Intel® Iris® Xᵉ MAX Graphics Open Source

Programmer's Reference Manual

For the 2020 Discrete GPU formerly named "DG1"

Volume 7: Memory Cache

February 2021, Revision 1.0

Notices and Disclaimers

Intel technologies may require enabled hardware, software or service activation.

No product or component can be absolutely secure.

Code names are used by Intel to identify products, technologies, or services that are in development and not publicly available. These are not "commercial" names and not intended to function as trademarks.

Customer is responsible for safety of the overall system, including compliance with applicable safety-related requirements or standards.

No license (express or implied, by estoppel or otherwise) to any intellectual property rights is granted by this document.

The products described 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.

You may not use or facilitate the use of this document in connection with any infringement or other legal analysis concerning Intel products described herein. You agree to grant Intel a non-exclusive, royalty-free license to any patent claim thereafter drafted which includes subject matter disclosed herein.

Intel disclaims all express and implied warranties, including without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement, as well as any warranty arising from course of performance, course of dealing, or usage in trade.

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.

Table of Contents

Memory Cache

This section describes the GFX L3 Cache, which is a large storage that backs up various L2/L1 caches in the GPU's internal units.

L3 Cache

This is the volume describing the L3/URB/SLM.

L3 Blocks Overview

L3 is formed via some number of logical banks that are identical to each other. The major blocks in each logical bank are:

- L3 Cache Arrays & Controller
- Super Q and related data buffer
- Ingress queues and related CAMs with arbitration
- Atomics Block/SLM pipeline & crossbar for data routing

Size of L3 Bank and Allocations

Multi-Bank Allocation Options with Tile Cache and Command buffer support

DG1 support for Large Cache

DG1 support will have two basic changes:

- URB allocation is no longer programmable via L3 but rather a fixed allocation of 768KB for entire DG1. S/W does not need to program a certain allocation for DG1 anymore. Note that dual-context mode can still a portion of URB for the second context when enabled to do so.
- L3 cache size has been increased to 16MB total storage (still organized as 8 banks of 2MB each)

L3 Allocation programming (KBytes per bank)								
URB	**Rest (DC+RO)**	**DC**	**RO (I/S/C/T)**	**Z**	**Color**	**Unified Tile Cache**	**Command Buffer/ State**	**Sum**
96KB (fixed)	0 to 2MB in increments of 32KB	0 to 2MB in increments of 32KB	0 to 2MB in increments of 32KB	0 to 2MB in increments of 32KB	0 to 2MB in increments of 32KB	0 to 2MB in increments of 32KB	0 to 2MB in increments of 32KB	2048KB

Note : The granularity of 32KB delta for each section arises from the fact that the L3 is implemented as a sectored cache with 2 ways per sector. Due to this, the number of ways programmed for each section of the L3 cache should be an even number. The programming of the **L3ALLOCREG** to configure the

sections should be determined based on the size of the cache per way. The total number of ways implemented in the design is available in the **L3 Parameter Information register**.

However, several restrictions apply.

- It is not allowed to allocate the entire cache to DC (Data space) with 0KB for reads.
- It is not allowed to have Rest and DC to be "0KB" at the same time. Cache requires either Rest or DC to have at least non-0KB allocation.
- It is not allowed to have Rest and RO to be "0KB" at the same time. Cache requires either Rest or RO to have at least non-0KB allocation.
- State is now stored as part of Command Streamer allocation. If specific allocation is 0KB, it will be placed in the Rest section.

L3 Allocation programming (KBytes per bank)									
Config		Rest (DC+RO)	DC	RO (I/S/C/T)	Z	Color	Unified Tile Cache	Command Buffer	Sum
0 (def)		2048	0	0	0	0	0	0	2048
1		1024	0	0	0	0	992	32	2048
2		0	1024	992	0	0	0	32	2048

Preferred configuration setting will be Config#0 with flat cache.

Memory Object Control State on Cacheability

This 7-bit field is used in various state commands and indirect state objects to define L3/LLC/eDRAM cacheability, memory type, and graphics data type for memory objects.

Note that memory type information from state is used for non-IA compatible paging structures (legacy context). For new context definition where IA compatible (IA32e) paging structures are used, memory typing follows the IOMMU defined structures.

MOCS[6:1] in L3 is used as an index to a set of programmable tables starting with address xB020h. GFX Software can set up the tables as part of the h/w context, and program various index values in surfaces to point to a table that best suits for that particular surface.

Atomics

An atomic operation may involve both reading from and then writing to a memory location. Atomic operations apply only to either u# (Unordered Access Views) or g# (Thread Group Shared Memory). It is guaranteed that when a thread issues an atomic operation on a memory address, no write to the same address from outside the current atomic operation by any thread can occur between the atomic read and write.

If multiple atomic operations from different threads target the same address, the operations are serialized in an undefined order. This serialization occurs due to L3 serialization rules to the same address.

