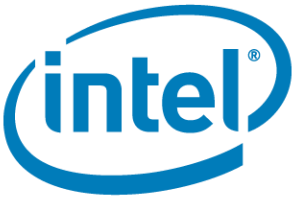


CCU VC

INTEGRATION GUIDE

IP Rev. 0.7

Rajitha Ravindran

Intel Restricted Secret

This document contains information on products in the design phase of development.

INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL PRODUCTS. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO SALE AND/OR USE OF INTEL PRODUCTS INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

UNLESS OTHERWISE AGREED IN WRITING BY INTEL, THE INTEL PRODUCTS ARE NOT DESIGNED OR INTENDED FOR ANY APPLICATION IN WHICH THE FAILURE OF THE INTEL PRODUCT COULD CREATE A SITUATION WHERE PERSONAL INJURY OR DEATH MAY OCCUR.

Intel may make changes to specifications and product descriptions at any time, without notice. Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them. The information here is subject to change without notice. Do not finalize a design with this information.

The products described in this document may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your Intel account manager or distributor to obtain the latest specifications and before placing your product order.

Copies of documents that have an order number and are referenced in this document or in other Intel literature can be obtained from your Intel account manager or distributor.

Contents

Author	Revision No.	Description	Revision Date
Rajitha Ravindran	0.7	Initial description of CCU VC including integration, configuration and reuse	

1 About This Document

1.1 Audience

The information in this document is intended for an integration team that is integrating this IP into an SoC.

1.2 Supported Projects

This document supports the following projects at the listed RTL maturity level.

Project Name	IP Maturity Level

1.3 Getting Help

If all else fails, here is human contact information.

Name	Function	Email
Rajitha Ravindran	Component Design Engr	rajitha.ravindran@intel.com

1.4 Related Documents

If you need more information on this IP, you may find these documents helpful:

- Chassis Clocking HAS 0.5

2 Quick Start

2.1 Downloading the VCs Required for Compilation

There are no VCs required for this VC to run. It is supposed to be self-contained and run out of the box.

2.2 Integrity Checks for Standalone IP

Following are steps for running standalone integrity checks of this IP.

1. Run the environment script:

   ```
   cd $IP_ROOT/scripts
   source setup
   ```

2. Compile the model:

   ```
   ace -cc
   ```

3. Run a simple test.

   To run basic test:

   ```
   ace -x
   ```

   To run the sample test “test01” (included in the release):

