0_MAX	50	Int value (Number of clocks)	CP will use this as the max number of clocks that the clocks should stay active after clkack -> 0.

3.1.3.1.2 Command line variable

Parameter	Default Value	Allowed Values	Description
MAX_UNGATE_TIME	null	<value>ns	This it the command line parameter which can be specified like this -simv_args +MAX_CLK_UNGATE_TIME=500ps This activates an assertion in the checker that is meant to catch cases where the clk ungating takes longer than expected. Example: ace -x -t test15 -sd gui -simv_args +MAX_CLK_UNGATE_TIME=500ns

3.1.3.1.3 Configuration-time variables

Parameter	Default Value	Allowed Values	Description
clk_sources	null	Up to 16 clock sources	These are the clock sources input to the CCU. They resemble the different PLLs in a system. An associative array of clksrc_cfg objects indexed by clk_num (int). Available fields are: int clk_num string clk_name time period

Parameter	Default Value	Allowed Values	Description
Slices – check if can be config	null	NUM_SLICES	This is the initial configuration of each slice in the system. It is an associative array ofslice_cfg objects indexed by an int representing the slice number. Available fields are: int slice_num; string slice_name; int clk_src; ccu_types::clk_gate_e clk_status; ccu_types::div_ratio_e divide_ratio; bit usync_enabled;
gusync_ref_clk	0	1 to 16	An int that selects which clock source is used as the reference clock to the CCU and hence used to drive the global usync
gusync_counter	50	1 to 2**16	The global usync counter that is used to count down and then generate a global usync pulse synchronized to the reference clock
is_active	OVM_ACTIVE	OVM_ACTIVE OVM_PASSIVE	Specify if the VC is used in active or passive mode

3.1.3.1.4 APIs

Function name	Arguments	Returns	Description
set_to_passive	none	void	Changes the VC to work in passive mode
get_num_slices	none	int	Returns the number of slices as configured in the compile-time parameter NUM_SLICES
add_clk_source	int clk_num string clk_name time period	bit	Adds a specific clock source to the clock sources table. User can only add up to 16 clock sources. Returns 1 on success and 0 on failure
randomize_clk_sources	none	bit	Randomly configures the 16 clock sources to random clock periods. Clock frequencies are in the range of 20KHz to 5GHz. Using this method after calling add_clk_source will override the clocks user already added

Function name	Arguments	Returns	Description
add_slice	Int slice_num String slice_name Int clk_src Clk_status Divide_ratio Half_divide_ratio Usync_enabled	bit	This API configures the slices in the CCU VC. It can be called up to NUM_SLICES times, after that it will fail and return 0. It takes a slice number and name. It sets the slices clock source to one of the clock sources configured in the clock sources table. It sets the default status of the clock (GATED/UNGATED) and the default divide ratio to be used. It also sets the default setting for usync generation. All these except the clice num abd name can be changed later during run time using ccu_xaction
set_ref_clk_src	int ref_clk_src	void	This API sets the reference clock to one of the clocks defined in the clock sources table. This must be called before the run phase and cannot be called once the run phase starts.add-for gusync purpose
set_global_usync_counter	bit[15:0] count	void	This API sets the value of global usync counter. Chagnes to the value will take effect only when the current count reaches 0
ctrl_gusync_cnt	Bit [0:0] ctrl_start_gusync	void	This api allows user to start the global usync counter. Some env require specific edge at which the global usync count should start. This is to give that control to user
set_sync_clkack	Bit[0:0] ctrl_sync_clkack	void	This api allows user to program the clkack generated by the VC to be synchronous. By default the clkack will be async.
get_clk_period	Int slice number	time	This API when invoked the user to gets the clk period of a given slice Example in test15

3.1.3.2 CCU Transaction and Config Object fields

The main transaction that can be used in the user's sequences to control the CCU functionality is the ccu_xaction. Here is a list of fields that are available in the transaction and possible values.

