Vivante GCNanoUltraV Series
Small Footprint Vector Graphics IP

Hardware Integration Guide
for
V2.0x

Document Revision 0.88
23 February 2023

This document is compatible with Vivante
GCNanoUltraV hardware versions 2.0.x

Legal Notices

COPYRIGHT INFORMATION

This document contains proprietary information of Vivante Corporation and VeriSilicon Holdings Co., Ltd. They reserve the right to make changes to any products herein at any time without notice and do not assume any responsibility or liability arising out of the application or use of any product described herein, except as expressly agreed to in writing by Vivante and/or VeriSilicon; nor does the purchase or use of a product from Vivante or VeriSilicon convey a license under any patent rights, copyrights, trademark rights, or any other of the intellectual property rights of Vivante, VeriSilicon or third parties.

DISCLOSURE/RE-DISTRIBUTION LIMITATIONS

The information contained herein is not to be used by or disclosed to third parties without the express written permission of an officer of Vivante Corporation or VeriSilicon Holdings Co., Ltd.

(VeriSilicon Distribution **LEVEL D: CONFIDENTIAL – NOT FOR REDISTRIBUTION**).

TRADEMARK ACKNOWLEDGMENT

VeriSilicon® and the VeriSilicon logo design are the trademarks or the registered trademarks of VeriSilicon Holdings Co., Ltd. Vivante® is a registered trademark of Vivante Corporation. All other brand and product names may be trademarks of their respective companies.

For our current distributors, sales offices, design resource centers, and product information, visit our web page located at http://www.verisilicon.com.

For technical support, please email vivante-support@verisilicon.com.

Preface

This document is the primary integration guide for Vivante vector graphics processing unit IP. It includes a discussion of integration, prototyping, and verification.

Audience

This document was prepared for both hardware and software integrators who are familiar with system-on-a-chip (SoC) digital design and related support devices. Those who would benefit from this technical manual are:

- Engineers and managers who are evaluating this Vivante GCNanoUltraV IP for use in a system
- Engineers who are designing this Vivante GCNanoUltraV IP into a system

Conventions Used in This Document

AHB – Advance High Performance Bus
APB – Advanced Peripheral Bus
AXI – Advanced eXtensible Interface
CDN – Composition Control Engine
DV – Design Verification
DMA – Dynamic Memory Access
FE – Graphics Pipeline Front End / Fetch Engine
FPGA – Field Programmable Gate Array
GPU – Graphics Processing Unit
GUI – Graphical User Interface
HI – Host Interface
ICG – Integrated Clock Gating
IM – Imaging Engine
MC – Memory Controller
P&R – Place and Route
PD – Physical Design
PE – Pixel Engine
PM – Power Management
RTL – Register Transfer Level
SB – Shared Buffer Control
SDC – Sensor Data Control
SDF – Synchronous Data Flow
SoC – System on Chip
TS – Tessellation Engine
VG – Vector Graphics

The word *assert* means to drive a signal true or active. Signals that are active LOW end in an "n."
Hexadecimal numbers are indicated by the prefix "0x" —for example, 0x32CF.
Binary numbers are indicated by the prefix "0b" —for example, 0b0011.0010.1100.1111.
Code snippets are given in `Consolas` typeset.

Table of Contents

List of Figures

List of Tables

1 Overview

This document is the primary integration reference guide for the Vivante GCNanoUltraV Series small footprint Vector Graphics processing unit IP cores. Designed for easy integration into SoCs, these VG cores provides high performance, high quality vector graphics, low power consumption, and the smallest silicon footprint for their class. The cores are delivered as synthesizable RTL. They are technology independent and can be synthesized using a variety of libraries.

1.1 References

- **Vivante GCNanoUltraV Series Hardware Features** provides the feature set available with this IP core.
- **Vivante GCNanoUltraV Series Hardware Integration Guide** is this document.
- **Vivante GCNanoUltraV Series Hardware Implementation** provides implementation reference material for the physical design implementation of this IP core.
- **Vivante GCNanoUltraV Series Accessible Registers** provides the accessible registers specification for this IP core.
- **Vivante VGLite API** provides a guide to Vivante's platform independent API for GCNanoUltraV Series IP.

1.2 Core Variants

The GCNanoUltraV Vector GPU IP includes a 32-bit AHB interface for register access and two 32-bit AHB interface for external memory access.

For available variants, refer to the Hardware Features document for this IP.

2 Hardware Design Release Package

The Hardware Design Package contains a README and five directories: RTL, Design Verification, SoC Tests, Wrapper Test and Documentation. Preliminary packages may not contain all directories.

2.1 RTL

The RTL directory contains all the RTL files and scripts to compile the GC Core design for simulation, synthesis and P&R.

Table 1. RTL Directory Description

rtl directory item	Description
clockgater	Directory containing CLOCKGATER.v file.
common	Directory of the Verilog modules for use by the FPGA
gcnanoultrav	Directory of the GCNanoUltraV design files
inc	Directory of the design include files (*.h, *.vh)
rams	Directory containing memory models
undef	Directory containing a list of `undef text macros which will cancel text macros previously defined in this rtl. NOTE: Please compile rtl/undef/<prefix>_vsi_undef.v as the last Verilog file for this RTL.
gcnanoultrav_filelist.txt	List of the RTL files

2.2 DV Design Verification

The Design Verification directory has a self-contained test bench and tests to verify that all RTL files are included and that they are valid.

Table 2. Design Verification Directory Description

dv directory item	Description
bench	Directory of the test bench files
pli	Directory containing a sparse memory PLI model
test	Directory of the functional tests for the stand-alone test bench
README	Instructions for running the stand-alone tests to verify the RTL
cmd.txt	Explains the command format used in the tests for the stand-alone test bench

2.3 SoC Tests

The SOC_Tests directory has the stimulus and response for the SoC tests which will make sure this GPU IP is well integrated with the SoC.

Table 3. SoC_Tests Directory Description

SOC_Tests Directory Item	Description
GCNANOULTRAV_README.txt	Description of the SoC tests
<tests>	Subdirectory per test, with the memory buffer stimulus and response to be compared

2.4 Documentation

The “doc" directory contains the memory specification.

Table 4. doc Directory Description

doc Directory Item	Description
memory_spec	Memory specification.

2.5 RAM Wrapper Test

The “wrapper_test" directory contains a python script for testing ram wrapper.

Table 5. wrapper_test Directory Description

wrapper_test Directory Item	Description
ram_wrapper_bench_gen.py	Python script for testing ram wrapper.
Readme.WrapperTest.txt	README for wrapper testing

3 Integration

The integration flowchart may be used for checking the basic sequence of steps and can help to make your integration proceed more smoothly.

The main steps of hardware integration are:

- setting up the interface connections
- integrating the technology specific internal memory components
- integrating the technology specific clock gating component
- running the provided test cases for sanity checking

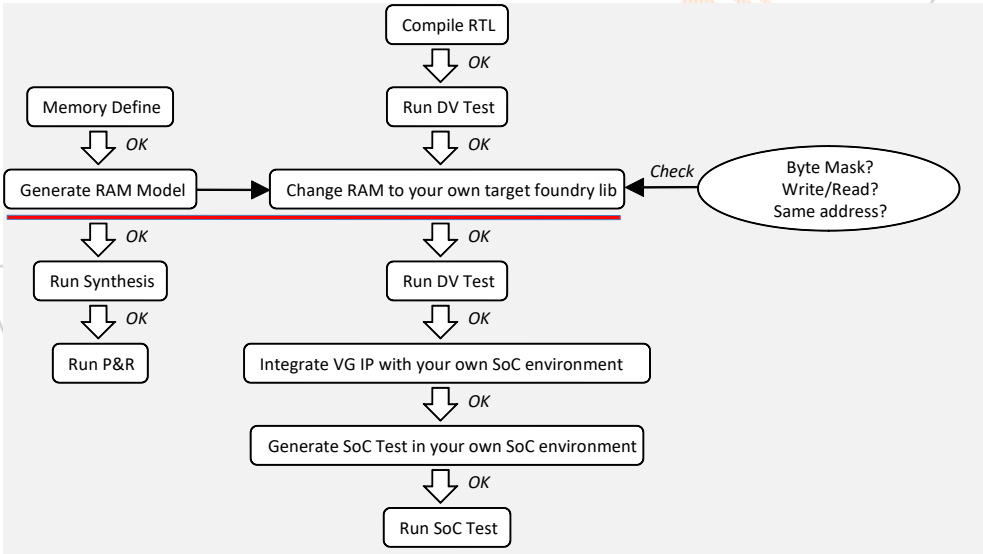


Figure 1. GPU IP Integration Flowchart

3.1 Design Hierarchy

The RTL hierarchy of this product configuration is shown here. Please refer to the README file in the *rtl* directory of the Hardware Design Release Package for the design hierarchy. Note that directory names do not always reflect functionality.