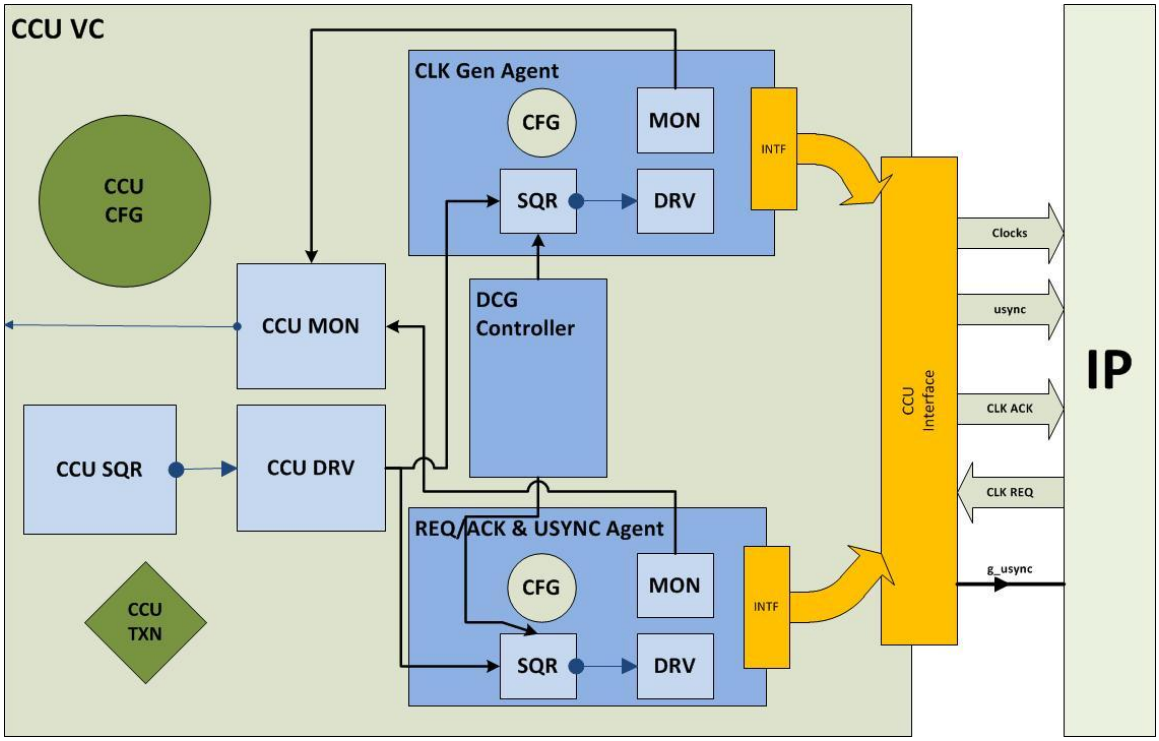
   ```
   ace -x -t test01
   ```

   To run the test interactively:

   ```
   ace -x -t test01 -sd gui
   ```

3 Verification Information for Integration

3.1 IP Testbench Overview



3.2 Setting up Testbench Env

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.
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

Signal	Connect to	Description
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

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
ccu_monitor	VC monitor that is monitoring all interface activities and broadcasting that to the outside via analysis ports	verif/ccu-vc/ccu_vc.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
ccu_intf	SV Interface that holds all the signals to be connected to the IP	verif/ccu-vc/ccu_intf.sv

Component Name	Description	File location
ccu_ti	Test Island that should be instantiated in user's top level module and passed a ccu_intf instance	verif/ccu-vc/ccu_ti.sv

Configuring the IP Environment

3.1.1.1.1 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

3.1.1.1.2 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

3.1.1.1.3 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

3.1.1.2 CCU Transaction

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.

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
clk_src	Defines the new clock source to be applied to this slice. Applicable only when cmd == CLK_SRC	0 to 15
div_ratio	Defines the divide ratio to be applied to post divider of the slice. Applicable when cmd == DIVIDE_RATIO	DIV_1 --- DIV_16

Field	Description	Possible values
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	0 or 1

3.1.1.3 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

3.1.1.4 SoC Requirements for Sequence Reuse

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 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 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 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.

3.2.3 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);
```

3.2.4 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.

3.3 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

3.4 Description of Tests

Test Name	Runcmd	Description
ccu_vc_test		
test01	ace –x test01 –sd –gui &	Randomizes clk src freq, clk gating
test02	ace –x test02 –sd –gui &	Changes clk freq for slices , clk gating
test03	ace –x test03 –sd –gui &	Randomizes variod operations like gating, div_ratio

4 Tools and Methodology for Integration

4.1 Supported Tools

The following tools are used in the integration of this IP. For versions supported by each release, see Release Notes in the "doc" directory of the release package.

- VCSMX F-2011.12-1
- OVM 2.1.1_2
- Ace 2.00.01
- Saola v20120206
- Verdi 2010.10

4.2 Directory Structure

|-- ace	ACE scripts
|-- doc	All documents related to this VC
|-- scripts	Environment setup and tools files
`-- verif	
|-- ccu-vc	VC Files reside here
|-- ccu_env	Example OVM envronment using the VC
|-- lib	
| |-- CCU_OOB	CLK REQ/ACK and USYNC agent resides here
| |-- CRG	Clock generator files reside here
| `-- sip_vintf_manager	
| |-- example	
| `-- src	
|-- tb	Top level module instantiting and using the VC interface and TI
`-- tests	Sample tests showing different use models

5 Integration Test Plan

Describe the integration test plan here.