Some of the fields which need not change at run time, are available in the config object field only

Field	Description	Possible values
slice_num	Defines which slice the operation will be performed on	0 to NUM_SLICE-1
cmd	What is the operation to be performed	GATE, UNGATE, DIVIDE_RATIO, HALF_DIV_RATIO, CLK_SRC, EN_USYNC, DIS_USYNC, PHASE_SHIFT, CLKACK_DLY, REQ1_TO_CLK1, REQ0_TO_ACK0, CLK1_TO_ACK1, FREQ_CHANGE_DLY
clk_src	Defines the new clock source to be applied to this slice. Applicable only when cmd == CLK_SRC	0 to 15 default = 0
clk_status	Clock running from the start	CLK_GATED CLK_UNGATED Default = CLK_GATED
def_status	Default (in reset) clock and clk ack behavior	DEF_ON DEF_OFF Default = DEF_OFF
div_ratio	Defines the divide ratio to be applied to post divider of the slice. Applicable when cmd == DIVIDE_RATIO	DIV_1 --- DIV_16 default = DIV_1
half_divide_ratio	If this divide ratio is set to 1, then your final clk divider ratio would be (div_ratio).5 and if it is 0 then your final clk divider ratio would be same as div_ratio You can issue a change in frequency from the test by changing this : cmd HALF_DIV_RATIO	0 or 1 default = 0
in_phase_with_slice_num	This has been added to enable the test writer to program a gated and ungated version of the same clock or program 2 clock slices to be in sync.	0..NUM_SLICES default = 0

Field	Description	Possible values
dcg_blk_num	Any number of slices can be added to the same dcg_blk_num and their output clock would behave the same way Clocks belonging to the same dcg blk will only gate if all the clkreqs in that dcg blk are 0 Will ungate as long as atleast one clkreq is 1 Freq change issued on one will imply freq change on all the slices in the same dcg block Clksrc change on one will imply clksrc change on all in the same dcg block The agent constraints the slices belonging to the same dcg block to have the same clk source and the same frequency, Test writers will get a solver error, if they try to configure otherwise NOTE: This is a config bit field only it is not valid to want to change this from the test at run time	int
set_zero_delay	Will ensure all the following delays are 0, since that would end up violating the req-ack protocol we will disable checking for that slice clkreq 1 -> clkrunning clkrunning -> clkack 1 clkreq 0 -> clk ack 0 This could also be used for a bfm that cannot handle gated clock (mostly the always running and zero delay could be set together for bfm connections)	1 or 0 default = 0
always_running	always_running will ensure that slice gets an always running clock, the clkreq drops that slice's clkreq will be ignored If this slice is in the same dcg block as the DUT, a clkreq -> 0 on it will be able to gate the DUT's clock This could be used for a bfm , which cannot handle gated clock	1 or 0 Default 0
req1_to_clk1	clkreq 1 tp clk running in time units, You can randomize this delay from your test, using test commands REQ1_TO_ACK1	Time Default [0:30]
clk1_to_ack1	clk running to clkack 1 – min 1 clk to any number max You can randomize this delay from your test runtime, CLK1_TO_ACK1	In clk units default = 1 clk
req0_to_ack0	clkreq 0 to clkack 0 – min 1 to any number max Test command – REQ0_TO_ACK0	In clk units default = 1 clk
clkack_delay	clkack 0 to clock gating – min 8 to max any number Test command CLKACK_DELAY	In clk units default = 8clks