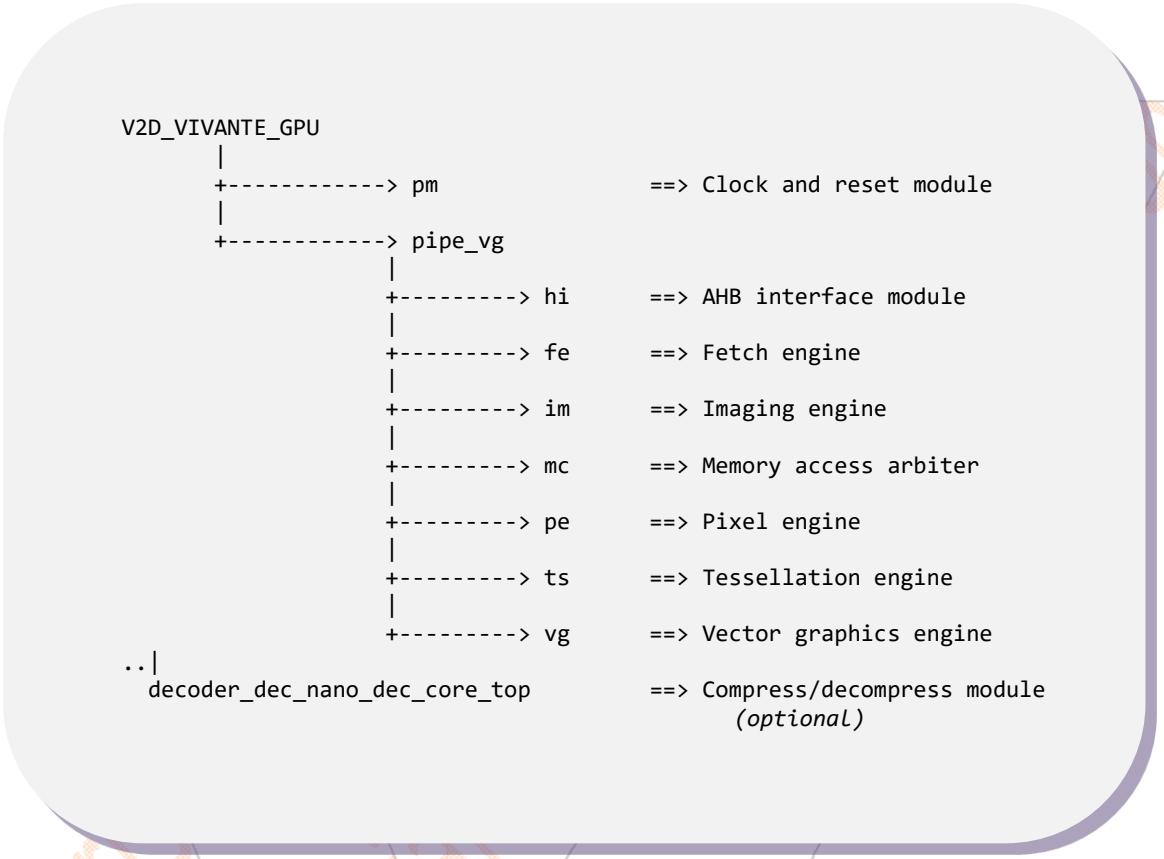


Figure 2. RTL Hierarchy

3.2 Host Interfaces

The GCNanoUltraV may be configured prior to RTL release to provide interface arrangements consisting of the following. APB is an alternate option to AHB.

- 1 64-bit AXI and 1 32-bit AHB (or APB)
- 32-bit AHB (or APB)

Two major interfaces are used to connect the GPU to the rest of the SoC: AXI and AHB. The AXI master interface included in the GPU IP is used to fetch data from memory that is attached to the SoC AXI interconnect. The AHB/APB slave interface is used by the SoC processor to access the graphics IP registers for configuration, debug, and test.

For 3 AHB/APB: Three AHB interfaces are used to connect the GPU to the rest of the SoC. Two AHB master interfaces included in the GPU IP are used to fetch data from memory that is attached to the SoC AHB interconnect. One AHB slave interface is used by the SoC processor to access the graphics IP registers for configuration, debug, and test.

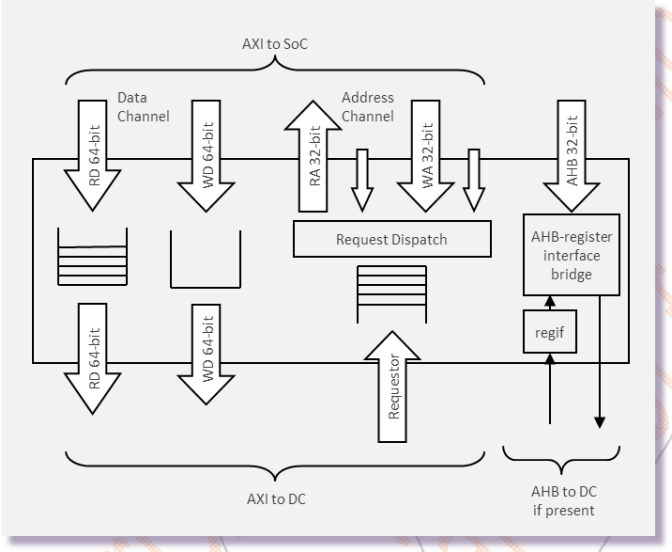


Figure 3. Host Interface Diagram with AXI+AHB

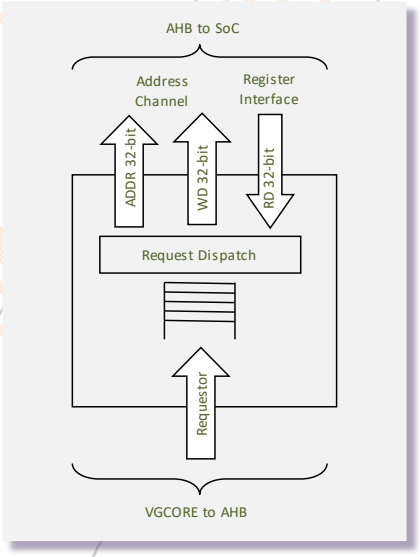


Figure 4. Host Interface Diagram with 3xAHB

3.2.1 AXI Interface

Main features of the AXI interface:

- One independent read and write 64 bit data bus
- Multiple burst length (8 bytes, 16 bytes, 32 bytes, or 64 bytes)
- Supports out-of-order return data for different IDs
- Asynchronous interface to the GPU
- Optimized for size while meeting performance requirements

If the AXI DMA master receives an ERROR response from the AXI fabric, The GPU will log it and generate an INTERRUPT if AXI interrupts are enabled.

Refer to the file in your hardware package ***rtl/gcnanoultrav/V2D_VIVANTE_GPU.v*** for a complete pin list specific to your variant.

3.2.1.1 AXI Pin Description

Table 6. AXI Pins Description

Signal Name	Width	I/O	Description	Clock Domain
ARREADY	1	IN/OUT	Read address ready	Core
AWREADY	1	IN/OUT	Write address ready	Core
BID	4	IN/OUT	Response ID	Core
BRESP	2	IN/OUT	Write response	Core
BVALID	1	IN/OUT	Write response valid	Core
RDATA	64	IN/OUT	Read data	Core
RID	4	IN/OUT	Read ID tag	Core
RLAST	1	IN/OUT	Read last	Core
RRESP	2	IN/OUT	Read response	Core
RVALID	1	IN/OUT	Read valid	Core
WREADY	1	IN/OUT	Write ready	Core
ARADDR	32	OUT/IN	Read address	Core
ARBURST	2	OUT/IN	Burst type (read), always INCR	Core
ARCACHE	4	OUT/IN	Cache type (read)	Core
ARID	4	OUT/IN	Read address ID	Core
ARLEN	4	OUT/IN	Burst length (read)	Core
ARLOCK	2	OUT/IN	Lock type (read), always NORMAL	Core
ARPROT	3	OUT/IN	Protection type (read), always NORMAL, NON-SECURE, DATA	Core
ARSIZE	3	OUT/IN	Burst size (read)	Core
ARVALID	1	OUT/IN	Read address valid	Core
AWADDR	32	OUT/IN	Write address	Core
AWBURST	2	OUT/IN	Burst type (write), always INCR	Core
AWCACHE	4	OUT/IN	Cache type (write)	Core
AWID	4	OUT/IN	Write address ID	Core
AWLEN	4	OUT/IN	Burst length (write)	Core
AWLOCK	2	OUT/IN	Lock type (write), always NORMAL	Core
AWPROT	3	OUT/IN	Protection type (write), always NORMAL, NON-SECURE, DATA	Core
AWSIZE	3	OUT/IN	Burst size (write)	Core
AWVALID	1	OUT/IN	Write address valid	Core
BREADY	1	OUT/IN	Write response ready	Core
RREADY	1	OUT/IN	Read ready	Core
WDATA	64	OUT/IN	Write data	Core
WID	4	OUT/IN	Write ID tag	Core
WLAST	1	OUT/IN	Write last	Core
WSTRB	8	OUT/IN	Write strobe (byte lane)	Core
WVALID	1	OUT/IN	Write valid	Core

3.2.1.2 AXI Burst Size and Burst Length Alignment

AxBURST should be set to INCR only. AxSIZE is always 3(0b011) for 64-bit. AxLEN can range from 0~7. Valid values are indicated in the table below.

Table 7. AXI Burst Size and Burst Length Alignment

	Burst Alignments for 64-bit AXI								
AxSIZE	always 3 (0b011) 8 bytes for 64-bit AXI								
AxLEN Burst length value	0	1	3		7				
AxLEN Burst length (valid bytes per cycle)	8	16	32		64				
Alignment (in bytes)	8	16	32		64				

3.2.1.3 Read and Write requests

If the SoC can only accept a limited number of read and/or write requests, the fabric can de-assert AWREADY/ARREADY/WREADY signals at any time and the GPU will stop issuing the related requests. Outstanding read and write requests impact GPU performance. The ideal number should be determined based on the latency of read and write responses from the bus. If the latency is low, a lower outstanding read/write request limit will not cause performance degradation.