Field	Description	Possible values
enable_random_phase	This is to enable random phase relationship between all the clock slices in your clt. Even if you have clocks getting derived from the same clock source, owing to buffers/wire delays they could be out of phase. This feature will be made default in the subsequent release so please make sure you validate your IP with this enabled.	0 or 1 Default 1
freq_change_delay	Whenever you do clock source switching or div ratio changes on any slice. There will be short period in between the old and the new freq clock that the clock is gated for. This delay is now programmable using this field. You can look at test12 for examples	Time Default [60:80]
randomize_req1_to_clk1	This tells the ccu vc to go an randomize the req1_to_clk1 delay within range (min = 0 and max = req1_to_clk1 config param). So the req1_to_clk1 will act as the max limit and for every clkreq -> 1 the clkungating with take a different delay	Time Default = 0
randomize_clk1_to_ack1	When this is set the clk1_to_ack1 delay will be randomized within range (min = 1 and max = clk1_to_ack1 config param).	In clk units Default = 0
randomize_req0_to_ack0	This randomizes the req0 to ack 0 delay within range (min = 0 and max = req0_to_ack0 config param). Every clkreq -> 0 to clkack -> 0 will have different delay if this is enabled	In clk units Default = 0
randomize_clkack_delay	Setting this randomizes the clkack -> 0 to clkgating delay within range (min = 8 to max = clkack_delay). The clkack_delay config parameter becomes the max limit and every time the clkack -> 0 the clk gating delay will be different.	In clk units Default= 0
duty_cycle	This specifies the high phase of the clock. So if you set the duty cycle to 20 , the high phase of the clk will be for 20% of the full clk period and the remaining will be low phase	Int Default = 50

3.1.3.3 Sequences

Test Sequence Name	Parameters	Function
ccu_seq	ccu_xaction	It has one random transaction that the user is supposed to assign and then start this sequence and it will execute the transaction.

For an example of using the ccu_seq, refer to verif/tests/test02.svh

3.2 Environment Settings and Files

3.2.1 Verification Component

The VC class that the user is required to instantiate in his environment to use the VC is ccu_vc. In addition, the user must instantiate a VC configuration class (ccu_vc_cfg), set the various configuration variables and pass that to the VC.

```
class env extends ovm_env;
// Instantiate config object and agent
  ccu_vc_cfg i_ccu_vc_cfg;
  ccu_vc i_ccu_vc;

 function void build ();
   super.build ();
// Create the agent and config object
   i_ccu_vc = ccu_vc::type_id::create ("i_ccu_vc" ,this);
   i_ccu_vc_cfg = ccu_vc_cfg::type_id::create ("i_ccu_vc_cfg");

//connect the config object to agent using set_config call
   set_config_object ("i_ccu_vc*", "CCU_VC_CFG", i_ccu_vc_cfg, 0);
endfunction :build
```

3.2.2 Configuration Object

Once the configuration object is created as shown in the previous section, the user needs to configure the VC to function as per his requirements. Please refer to section 3.2.2.1 for details on the available configuration variables and the corresponding APIs

3.2.3 HDL file to be used

The package HDL file should be used in your env

verif/ccu-vc/ccu_vc_pkg.hdl

If you are using any revision of ccu prior to r130521 then you would need the sip_vintf_manager.hdl. We have now removed dependency on this package and rely on the saola vif containers instead. If

you are already using the iosf svc or pvc, you are already have the sip_vintf_manager getting compiled with one of those vcs. Please point to the sip vintf manager hdl, wherever it is in your clone

sip_vintf_manager.hdl

The ccu_env is there ONLY for your reference, in your IP environment there will will be a IP_env file extended from the ovm_env, the type_id creations for the ccu vc, the object and the set config calls can go there. Currently I use the ccu_env for my stand alone testbench.

Please do not use the ccu_env package in your IP env/TI, it is only meant to be used an example.

3.2.4 CCU_VC TI example

```
IP Env – ovm name
usb_env = USBEnv::type_id::create:: ("usb1_env", this);

Agent – ovm name
ccu = ccu_vc::type_id_create:: ("usb_ccu_agent", this);

CCU vc TI instance
ccu_vc_ti #(.NUM_SLICES(4),
IS_ACTIVE(1),
.IP_ENV_TO_AGT_PATH ( "*usb1_env.usb_ccu_agent ") ,
....
Ccu_ti (ccu_if);
```

3.3 Steps to Compile and Run Unit Tests

1. Download and extract the VC tar file into a directory ($IP_ROOT)
2. cd $IP_ROOT/scripts
3. source setup
4. ace –cc
5. ace –x
6. ace –x –t test02