The AXI ID bus uses the values shown in the table below to indicate the source or destination of the request. The AXI ID bus is 4 bits wide. Reserved values are for internal use by the GPU.

Table 8. AXI Requestor ID [3:0]

AXI ID[3:0]	Requestor
0	Pixel Engine Color Read (frame buffer read)
1	Pixel Engine Color Write (frame buffer write)
2	Tessellation Read
3	Tessellation Write
4	Reserved
5	Front End (DMA) Read
6	Vector Graphics Read
7~9	Reserved
10	Imager Read
15~11	Reserved

3.2.1.4 AXI3 AXI4 Considerations

Vivante IP implements a limited subset of possible features specified in AMBA3 AXI3 and AMBA4-AXI4.

For implementations with an AXI3 system bus, AMBA4 signals should not be connected on the system side.

Note that the AxUSER field size may vary. The following table summarizes the key differences between AXI3 and AXI4 in respect to Vivante GPU implementations.

Table 9. Key Interface Differences between AXI3 and AXI4

Feature	AXI3 Specification	AXI4 Specification	Vivante Support
WID (Write interleave)	Yes	Discontinued	Never issue interleaved write transactions, tie to 0.
AxLEN (Max Burst Length field size)	16 (4 bits)	256 (8 bits)	Vivante currently does not use a value over 4-bit length. Default is 0 (length 1)
AxLOCK (Lock access)	Yes	Discontinued	No locked transactions, set to 0(NORMAL)
AxQOS	No	Introduced	No support, tie to 0. *"A default value of 0b0000 indicates that the interface is not participating in any QoS scheme."*
AxREGION	No	Introduced	No support, tie to 0 (default).
AxUSER	No	Introduced	If ACELite signals present, often used mainly for virtualization (field size varies)

3.2.2 AHB Interface

Main features:

- 256 Kbyte addressable register space
- 32-bit accesses for register access, 8, 16, 32, 64 Byte bursts for memory data access.
- 32-bit data bus
- Handles error response for illegal accesses
- Asynchronous interface to the GPU
- Interrupt support

The AHB interface uses two separate clocks, one is the register access clock which is slower than the memory access clock and allows more relaxed logic design. Because the core clock is different from the interface clock, incoming data and outgoing data are handled properly across the clock boundary.

For register access, this GCCORE occupies 256 Kbytes (64K 32-bit words) of system address space; GCCORE decodes bits HADDR[17:2] for the GPU internal address; however, the range of addresses currently used is only 4 Kbytes (HADDR[11:0]). HADDR[31:18] should decode HSEL. HADDR[31:18] should be tied to 0.

An AHB ERROR response will be returned if an illegal access is detected.

This GCCORE AHB only accepts 32-bit read/write data. AHB transfers initiated with 8-bit (HSIZE[2:0]='b000) or 16 bit (HSIZE[2:0]='b001) are treated as 32-bit transfers and do not return an AHB error response.

Refer to the file in your hardware package ***rtl/gcnanoultrav/V2D_VIVANTE_GPU.v*** for a complete pin list specific to your variant.

3.2.2.1 AHB Pin Description

Table 10. AHB Pin Descriptions

Signal Name (for register access)	Width	I/O	Description (1x AHB)
HADDR	32	IN	AHB address bus
HBURST	3	IN	AHB burst type (not used)
HCLK	1	IN	AHB interface clock for register access
HPROT	4	IN	AHB protection control (not used)
HREADY	1	IN	AHB transfer status. HIGH indicates transfer has finished on the bus.
HRESETn	1	IN	AHB reset for register access (active LOW)
HSEL	1	IN	AHB Select
HSIZE	3	IN	AHB transfer size (not used, fixed transfer size of 32-bits)
HTRANS	2	IN	AHB transfer type (NONSEQUENTIAL, SEQUENTIAL, IDLE or BUSY)
HWDATA	32	IN	AHB write data
HWRITE	1	IN	AHB write. HIGH indicates a write transfer and LOW a read transfer.
HRDATA	32	OUT	AHB read data
HREADYOUT	1	OUT	AHB ready out
HRESP	2	OUT	AHB bus response
Signal Name (for memory access with 3xAHB designs)	**Width**	**I/O**	**Description (for 2x AHB in designs with 3xAHB)**
ACLK	1	IN	AHB interface clock for memory access
ARESETn	1	IN	AHB reset for memory access (active LOW)
HADDR0, HADDR1	32	IN	AHB address bus
HBURST0, HBURST1	3	IN	AHB burst size (not used)
HPROT0, HPROT1	4	IN	AHB protection select (not used)
HREADY0, HREADY1	1	IN	AHB transfer status. HIGH indicates transfer has finished on the bus.
HSEL0, HSEL1	1	IN	AHB Select
HSIZE0, HSIZE1	3	IN	AHB transfer size (not used, fixed transfer size of 32-bits)
HTRANS0, HTRANS1	2	IN	AHB transfer type
HWDATA0, HWDATA1	32	IN	AHB write data
HWRITE0, HWRITE1	1	IN	AHB write
HRDATA0, HRDATA1	32	OUT	AHB read data
HREADYOUT0, HREADYOUT1	1	OUT	AHB ready out
HRESP0, HRESP1	2	OUT	AHB bus response

3.2.2.2 Accessible Register Description

A description of those registers accessible via AHB or APB which can be “exposed” to your customers is provided in PDF or XML format. IPXACT format is available on request.

Register and field descriptions are not verbose, as their function may usually be discerned from the module, register and/or field name. Vivante tries to maintain a consistent register definition scheme across its cores. For this reason, some field bits may have a field name defined, but the field may not be in use for that specific core revision. For example, a field may be obsolete, or a field defined for use in 2D or 3D cores would not be used in a VG core. A reset value for such fields may be ignored.

Table 11. Exposed Registers for GCNanoUltraV Series

Module	Exposed Register	Byte Address	DWord Address	Read/Write
HI	AQHiClockControl	0x00000	0x0000	R/W
HI	AQHiIdle	0x00004	0x0001	R
HI	AQIntrAcknowledge	0x00010	0x0004	R
HI	AQIntrEnbl	0x00014	0x0005	R/W
HI	GCChipRev	0x00024	0x0009	R
HI	GCChipDate	0x00028	0x000A	R
HI	gcregHIChipPatchRev	0x00098	0x0026	R
HI	gcregHIProductId	0x000A8	0x002A	R
PM	gcModulePowerControls	0x00100	0x0040	R/W
PM	gcModulePowerModuleControl	0x00104	0x0041	R/W
PM	gcModulePowerModuleStatus	0x00108	0x0042	R
MC	AQMemoryDebug	0x00414	0x0105	R/W
MC	AQRegisterTimingControl	0x0042C	0x010B	R/W
FE	gcregFetchAddress	0x00500	0x0140	R/W
FE	gcregFetchControl	0x00504	0x0141	R/W
FE	gcregCurrentFetchAddress	0x00508	0x0142	R

The exposed registers tend to be stable across revisions, and XML and IPXACT files are not necessarily updated when the only changes to exposed registers are revision related. Key revision related registers are listed below. If an IPXACT or XML file specific to a particular release is needed, reset values can be updated to values unique for the IP release. Reset values can be found in the file ***rtl/gcnanoultrav/v2d_gc_hi_ahb2regif.v*** in the hardware release package. To locate the revision related values, search for the rtl year or for the 4 digit revision number, e.g., search on 2000 for a revision 2_0_0_rc0n.

```
assign hi_GCChipRevReg_d            = 32'h00002000;
assign hi_GCChipPatchRevReg_d       = 8'd14;
assign hi_GCChipDateReg_d           = 32'h20211209;
```

Table 12. Revision Related Registers

Revision Related Register (Read Only)	DWord Address	Byte Address	Description
GCChipRev	0x0009	0x024	Shows the revision for the chip in BCD. This register has no set reset value. It varies with the implementation.
GCChipDate	0x000A	0x028	Shows the release date for the IP in YYYYMMDD (year/month/day) format. This register has no set reset value. It varies with the implementation.
GCChipPatchRev	0x0026	0x098	Patch revision level for the chip.

3.2.3 APB Interface

For cores with APB the Vivante APB bridge is implemented, and APB signals are provided instead of AHB signals.

Key APB features of the Vivante APB Bridge implementation include:

- AMBA® APB Protocol Version 2.0 Spec (IHI0024C)
- The APB address bus is 32-bit wide.
- The APB write and read data buses are 32-bit wide each. Only 32-bit reads and writes are permitted.
- Support for both read/write wait and no wait states.
- APB error response support for both read and write transactions.
- Partial write transfers are not supported.
- Protection unit is not supported.

3.2.3.1 APB Signal Descriptions

Implementers should refer to the file ***rtl/gcnanoultrav/V2D_VIVANTE_GPU.v*** provided with their specific hardware release package. This file includes a list of the supported APB signals along with their width and IO.

Table 1. APB Pin Descriptions

Signal Name	Width	I/O	Description
PCLK	1	IN	APB bus clock, rising-edge triggered.
PRESETn	1	IN	APB bus reset, active low.
PADDR	32	IN	APB address bus, 32 bits wide.
PSEL	1	IN	Address decode start signal.
PENABLE	1	IN	Enable signal to indicate the second and subsequent transfer.
PWRITE	1	IN	APB in write access when high, and read access when low.
PWDATA	32	IN	APB write data, 32 bits wide.
PREADY	1	OUT	APB ready.
PRDATA	32	OUT	APB read data, 32 bits wide.
PSLVERR	1	OUT	Signal to indicate a transfer failure.
PPROT	3	IN	Protection type. Signals from PPROT are not used. Refer to APB spec chapter 3.5.
PSTRB	4	IN	Write strobes. Must always be4'b1111 to this signal.

3.2.4 Miscellaneous Interface Pinouts

Table 13. Miscellaneous Pins Description

Domain is ACLK unless specified otherwise)

Signal Name	Width	I/O	Description
clk1x	1	IN	Clock source from pin.
CSYSREQ	1	IN	System low-power request
disableRamClockGating	1	IN	Disables clock gating of the RAMs (during BIST)
scanMode	1	IN	Enable/Disable the test mode
CACTIVE	1	OUT	Clock active
CSYSACK	1	OUT	Low-power request acknowledge
DEBUG_OUT	8	OUT	General purpose debug bus
xaq2_intr	1	OUT	Interrupt from graphics core (Domain HCLK)

3.2.4.1 Interrupts

The GPU can send an interrupt signal to the host processor. The signal name is xaq2_intr, and it remains asserted until the host processor clears the interrupt by reading the interrupt acknowledge register (AQIntrAcknowledge). The interrupt is level triggered. The SOC should not use edge to detect the interrupt, as using edge to detect the GPU/VIP interrupt will miss some back-to- back interrupts.

Each bit of AQIntrAcknowledge represents one of the 32 possible events that the GCCORE can signal to the host processor. By setting or clearing the bits of the interrupt enable register (AQIntrEnbl) the programmer can control which of those events will generate an interrupt. The bits of these registers have no predefined meaning, except for bit 31 which is used for AXI_BUS. Instead, all of them are used by and controlled through the VGLite driver. The VGLite driver puts events with different event IDs (0~30) in the command buffer, and the GCCORE will generate an interrupt with the corresponding interrupt bit as set by the event ID in AQIntrAcknowledge. The VGLite driver will process the interrupt accordingly to synchronize the driver/CPU with GCCORE. The meaning of the GCCORE interrupt bits is entirely private to the VGLite driver. The event IDs (0~30) are arbitrarily used by the driver thus the interrupt bits in AQIntrAcknowledge are also arbitrary. SoC designers need only to ensure that the xaq2_intr signal is connected to their interrupt controller. Since AQIntrEnbl is set to 0 at reset, the xaq2_intr signal will never be asserted during the SoC validation phase.

3.2.4.2 Debugging Support

For lab debugging it is desirable to bring out the DEBUG_OUT bus to make it observable from the SoC pins. `DEBUG_OUT[7:0] = {5'b1, idle_pe, 1'b0, idle_fe}` tell if the PE and FE blocks are idle or not. It is also recommended to bring out the AHB bus control signals and data signals to the SoC pins.

The DEBUG_OUT debug idle status signals from the various functional blocks are very useful when the GCCORE is hung or in an unrecoverable state, as they provide visibility into the idle states of the blocks. The signals are intended to be read out as static signals, not sampled actively on the fly. The following options that can be used to handle this bus:

- Don't connect it. This is not recommended.
- Bring it out on the pin(s) as the debug port (recommended). If pin count is a consideration, the port does not need to be 8 pins, design simple mechanism to shift it out.
- Hook it up to one of the AHB registers. But if AHB is also dead, then you cannot read it. So it would be better to go with option #2 above.

3.2.4.3 Disable Ram Clock Gating

The GPU port "disableRamClockGating" allows the customer to control GPU from hardware pin – for instance, from the SoC BIST controller.

As long as a sufficient number of clock cycles have passed between setting this mode and starting BIST operation, this signal can be treated as a static signal for timing purposes.

3.2.4.4 CSYSREQ, CSYSACK and CACTIVE Signals

If these three AXI low-power signals are not used, CYSREQ MUST be tied off to logic one (1'b1). CACTIVE and CSYSACK can float. If these signals are used, they should be connected accordingly (no floating, no-tie-to 1).

Table 14. AXI Low Power Signal Settings (if not used)

Signal Name	Width	I/O	Setting if not used
CSYSREQ	1	IN	Tie to 1
CACTIVE	1	OUT	float
CSYSACK	1	OUT	float

3.2.5 DECNano Interface

DECNano is optional for this IP. If present, DECNano has a simple interface signal design to connect the DEC with its companion Vivante IP. Encoder and decoder have the same port. Characteristics of this interface include:

- There are no configuration registers inside the DECNano IP
- Encoder and decoder have the same ports.
- There is an error signal but not an interrupt.
- Data width is limited to 32 bits.

3.2.5.1 DEC Interface Signals

For pins specific to your package, refer to hardware package file named ***rtl/gcnanoultrav/v2d_dec_nano_dec_core_top.v*** or similar. Custom variants may have different widths for some signals. For DEC cores integrated with Vivante GPU, refer to the release documentation applicable to the release revision you are using, or for greatest accuracy, check the release specific rtl file for specific signal widths.

Table 15. DECNano Interface Signal Descriptions

Note: Encoder and decoder have the same ports. Signals with a width greater than 1 may include a defined parameter such as [NANO_CORE_*_W] in RTL.

Signal Name	Width	I/O	Description
clk	1	IN	
reset_	1	IN	
idle_dec_core	1	OUT	core idle
bypass_i	1	IN	bypass (under development at this writing)
cm_i	2	IN	comp mode. 0: non-subsampled; 1: H; 2: H&V
data_i	32	IN	{A[7:0], R[7:0], G[7:0], B[7:0]}
format_i	1	IN	0: RGBA; 1: XRGB
last_i	1	IN	tile last
ready_o	1	IN	Handshake signal
tilemode_i	1	IN	0: 16x1; 1: 4x4
valid_i	1	IN	Handshake signal
data_o	32	OUT	data out
last_o	1	OUT	compressed tile last
nano_core_error_o	1	OUT	[0] for illegal tilesize; [1] for illegal configuration.
ready_i	1	OUT	Handshake signal
valid_o	1	OUT	Handshake signal

3.2.5.2 DECNano Error Handling

Use the **nano_core_error_o** signal for error handling.

- 0 indicates the tilesize is illegal.
  - The input tile size cannot be larger than or less than 64 Bytes; if this condition occurs, the core will toggle ERROR.
- 1 indicates the configuration is illegal.
  - For example, some **tilemode** and **comp_mode** combinations are not valid, e.g. a tile configured with 16x1 tilemode and HV compression mode will toggle an error.
- Note that bypass is not supported.

3.3 GPU Clocking

GCNanoUltraV Series employs a simple clocking scheme. There are three independent clock domains in the GPU IP:

- Core clock domain, which is derived from the clk1x pin,
- AHB interface clock domain for register access, which is derived from the HCLK pin, and
- AHB interface clock domain for memory access (in 3xAHB designs), which is derived from the ACLK pin.
- APB is an alternate to AHB.

ACLK is the AHB interface clock domain for memory access and *HCLK* is the AHB interface clock for register access. clk1x is the functional clock for the core logic.

Communication between the different domains in the GPU occurs mostly by way of asynchronous FIFOs. All the clocks are asynchronous to each other. There is no communication between AHB clock domains.

The core clock in the GPU core can be scaled down dynamically without reprogramming its source. This scaling is controlled through the use of an internal register.

Table 16. Clock Definitions

GPU Clock Pins	Derived Internal Clocks	Clock Disable Signal	Description
HCLK	-	-	AHB interface clock for register access; provided external to the IP. (or alternately PCLK)
ACLK	-	-	AXI interface clock or AHB interface clock for memory access; provided external to the IP.
clk1x	clk_2d	clk_2d_dis	clk1x is the main core clock; provided external to the IP.

GPU_CORE Clock information:

clk1x the main core clock; provided external to the IP
clk_2d is generated from clk1x

clk_2d_dis Disables clk_2d

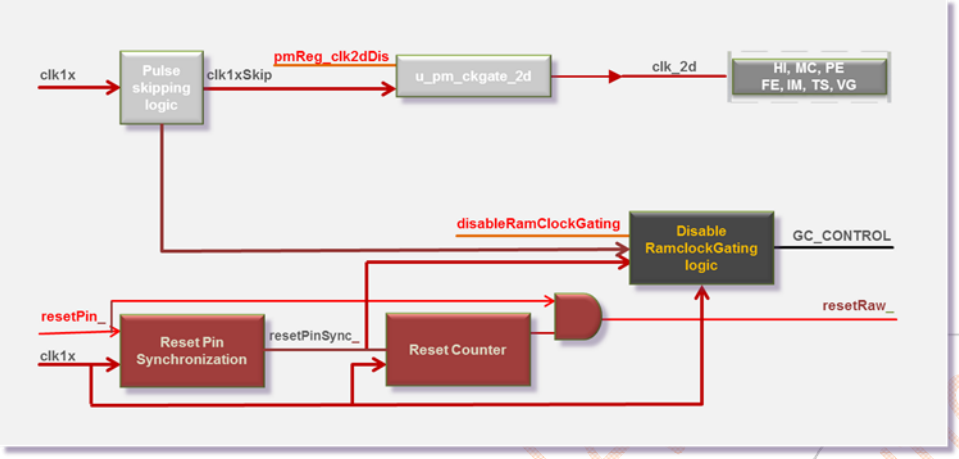


Figure 5. Clock Logic Diagram for clk1x

3.3.1 Clock Gating

Automatic localized clock gating is used throughout the design in order to minimize the dynamic power consumption. Almost all of the registers are gated after synthesis.