Description of Tests

Test Name	Runcmd	Description
ccu_vc_test		
test01	ace –x test01 –sd –gui &	Randomizes clk src freq, clk gating

Test Name	Runcmd	Description
test02	ace –x test02 –sd –gui &	Enables usync on all slices except 3. It has programmed all the frequencies that BXT uses to show how usync is generated for different divide ratios
test03	ace –x test03 –sd –gui &	Randomizes variod operations like gating, div_ratio
test04	ace –x test04 –sd –gui &	Makes sure all operations on the slices take effect. Self checking test
test05	ace –x test05 –sd –gui &	Has slices belonging to same dcg blk num
test06	ace –x test06 –sd –gui &	Clk gating with slices on same dcg blk
test07	ace –x test07 –sd –gui &	Negative test for gating clkack so most and keep asserting clkreq to violate the req-ack protocol
test08	ace –x test08 –sd –gui &	Randomly programs req-ack handshake from test
test09	ace –x test09 –sd –gui &	Randomly programs req-ack handshake from test with 2 slices in the same dcg blk num
test10	ace –x test10 –sd –gui &	Programs req0_to_ack0 delay on one slice outside of the default range, randomly.
test11	ace –x test11 –sd –gui &	Programs varios handshake delays and enables random phase delay for all valid slices.
test12	ace –x test12 –sd –gui &	Randomizes the freq change delay within a min-max range and it changes freq on slices 1 and 2 randomly. Also forces clkack to be synchronous
test13	ace –x test13 –sd –gui &	Enables random_phase enable on all the slices, so all the clocks come out of phase by default
test14	ace –x test14 –sd –gui &	DEF_ON configuration on the clock slices 6 and 1 and DEF_OFF on others
test15	ace -x -t test15 -sd gui -simv_args +MAX_CLK_UNGATE_TIME=500ns	Shows the new api for clk period reporting. It is a longer test for performance measuring. It randomizes the clkack delay through the newly added config switch and the previously supported commands

Test Name	Runcmd	Description
test16	ace –x test16 –sd –gui &	Programs a CLK_SRC switch on clk slice 3 which is the free running clk other clk slices are in phase with. Programs the config object for slice 2 with unequal duty cycle of 20.

4. Agent Packages

The following package needs to be imported to be able to use all the components of this verification component.

```
import ccu_vc_pkg::*;
```

List of files included in the package are as follows.

```
ccu_vc.svh
ccu_driver.svh
ccu_seq.svh
ccu_vc_checker.svh
ccu_seqr.svh
ccu_vc_params.svh
ccu_vc_cfg.svh
ccu_types.svh
ccu_xaction.svh
clksrc_cfg.svh
slice_cfg.svh
dcg_controller.svh
```

The interface file and test island file

```
ccu_vc_ti.sv
cci_intf.sv
```

5. Agent Parameters

The parameters are used to set the number of clocks and number of resets.

```
parameter NUM_SLICES;
parameter IP_ENV_TO_AGT_PATH;
parameter IS_ACTIVE;
parameter REQ1_CLK1_MIN[NUM_SLICES];
parameter REQ1_CLK1_MAX[NUM_SLICES];
parameter REQ0_ACK0_MIN[NUM_SLICES];
parameter REQ0_ACK0_MAX[NUM_SLICES];
parameter CLK1_ACK1_MIN[NUM_SLICES];
parameter CLK1_ACK1_MAX[NUM_SLICES];
parameter ACK0_CLK0_MIN[NUM_SLICES];
parameter ACK0_CLK0_MAX[NUM_SLICES];
```

6. Agent Interface

6.1 CCU_VC_ti Interface

```
output logic clk[NUM_SLICES-1:0];
input logic clkreq[NUM_SLICES-1:0];
output logic clkreq[NUM_SLICES-1:0];
output logic usync[NUM_SLICES-1:0];
output logic global_usync;
input logic reset[NUM_SLICES-1:0];
input logic pwell_pok[NUM_SLICES-1:0];
input logic global_rst_b[NUM_SLICES-1:0];
```

7. Checker

The ccu vc checker, uses SV assertions to validate the IP-DCG/CCU clkreq-clkack protocol rules that are listed in the Chassis Clocking Spec.