3.3.2 Second Level Clock Gating

Block level clock gating is implemented in a majority of the blocks. If a block and the interface to the block are idle, that particular block's clocks will be gated. This is done automatically. This feature can be disabled by software.

3.3.3 Clock Disabling

The core clock can be shut down through software control by setting the proper register bits.

3.4 Reset

Reset input is synchronized to the core clock domain by a dual flip-flop synchronizer. The resetPin_, HRESETn or PRESETn, and ARESETn signals should be asserted together for a minimum of 32 core clock cycles, using the slowest of the three clocks.

After deassertion, wait for 128 cycles of the slowest clock before starting any activity on AHB/APB.

All registers use synchronous reset flip-flops except three in the HI module to minimize noise sensitivity on the reset line. Note, however, that this means the clocks must be running in order to reset the state of the chip. Assuming the clock is generated from a PLL, the diagram in the figure below illustrates the proper reset sequence.

HRESETn/PRESETn and ARESETn signals are used to reset the AHB/APB interface blocks. For designs with 3xAHB, 2xAHB are driven by ACLK and ARESETn. The resetPin_, HRESETn/PRESETn, and ARESETn signals can be tied together to facilitate synchronization on assertion and deassertion.

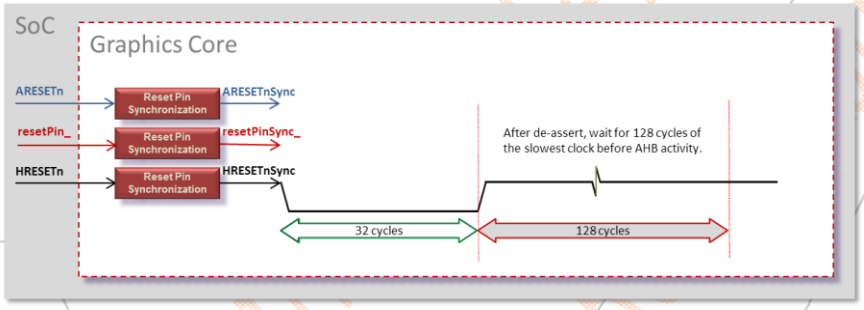


Figure 6. Reset Sequence Diagram

For reset signal assertion and de-assertion to work properly:

- **All reset signals should assert and de-assert together.**
- For Core reset:
  - Make sure all the GPU blocks are idle.
  - Clk1x needs to be running when asserting and de-asserting resetPin_.
- For AHB/APB reset:
  - Make sure no AHB transaction is in progress.
  - HCLK/PCLK needs to be running when asserting and de-asserting HRESETn/PRESETn.
  - ACLK needs to be running when asserting and de-asserting ARESETn.
- Wait for 128 cycles after deassertion.
- The de-assertion should adhere to the 32-cycle assertion wait time.

3.4.1 DEC Reset

The signals **reset_** is used to reset the DEC core clock.

The **reset_** signal must be asserted for a minimum of 32 core clock cycles.

The signal **reset_** should be de-asserted for 32 cycles. If the design includes a 2-cycle synchronizer, then wait for 130 cycles before the DEC starts to work.

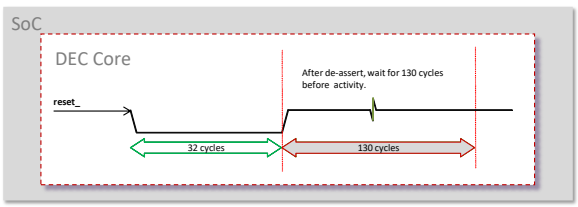


Figure 7. DECNano Reset

3.5 Power Off and Power On Sequences

To properly power OFF and power-ON (wake-up) the GPU, both Vivante and SoC software must follow a pre-defined protocol and sequence as defined below.

3.5.1 Power Off

The power-off sequence for the GPU is as follows:

- SoC SW issues a request to go into “OFF” state, or the GPU internal timer triggers a transition into the “OFF” state.
- Vivante SW performs all the necessary preliminary steps required to put the GPU into the “OFF”state.
- Vivante SW issues a callback to gate the GPU clock and GPU power.
- SoC SW initiates a power down sequence to the SoC controller HW.
- SoC HW asserts an isolation signal to turn on isolation from the GPU.
- SoC HW asserts a power gate signal to turn off power to the GPU.
- Upon return of the HW acknowledgement, SoC SW returns call-back status to Vivante SW.
- Upon receipt of a successful callback, Vivante SW puts the GPU into the “OFF” state.

3.5.2 Power On

The power-on sequence of the GPU is as follows:

- SoC SW requests a transition from the “OFF to the “ON” power state.
- Vivante SW issues a call-back to enable clock and power for the GPU.
- SoC SW initiates a power up sequence to the SoC controller HW.
- SoC HW de-asserts the power gate signal to turn on power to the GPU.
- Upon HW acknowledgement, SoC HW turns on the clocks to the GPU.
- SoC SW performs an AHB domain reset.
- SoC HW de-asserts the isolation signal to turn off isolation from the GPU.
- SoC SW returns call-back status to Vivante SW.
- Upon a successful return, Vivante SW performs an AHB write that will soft reset the core domains.
- Vivante SW enables FE and waits for commit.

3.6 Low Power State Control

This GPU series supports Vivante's proprietary Power Management scheme, controlled through a register which is accessible from the AHB.

3.6.1 Proprietary Power Management

There are two registers (**AQHiClockControl** and **AQHiIdle**) in the HI module dedicated for power management. Software can control power management through these registers. No automatic sequencing is involved in this scheme. The software needs to make sure that a block is in idle before shutting down the clock to the block.

This scheme also offers a register interface to the clock scaling mechanism available in the graphics core.

3.6.2 Clock Control Register

All clock control registers reside in the host clock domain. We need to be able to access or re-program these registers if a PLL fails for any reason. Also, during power-down modes, waking-up the different blocks will not be a problem.

Table 17. AQHiClockControl Clock Control Register Select Bits

Bit Field	Type	Name	Description
1	R/W	CLK2D_DIS	Software core clock disable signal. The AHB master interface clock is the only block not stalled at that point. Reset value = 1'b0
8:2	R/W	FSCALE_VAL	Frequency scaling value. Reset value = 7'd64 (no pulse skipping)
9	R/W	FSCALE_CMD_LOAD	Frequency scaling load command. When writing a "1" to this bit, it updates the frequency scale factor with the value FSCALE_VAL[6:0]. The bit must be set back to "0" after that. Reset value = 1'b0

3.6.3 Core Clock Frequency Scaling Logic

A basic period of 64 clock cycles is chosen as the basis for clock skipping. Any number of clocks within that 64 clock period can be skipped. This yields a fine granularity of 1.6% (1/64). The scaling factor is a 7bit value which represents the number of clocks NOT to skip (the reset value is 64). Pulses will be skipped uniformly along the 64-cycle period to smooth out the power demand of the power supply.

Scaling is controlled by software through the use of control registers. The software determines the number of cycles not to skip within a 64-cycle period. The frequency scaling value is double-buffered as well, and it is updated at the beginning of a 64-cycle sequence.

The following two diagrams describe the pulse skipping logic and show an example of pulse skipping for the case of a 16 cycle period. (The actual period is 64 cycles, but 16 were chosen for ease of illustration).