Check name	Cspec Rule	Assertion pseudo code
clkreq_rise	5 - Clkreq can assert only if clkack is deasserted	assert property (disable iff(pwell_pok !== 1'b1) $rose(clkreq) |-> clkack == 1'b0 else `ovm_error("slice num <>, clkreq rise forbidden when clkack 0")
clkreq_fall	6 - Clkreq can deassert only if clkack is asserted	assert property (disable iff(pwell_pok !== 1'b1) $fell(clkreq) |-> clkack == 1'b1 else `ovm_error("slice num <>, clkreq fall forbidden when clkack 1")
clkack_rise	8 - Clkack can assert only if clkreq is asserted	assert property (disable iff(pwell_pok !== 1'b1) $rose(clkack) |-> clkreq == 1'b1 else `ovm_error("slice num <>, clkack rise forbidden when clkreq 0")
clkack_fall	7 - Clkack can deassert only if clkreq is deasserted	assert property (disable iff(pwell_pok !== 1'b1) $fell(clkack) |-> clkreq == 1'b0 else `ovm_error("slice num <>, clkack fall forbidden when clkreq 1")
clk_gate_min_8_chk	9 - Clock must continue to operate for a minimum of 8 clock cycles after deasserting clkack	assert property (disable iff(pwell_pok !== 1'b1) $fell(clkack) && !clkreq |-> (gate clks within 8 clks) else `ovm_error("slice num <>, clk needs to stay active for 8 min clks before gating")
clkack_1clk_chk	10 - Clock must operate for minimum 1 clock cycle before asserting clkack	assert property (disable iff(pwell_pok !== 1'b1) $rose(clkreq) |-> ##1 clkack else `ovm_error("slice num <>, clkack should assert 1 clk after clk ungates")
req1_ack1_chk	5	assert property (disable iff(pwell_pok !== 1'b1) $rose(clkreq) |-> eventually clkack else

		`ovm_error("slice num <>, clkack never asserted in response to clkreq rise")
req0_ack0_chk	6	assert property (disable iff(pwell_pok !== 1'b1) $fell(clkreq) |-> eventually !clkack else `ovm_error("slice num <>, clkack never de-asserted in response to clkreq fall")
req1_clk1_max_chk	This check is added to detect if any IPs ungate request-granting takes longer than expected	assert property (disable iff $isunknown(max_ungate_time) $changed(clk_ungate_time) |-> clk_ungate_time < max_ungate_time `ovm_error("slice num <>, clk ungate time %0t exceeds the max ungate limit %0t")

- These are SV assertions and can be disabled in your testbench using the $assert_off.
- Anytime from the test you try to GATE/UNGATE the clocks which are like a force ungate and ate there is a chance that the clkreq-ack protocol will get violated and the checks will flag.
- If you do not connect anything to the pwell_pok then all your checks would be disabled always and you do not get any protocol checking. There will be coverage that will eventually catch all false passes. So please avoid this.
- The IP_ENV_TO_AGT_PATH is the parameter used for making the error messages unique, so in the log file with the OVM_ERROR you will the path to the checker that is flagging the error
  Example:
  ...../verif/ccu-vc/ccu_vc_checker.svh", 40: **ccu_vc_tb.ccu_ti_1.ccu_vc_assertions[5].clkreq_rise:** started at 351915000fs failed at
  Offending '((ccu_intf.clkack[5] == 1'b0) ..
  The ovm error line will give you the instance that is giving this error, ONLY if you provide the instance name in the IP_ENV_TO_AGT_PATH parameter,
  OVM_ERROR /nfs/fm/disks/fm_cse_04233/rravindr/ip-ccu-vc-2013WW08r130219/ccu_vc_WW52/verif/ccu-vc/ccu_vc_checker.svh(40) @ 351915000: reporter [***ovm_ccu_vc_env.ccu_vc_1 ccu checker**] Slice num 5, clkreq rise forbidden when clkack = 01