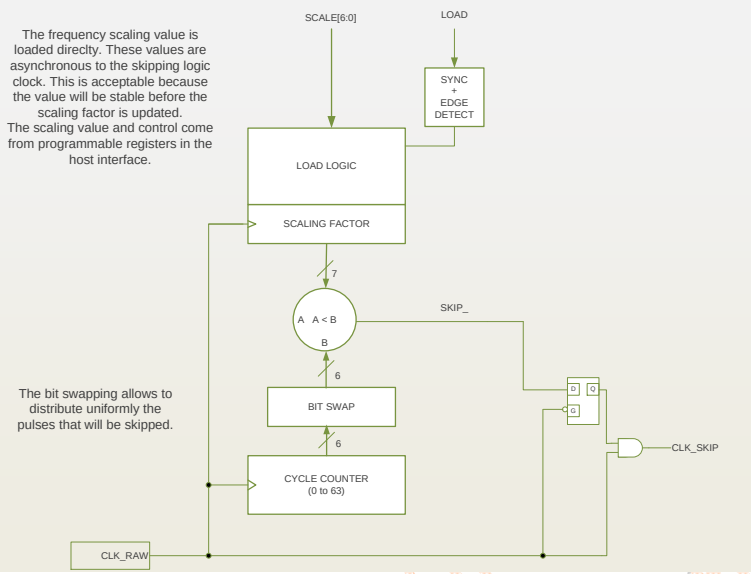


Figure 8. Pulse Skipping Logic

Example of pulse skipping for 16-cycle period

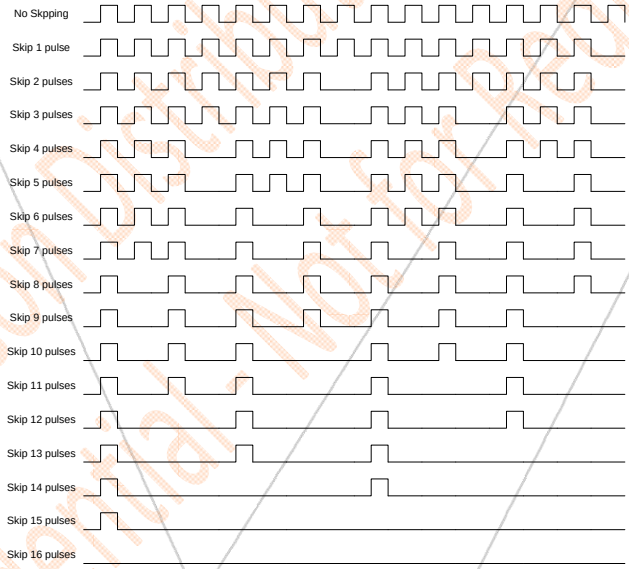


Figure 9. Pulse Skipping Timing Diagram

3.7 Integrating the Clock Gating Components

There are some hard-coded ICG (Integrated Clock Gating) cells in the RTL code which are technology specific. In the ***CLOCK_GATER.v*** file of the ***rtl/clockgater*** directory, Vivante provides the technology-independent default. SoC integrators or front-end engineers must edit this file and choose an integrated clock gating cell to match their target library. If this step is not done at the RTL simulation stage, then the PD engineers will need to do the clock gating cell setup to ensure a successful synthesis run. Edit the file ***rtl/GC_CG_MOD.v*** to choose an integrated clock gating cell from the target library for synthesis. An example of the clock gating cell from the SEC14LPP standard cell library is provided for your reference. Shown below is the code section from ***GC_CG_MOD.v*** where editing needs to be done.

```
`ifdef AQ_SEC14LPP_A9TR_C14_LIB
    PREICG_X4N_A9PP84TR_C14 GC_CG_INST(
        .E (enable),
        .CK (ck_in),
        .SE (test),
        .ECK (ck_out));
    `else
```

The correct clock gating cell from the target library should be instantiated to ensure a successful synthesis run. The synthesis tool should not be used to synthesize this logic from the *`else* section with the default one.

The correct clock gating cell for the module can be found by looking for the Synopsys *.lib of the target library for an integrated clock gating cell with one of the following attributes:

```
latch_posedge_postcontrol_obs
latch_posedge_precontrol
latch_posedge_postcontrol
```

An example from the TSMC28HPM library is shown:

```
cell(PREICG_X1N_A9PP84TR_C14) {
    area : 0.72576 ;
    cell_footprint : PREICG_X1N ;
    clock_gating_integrated_cell : "latch_posedge_precontrol" ; Memory Macro
        Setup
…
```

3.8 Integrating the Embedded Memory Components

2-port register files are used throughout the IP as temporary storage, FIFOs and caches. Some of the memory blocks require byte write capability. The IP is developed using behavioral memory models. Depending on the available memory configurations, the synthesizable memory blocks are added by specific *ifdef* Verilog macros. We provide detailed memory information in the "…/doc" directory of the hardware implementation (physical) package.

memory_instance_list:

- CELLNAME –cell name
- WORD - this memory word number
- BIT - this memory bit number
- AREA - this memory area number
- MASK - if this memory requires byte or bit masking
- INSTANCE NAME - this memory hierarchical instance name in this GPU
- CLOCK DOMAIN - clock domain of this memory (Note: You may need to check which delivered timing constraints to get if this clock is sync or async or same phase multiplied frequency with the core input clock.)

Generally, we do not allow simultaneous read/write operations on the same address line and same bit/byte. But we do allow simultaneous read/write operations on the same address line and different bit/byte.

3.9 VGLite Driver and API Support

The VGLite driver and API are designed to provide a simple and small footprint interface to the hardware. For this reason the driver and API are already optimized for efficient behavior and the software does not provide full flexibility for all functionality available in the GCNanoUltraV hardware. For example, there are no APIs for MMU management, clock gating, frequency scaling, or low power control.

3.10 DECNano Lossy Data Compression Overview

The DECNano compression module is supported for some series variants.

3.10.1 Supported Formats, Tile Modes and Compression Mode

The DECNano Series supports the following formats, tile modes and Compression modes.

- Formats are 32 bit-per-pixel ARGB8 and XRGB8, specified in signal **format**. A/X is in the upper bits and B in the lower bits.
- Tile mode is either linear 16x1 or tiled 4x4, as specified in signal **tilemode**, where 0=16x1 raster line and 1=4x4 tile. Each tile occupies 64 Bytes. If a tile is any other size, a **Nano_core_error_o**=0 signal will be generated.
- One of three compression modes is specified in the signal **comp_mode**, where 0=no subsample, 1=Horizontal sample of 4:1, or 2=Horizontal and vertical sample of 1. A **Nano_core_error_o**=1 signal will be generated if a configuration attempts to configure a tile with a **tilemode**=0: 16x1 and **comp_mode** =2: Horizontal and Vertical Sampling.

Table 18. Supported RGB Formats

32 bit formats	31	First Pixel=31:0						24	23						16	15							8	7							0
A8R8G8B8	A	A	A	A	A	A	A	A	R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B
X8R8G8B8	X	X	X	X	X	X	X	X	R	R	R	R	R	R	R	R	G	G	G	G	G	G	G	G	B	B	B	B	B	B	B	B

Table 19. Compression Ratios per Compression Mode

Color Subsample / Compression Mode	ARGB	XRGB
No sample (raster or tile)	1.6:1	1.6:1
Horizontal Only (raster or tile)	2.1:1	2.1:1
Horizontal and Vertical (tile only)	2.6:1	3.0:1

Table 20. Tile Organization

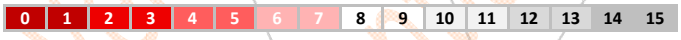
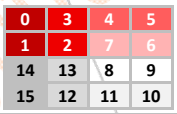
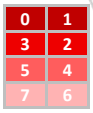
Tile Mode	Compression Mode Subsampling	Compression Ratio for ARGB	Compression Ratio for XRGB
16x1 Raster 	0: No sample	1.6:1	1.6:1
16x1 Raster 	1: Horizontal Only	2.1:1	2.1:1
4x4 Tile 	0: No sample	1.6:1	1.6:1
4x4 Tile 	1: Horizontal Only	2.1:1	2.1:1
4x4 Tile 	2: Horizontal and Vertical	2.6:1	3.0:1

Table 21. Format, Tile Mode and Compression Mode Summary for DECNano

Format	Tile Mode tilemode 0=16x1; 1=4x4	Compression Mode comp_mode 0= ; 1=no sample; 2=Horizontal 4:1; 3=Horizontal&Vertical 1
ARGB8 (format=0) or XRGB8 (format=1)	16X1	No sample
		H
	4X4	No sample
		H
		HV

3.10.2 Considerations

The DECNano Series is a small footprint design intended to provide basic lossy compression capabilities. As such, it provides a limited subset of formats, tile modes, compression modes and advanced features. Note that:

- Compression and decompression are supported, but some variants may support only one.
- Lossless compression is not supported. Only lossy compression/decompression is available.
- The tile to be compressed must be 64 Bytes. If not, an error will result.
- The decompressed tile must be 64 Bytes. If not, an error will result.
- No YUV format support is provided. Only A/XRGB8 formats are supported.
- A2RGB10 is not supported.
- Bypass is not supported.
- Burst size is 4 bytes.
- Throughput is 1 RGBA or RGB pixel per cycle.

4 Verification

4.1 Using the Portable Test Bench

Vivante provides a portable test bench and a collection of basic tests with the RTL release package. Customers can use the bench and the tests to confirm the correctness of the RTL model without having to integrate the GPU into their test environment. The test bench includes a memory model implemented in C, simple models of the AHB buses, and a command interpreter that drives the testing.

4.2 Organization of the Tests

All tests contain multiple text files; each text file is the memory image for a data buffer. The Vivante GCCORE uses a number of memory buffers for different purposes. Each test program uses at least one buffer to read the state loads and commands. This buffer is called the Command Buffer, and it is where all of the GCCORE states and commands are stored in memory.

When GCCORE begins test execution, it reads the Command Buffer from the beginning until it encounters an END command. The execution is started by two register writes through the AHB bus. The first register write programs the starting address of the Command Buffer, and the second register write tells GCCORE to begin running the test program. When the test finishes, all the idle signals will become high. By polling the idle register through the AHB bus the user can know when the test program is finished. (This can also be done by using an event/interrupt sequence).

Depending on the function of the test program, it may use other buffers.

The common types of test buffers are:

Table 22. SoC Test Data Buffers

Data Buffer .txt filename	Buffer Name	Description
call_cmd_buffer.txt	Call Command Buffer	This file provides separate access to the CALL Command to allow for controlled testing of CALL and RETURN.
cmd_buffer.txt	Command Buffer	Contents of the Command Buffer the program uses. This file contains the contents of the COMMAND BUFFER the test program uses.
idx_buffer.txt	Index Buffer(s)	If the test program uses the DrawIndex command, this buffer specifies the order in which it will draw the vertices.
image_buffer.txt	Image Buffer	If the test program uses image, this buffer is used to store the image data for single image modes, or for double image mode, this buffer stores data for the first image. The image buffer size depends on the image format, image width, image height and if it contains a double-image or not.
image0_buffer.txt	Image Buffer	For double image modes, if the test program uses image0, this buffer is used to store the image0 data.
paint_buffer.txt	Paint Buffer	This buffer stores paint data.
result.txt	Result Frame Buffer	This file is the contents of the frame buffer after the test is run. Compare the contents of your frame buffer to see if the test program worked properly.
scissor_buffer.txt	SCISSOR BUFFER	If the test program uses scissor, then this buffer contains the scissor data.
src_buffer.txt	Source Buffer(s)	Target memory to which the test program writes its source values.

4.2.1 Running the Tests

After unpacking the release files, the user can run the tests by going into the "dv/test" directory and typing "make test." All files needed to run the tests are included in the package. The makefile uses the gcc compiler and assumes it is in the user's path. A PASS/FAIL will be given for each test, as well as a final summary. Users can run individual tests by typing "make <test name>" in the same directory, where <test name> is the name of the directory of the test you want to run. Customers can get the current PASS/FAIL results of all tests by typing "make results" and can remove all tests' results by typing "make clean."

4.2.2 Compiling the Verification Package

The test bench uses NC by default. Customers who want to specify NC-Verilog can do so by adding the option "SIMULATOR=nc" to the make command.

To run with NCSim, type: `make <test name> SIMULATOR=nc`

Before compiling under VCS, make sure the VCS_HOME environment variable points to your VCS installation. For example (from the dv/README file), your setting should be something like VCS_HOME=/home/tools/cad/synopsys/VCS/vcs-mx_vK-2015.09-SP1.

To run with VCS, type: `make <test name> SIMULATOR=vcs`

4.2.3 Design Defines

The correct "Design Defines" must be set to read the RTL Verilog files when running simulation. See the table below for a listing of some key verification defines and their description.

Table 23. Verification Design Defines

Define (Verilog `define)	Description
AQ_SEC14LPP_A9TR_C14_LIB	Use SEC 14nm LPP technology specific ICG cells. Reference files: Synopsys dotlib Modify in file: ***rtl/gcnanoultrav/GC_CG_MOD.v***
AQ_SEC14LPP_RAM_MODEL	Use the SEC 14nm LPP technology-specific memory library. Reference files: ***rtl/rams/RAMxxx.v*** Modify in file: ***rtl/gcnanoultrav/gc_xx_xxxx_wrapper.v***
AQ_TIMESCALE_ON	Use in simulation.
FPGA	Use when compiling to Xilinx FPGA.
Define Cancellation (Verilog `undef)	
`undef <DEFINE> as specified in **rtl/undef/<prefix>_vsi_undef.v**	Cancels previously defined text macro definitions. NOTE: Please compile ***rtl/undef/<prefix>_vsi_undef.v*** as the last Verilog file for this RTL.

4.3 Using the Integration (SoC) Tests

Vivante provides a suite of tests that customers can use to test the integration of the GPU into their designs. The tests are included in a directory in the release package with the RTL model. Users should note that these tests are low level, and in real-life the programmer will not be required to interact with GPU this way. Instead the user can program the GPU with a high-level language that will then be translated into low level transactions by the driver and HAL layer.

The tests include all the necessary inputs in the form of memory images. These memory images are provided as text files. The tests also include a result file that contains the final contents of the frame buffer and is used to check the results generated by the test.

4.3.1 Contents of the Test Directory

Each test is stored in a folder labeled with the test name. In each folder there will be at least a cmd_buffer.txt and result.txt file. Additional files may include src_buffer.txt, tlb_buffer.txt, frame_buffer.txt, README.txt and clear_buffer.txt.

Availability of tests will vary between variants and from package to package.

Refer to the ***SOC_Tests/GCNANOULTAV_README.txt*** file for information about the tests in your package.

Table 24. Integration SoC Test Categories for GCNanoUltraV Vector Graphics

Test Category	Feature Code *(used in Table 19)*	Description
Blend	**B**	Test all blending modes
Clear	**C**	Clear the screen, optionally with scissoring
Color transform	**T**	Test color transformation
Draw	**D**	Exercise path, stroke, and fill modes
Image	**I**	Test all image modes, including perspective correction
Mask	**M**	Test masking
Paint	**P**	Test paint modes: solid, linear gradient, radial gradient, and pattern
Pixel format	**F**	Test pixel format conversions

Table 25. Sample Integration SoC Tests for GCNanoUltraV

(test list subject to change; tests may vary release to release and per feature support levels in the IP)

Test Name	Test Feature Code	Description
blend10	B C D P	A test that draws a picture to test VG_BLEND_ADDITIVE.
call		A test that verifies the call return function.
draw_index8_Image	D C I	A test that draws an Index8 image.
imageA4	D C I	A test that for A4 image.
normalimage	B C D I	A test that draws a picture to test vgDrawImage with VG_DRAW_IMAGE_NORMAL.
perspectiveimage	B C D I	A test that draws a picture to test vgDrawImage with VG_DRAW_IMAGE_NORMAL and perspective.
solidclear	C	A test that draws a picture to test clear with a simple color.
tess0	C D P	A test that draws a star.
tess0_call	B C D P	A test that draws a star with call-return command.
tess7	B C D P	A test that draws a star using even-odd rule.
tiger	B C D P	A test that draws an image of a tiger out of range.

4.4 Initialization of the Tests

AHB Register writes are required only to load the Command Buffer address. Once the Command Buffer address is loaded and the program starts (by a second register write), all other buffer addresses, if they are used, will be read by the state loads given in the Command Buffer program.

The GPU Core accesses memory through an AHB bus, and thus the test environment should have a way of initializing the memory either directly through PLI function calls, or through the CPU which is also part of the test bench. The contents of each text file provided in the test directories should be written to memory before the GPU Core starts execution. In the text file that corresponds to the memory image of the Command Buffer (cmd_buffer.txt) symbolic names are given for the state loads that specify the addresses of all other buffers. For example, the state load for designating the start address of the tessellation buffer is given as:

```
0x30010A35,
TS_BUFFER_ADDR
```

The user must edit the cmd_buffer.txt file and replace the symbolic names with the actual address values. The figure below shows and example arrangement of the cmd_buffer.txt file.

After editing cmd_buffer.txt, the user should load it and the contents of the other buffer files into the test bench memory. The locations of the different buffers do not matter as long as the symbolic addresses in the Command Buffer are replaced with the correct values and none of the buffers overlaps with any other. The figure below shows one possible arrangement of the data in the memory.

Once all the buffers for the test are written into the memory, two register writes are necessary to start the DMA.

First, the address of the command buffer is written to register 0x500. The second write loads the data number of the command into register 0x504, which signals the beginning of DMA accesses from GCNanoUltraV core. Note that these register addresses should be offset with the base address of the GPU core. For example, if you placed the GPU at address 0x80000000, then the write to register 0x500 will be at memory address 0x80000500.

The tests contain multiple text files, one for each buffer they need. In addition to allocating space for these buffers, the tester should allocate TS BUFFER, COUNT BUFFER and CMD BUFFER for all tests. A result.txt file is provided to compare with the final output contents of a given test, as written to the FRAME_BUFFER. All tests assume a 256x256 screen with 4 bytes pixel data. Thus a memory of 256x256x4 bytes should be allocated for FRAME_BUFFER. The size of TS BUFFER is 0x40000 and the size of COUNT BUFFER is 0x4000. All other buffer sizes are dictated by the contents of these buffers. TS BUFFER and COUNT BUFFER contain the tessellation data for the picture drawn by the test program.

A test with _call in its name is designed and provided to specifically test the CALL and RETURN command flow. These tests will include a file call_cmd_buffer.txt. In the cmd_buffer.txt file there will be a reference to CALL_COMMAND_ADDR. This address should be allocated by the user, and it should provide the specific address of the CALL command. The reference evokes a macro. Once it executes the evoked CALL command, it will call other commands and execute them. When finished, it will return.

README files provided in the SOC_Tests folder and/or with some individual tests provide additional test instruction that may not be included in this discussion.

The following figure shows a block diagram of a basic test environment for the GCNanoUltraV.