8. Coverage

Coverage is added to the ccu. Will be located in ccu_vc_ti.ccu_vc_checker.svh,

IMPORTANT NOTE: To disable all the coverage you get from the CCU please set this in your env, DISABLE_CCU_COVERAGE. The coverage is wrapped in a `ifndef DISABLE_CCU_COVERAGE , so you should be able to disable the coverage if needed.

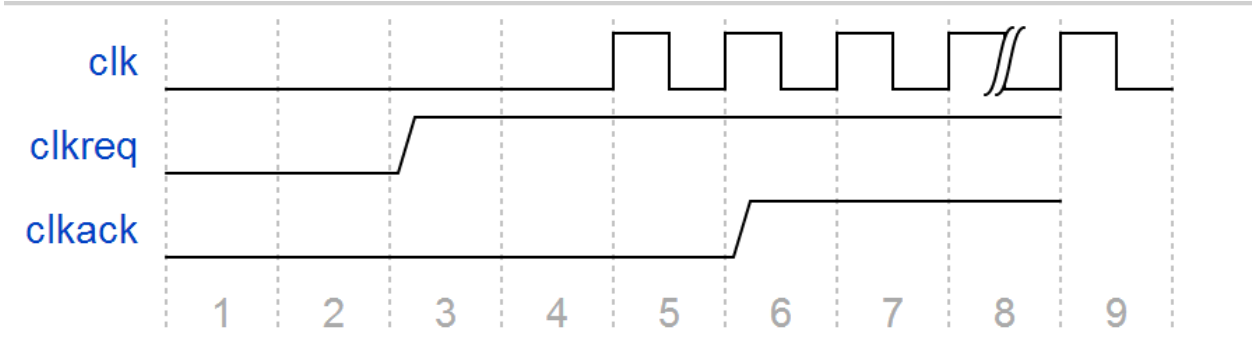
8.1 Basic Coverage

Just checking for rising falling edges on the req-ack and making sure that global reset and pwell pok are not just tied off to 0 (since that would disable all checks)

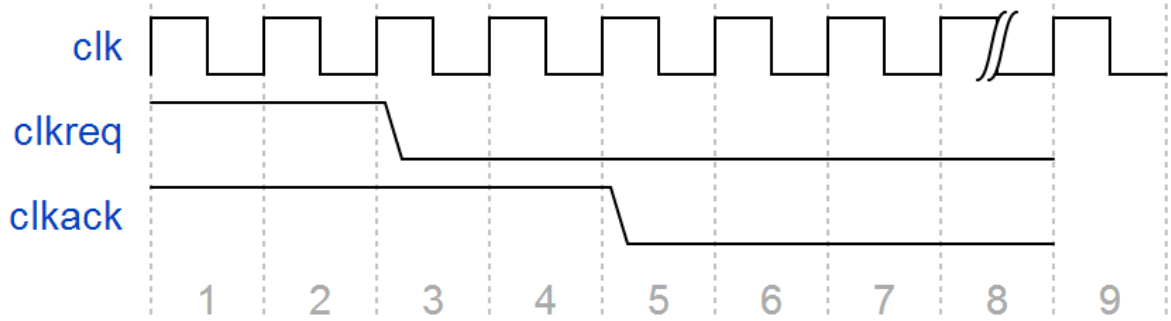
Cover Property	Description
$rose(global_rst_b)	Rising edge global reset, Helps check if IPs have not enabled checking, global rst is tied off to 0
$rose(pwell_pok)	Rising edge pwell_pok Helps check if IPs have not enabled checking, pwell_pok is tied off to 0

8.2 Combinations of req-ack

Cover Property
clkreq === 1'b1 && clkack === 1'b1
clkreq === 1'b1 && clkack === 1'b0



Cover Property
clkreq === 1'b0 && clkack === 1'b1
clkreq === 1'b0 && clkack === 1'b0



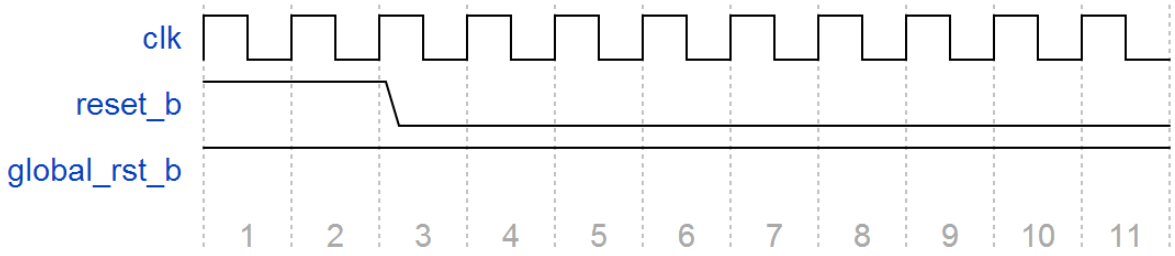
8.3 Req-ack delay bins

Currently just the min delay ones are implemented will add to this, may be give users the ability to program the delays they want to see covered.

Cover Property	Description
clkreq = 1 to clk ungating [req1_to_clk1]	Min-med-high values of delay from clkreq rising to clk ungating.
Clk ungating to clkack = 1 [clk1_to_ack1]	Min-med-high values of delay from clkungating to clkack rising
Clkreq = 0 to clkack = 0 [req0_to_ack0]	Min-med-high values of delay from clkreq falling to clkack falling
Clkack = 0 to clk gating [clkack_delay]	Min-med-high values of delay from clkack falling to clk gating

8.4 Global reset

Cover Property	Description
$fell(reset_b) && global_rst_b	Making sure IPs assert their resets when the global reset is still asserted , This is to ensure that the req-ack handshake is maintained at all times even if reset is asserted



8.5 Example of coverage collected

Name	Type	Cov%	Target	RealCov	Violated
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[0].clk_gate_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[0].req0_ack0_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[3].clk_gate_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[3].req0_ack0_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[0].clk_gate_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[0].req0_ack0_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[1].clk_gate_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[1].req0_ack0_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[2].clk_gate_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[2].req0_ack0_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[3].clk_gate_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[3].req0_ack0_cover	Skew	0.00	1	0	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[0].req1_ack1_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[1].clk_gate_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[1].req0_ack0_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[1].req1_ack1_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[2].clk_gate_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[2].req0_ack0_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[2].req1_ack1_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[3].req1_ack1_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[4].clk_gate_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[4].req0_ack0_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[4].req1_ack1_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[5].clk_gate_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[5].req0_ack0_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[5].req1_ack1_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[6].clk_gate_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[6].req0_ack0_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_1::ccu_vc_assertions[6].req1_ack1_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[0].req1_ack1_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[1].req1_ack1_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[2].req1_ack1_cover	Skew	100.00	1	1	0
ccu_vc_tb.ccu_ti_2::ccu_vc_assertions[3].req1_ack1_cover	Skew	100.00	1	1	0
total:Groups=0 Covers=33 Items=33 Asserts=0 Modules=2	Mixed	63.64	33	21	0

9. Tracker

9.1 Tracker Overview

Describe the tracking features that this tracker support - ie. packets, linkstates, sideband signalling.

9.2 Sample Output

Shows snippet of the tracker format with explaination

10. Open Issues and To Do List

11. Requesting Support

Here is the current list of things to do to finish Agent development.

TODO: analysis port addition – low priority
Throw events freq change/gate/etc – use for checker
Handoff check list items see if anything belongs here
~~Mention tool versions used for the compilation~~
Gusync counter
Power state sequencning comprehend Pwrok
Sources are not enabled by pmc