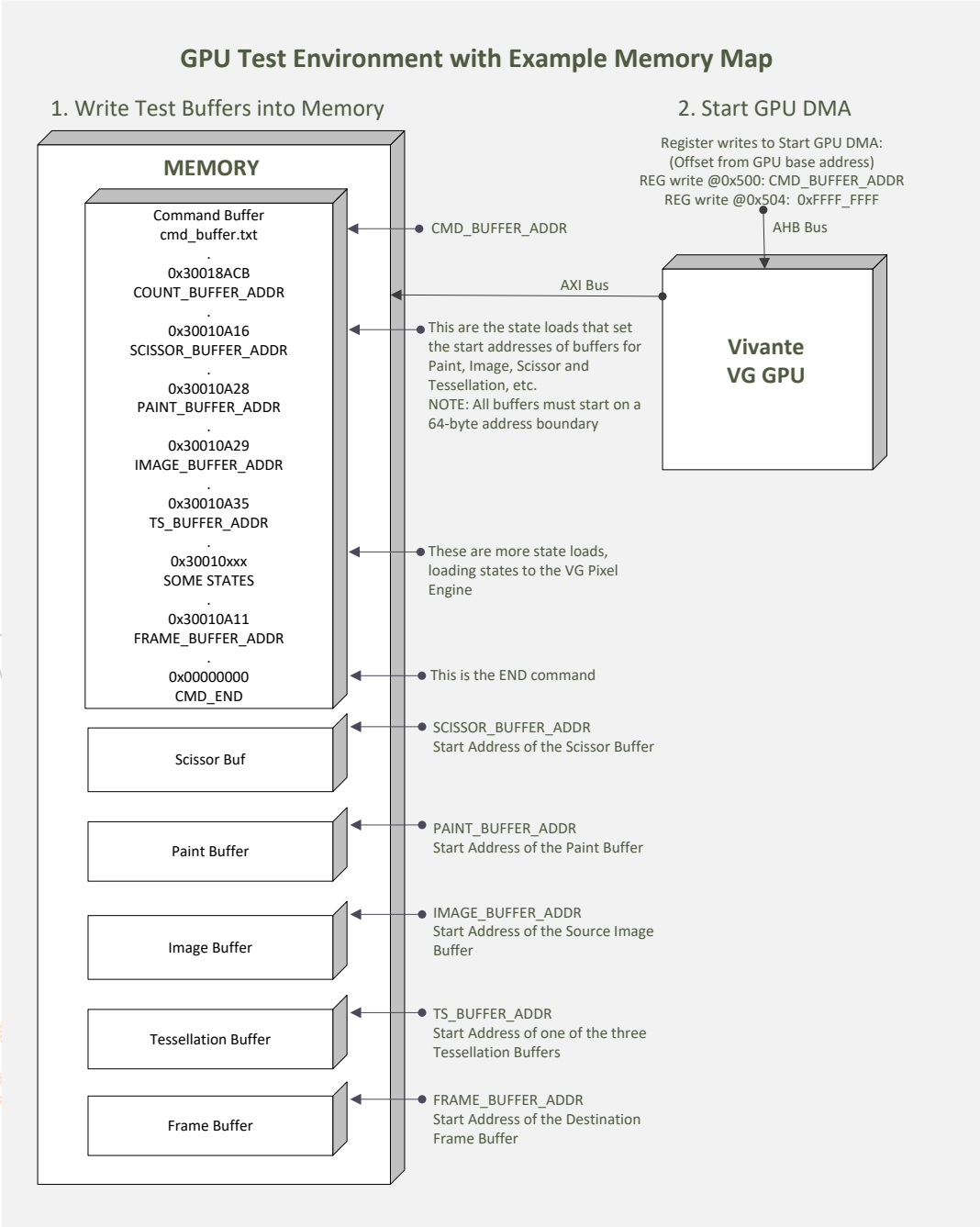


Figure 10. Example Memory Map for Verification Tests

4.5 Troubleshooting

4.5.1 Buffer Addresses

Note that this GCCORE can handle both physical addresses and virtual addresses. When the MSB (bit 31) of a memory address is set to “1” the GCCORE assumes that the address is a virtual address. If the test environment has a large physical memory, addresses above 0x8000_0000 will have “1” in bit 31. The GCCORE will treat any test data buffers in this physical address space as having virtual addresses. (Or you can move your buffers to addresses below 0x8000_0000.)

4.5.2 Enable Interrupt

In order to generate an interrupt at the end of each test, enable the AQ_INTR_ENBL register before starting the GCNanoUltraV by AHB write data 0xFFFFFFFF to address 0x014 (AQ_INTR_ENBL_Address).

4.6 Ram Wrapper Testing

If present, the **wrapper_test** directory in the hardware package includes a python script (ram_wrapper_bench_gen.py) which serves as a test harness for memory wrapper testing. The file Readme.WrapperTest.txt provides release accurate instruction for setting up the appropriate environment and running the script.

To use the script:

- Set up your ncsim simulation tools
- Create a working directory <work_dir>
- Move or copy the rtl/ folder found in the hardware package to <work_dir>
- Create (mkdir) a wrapper_test/ sub-directory under <work_dir>
- Move or copy the run script ram_wrapper_bench_gen.py into the new wrapper_test/
- Do: cd wrapper_test/
- Execute: ./ram_wrapper_bench_gen.py
- To check the run status, do these in the wrapper_test/ dir:
  - grep PASS wrapper_bench/sim/*/simulation.log
  - grep ERROR wrapper_bench/sim/*/simulation.log
- When all the tests are done, result will be printed on the screen, and a result file named wrapper_status will be created in the folder wrapper_bench.
- All the bench will be in a folder named wrapper_bench. If you need to rerun any one of the the tests, cd into it and do Make [wrapper_name],

5 FPGA Prototyping

This section describes the FPGA bring up using SOC tests. SOC tests can also be used to check the SoC environment and the FPGA synthesis of the GPU. These tests do not require any kind of operating system, and they directly run on the GPU. The necessary data and commands needed for the GPU to run the tests are loaded to the system memory by the system CPU. Then, the GPU is given the base address of the command buffer, and it begins fetching commands and executes the test.

5.1 Generating the Bit File

In order to verify RTL code on the FPGA, we need to map the RTL to the FPGA logic array. We introduce a reference flow and use “Synplify+Vivado” EDA tools.

5.1.1 Files Needed

- RTL
- Header files

5.1.2 FPGA Device-related Files

- The DCM file is specific to the given Xilinx FPGA being used. It is essential to generate such a cell that corresponds to the FPGA device. During generation of the DCM cell, make sure the frequency of this cell corresponds to the real clock frequency of the FPGA.
- FPGA cell library
- Synplify project file, constraints file
- Xilinx ucf (user constraints file)

5.1.3 Generating the Bit File

There are some things to remember when generating the bit file:

- Make sure the frequency definition of DCM matches the real clock frequency on the FPGA platform.
- Make sure the timing constraints are good enough, especially the I/O timing constraints you specify when using the Vivado tool.

Bit file generation is performed in the following two steps:

- Generate a netlist file using the Synplify tool.
  - Create a project, and choose the correct FPGA device.
  - Add HDL define “FPGA” on this project.
  - Read the RTL code into Synplify.
  - Create timing constraints for this project.
  - Run synthesize to produce a netlist file *.edf.
  - Generate the bit file with the Vivado tool.
- Create a project, and choose the correct FPGA device.
  - Read in *.edf file which is generated by Synplify.
  - Set constraints, including those for timing and pin placement.
  - Run bit file generation to produce the *.bit file.

5.2 Testing the Bit File using SOC Tests

SOC tests listed in Section 4 can be used to test the FPGA functionality.

6 Integration Check Points

This list provides key check points indicating SOC vendor activities which may be performed during their integration process.

SoC Product:_____

Vivante GPU Core and Tag or Rev Number:_____

Reviewer Name and Date:_____

	Check Points for SOC Vendor's Integration of GPU	Result: Pass or Not
1	**DV TESTS PASS**	
1-1	All DV tests pass in DV simulation with updated memory lib	
2	**SoC TESTS PASS IN SoC SIMULATION**	
2-1	All SoC tests pass in SoC simulation with updated memory lib	
3	**SoC TESTS PASS IN GATE LEVEL SIMULATION**	
3-1	All SoC tests pass in Gate Level simulation without SDF.	
3-2	All SoC tests pass in Gate Level simulation with SDF.	
4	**PHYSICAL DESIGN REVIEWS:**	
4-1	Types of reviews to include: Synthesis, Floorplan, P&R, Clock Tree, Timing, Power, Routing and Formal Verification, etc.	
5	**OPTIONAL CHECKPOINTS FOR FPGA:**	
5-1	All SoC tests pass on FPGA	
5-2	All SW/Driver tests pass on FPGA	

Additional Notes:

Document Revision History

This section describes differences between document revisions.

Note: This document is not necessarily updated for each patch or minor revision. The information in this document tends to be stable across a revision (nnn) series.

Document Revision	Doc Date	Compatible Hardware	Description
0.88	2023-02-23	GCNanoUltraV	Update document template and related format adjustments, Section 3.1: Add DECNano optional module. Section 3.2.5: added for DEC signals Section 3.4.2: added for DEC reset Section 3.10: added for DEC overview.
0.87	2022-12-06	GCNanoUltraV	Section 5.1: Replace ISE with Vivado tool.
0.86	2022-11-08	GCNanoUltraV	Section 4.2 Table 17: remove tess*.txt. Section 4.4: update TS_BUFFER_ADDR; remove TSLEVEL1 TSLEVEL2 buffers; add COUNT BUFFER. udate tessellation buffer size to 0x40000; Figure 9: add COUNT BUFFER; update TS_BUFFER_ADDR. Document template updated and related adjustments.
0.85	2022-05-16	GCNanoUltraV	Section 3.2,2.2: Remove AQMemoryFEPageTable Miscellaneous format refinements.
0.84	2022-02-03	GCNanoUltraV	Legal Notices, General: Update branding layout to include VeriSilicon. Miscellaneous format refinements. Section 3.5: Refine reset description and for description for 3xAHB reset.
0.83	2022-01-12	GCNanoUltraV	Section 3. Add AXI and APB detail in subsections 3.2.1 and 3.2.3.
0.82	2021-12-08	GCNanoUltraV	Section 3.4.1: update second paragraph to expand EventID detail.
0.81	2021-11-12	GCNanoUltraV	Various: remove F variant. Section 2 and subsections. Update to match package. Section 2.1 and 4.1.3: Add undef Section 2.5: and 4.6: Add wrapper test. Miscellaneous refinements.
0.80	2020-07-13	GCNanoUltraV V2 pre-release	Initial Edition