Atomic operations do not imply a memory or thread fence. If the program author/compiler does not make appropriate use of fences, it is not guaranteed that all threads see the result of any given memory operation at the same time, or in any particular order with respect to updates to other memory addresses. However atomic operations are always stated on a global level (except on shared local memory), when atomic is operation is complete final result is always visible to all thread groups.

Atomics features:

- Double size operands where 8B atomic operations are introduced for 64-bit data types. There is also an addition 16B "atomic_cmp/wr16B" to the table. All these new operands apply to global memory only.
- Floats for 4B/8B accesses, only floating-point adder is used.
    - (*note: if L3 atomics are disabled, floating point atomics can be used*)
- Move global atomic ops to L3 (keep the GTI support for non-coherent L3 mode) and share same atomic OPs as SLM.
    - GT3 can process up to 192 IA coherent atomics per cycle.
- Hardware improves the performance of atomics to the same cacheline. Specifically, back-to-back atomics to the same memory location are done without the need to re-fetch the result of the previous atomic from L3. This should increase bandwidth for atomics to the same location significantly over previous.
- Single-Precision FP Min/Max and Compare/Exchange Instructions are added.
- URB atomics are not supported.

Programming Note	
Context:	Atomics
API Specification: The API specification says that atomic operations on Thread Group Shared Memory are atomic with respect to other atomic operations, as well as operations that only perform reads (loads). However, atomic operations on Thread Group Shared Memory are *not*atomic with respect to operations that perform only writes (stores) to memory. Mixing atomics and stores on the same Thread Group Shared Memory address without thread synchronization and memory fencing between them produces undefined results at the address involved. This restriction arises because some implementations of loads and stores do not honor the locking semantics for implementing atomics. It turns out this has no impact on loads, since they are guaranteed to retrieve a value either	

Programming Note	
Context:	Atomics
before or after an atomic (they will not retrieve partially updated values, given they are all defined at 32-bit quanta). However, store operations could find their way into the middle of an atomic operation and thus have their effect possibly lost.	

In L3 or SLM, the atomic operation leads to a read-modify-write operation on the destination location with the option of returning value back to requester. The table below is defined as a list of atomic operations needed:

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (Optional)
Atomic_AND	0000_0001	Single component 32-bit bitwise AND of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	"old_dst" AND "src0"	global/SLM	old_dst
Atomic_OR	0000_0010	Single component 32-bit bitwise OR of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	"old_dst" OR "src0"	global/SLM	old_dst
Atomic_XOR	0000_0011	Single component 32-bit bitwise XOR of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	"old_dst" XOR "src0"	global/SLM	old_dst
Atomic_MOVE	0000_0100	Replace the dst with src0.	"src0"	global/SLM	old_dst
Atomic_INC	0000_0101	Single component 32-bit integer increment of dst back into dst.	"old_dst + 1"	global/SLM	old_dst
Atomic_DEC	0000_0110	Single component 32-bit integer decrement of dst back into dst.	"old_dst - 1"	global/SLM	old_dst
Atomic_ADD	0000_0111	Single component 32-bit integer add of operand src0 into dst at 32-bit per component address dstAddress, performed atomically. Insensitive to sign.	"old_dst + src0"	global/SLM	old_dst
Atomic_SUB	0000_1000	Single component 32-bit integer subtraction of operand src0 from dst at 32-bit per component address dstAddress, performed atomically. Insensitive to sign.	"old_dst - src0"	global/SLM	old_dst
Atomic_RSUB	0000_1001	Single component 32-bit integer subtraction of operand dst from src0 into dst at 32-bit per component address dstAddress,	"src0 - old_dst"	global/SLM	old_dst

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (Optional)
		performed atomically. Insensitive to sign.			
Atomic_IMAX	0000_1010	Single component 32-bit signed MAX of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	IMAX(old_dst, src0)	global/SLM	old_dst
Atomic_IMIN	0000_1011	Single component 32-bit signed MIN of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	IMIN(old_dst, src0)	global/SLM	old_dst
Atomic_UMAX	0000_1100	Single component 32-bit unsigned MAX of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	UMAX(old_dst, src0)	global/SLM	old_dst
Atomic_UMIN	0000_1101	Single component 32-bit unsigned MIN of operand src0 into dst at 32-bit per component address dstAddress, performed atomically.	UMIN(old_dst, src0)	global/SLM	old_dst
Atomic_ CMP/WR	0000_1110	Single component 32-bit value compare of operand src0 with dst at 32-bit per component address dstAddress.	(src0 == old_dst)?	global/SLM	old_dst
		If the compared values are identical, the single-component 32-bit value in src1 is written to destination memory, else the destination is not changed.	src1:		
		The entire compare+write operation is performed atomically.	old_dst		
Atomic_ PREDEC	0000_1111	Single component 32-bit integer decrement of dst back into dst.	"old_dst - 1"	global/SLM	new_dst
Atomic_AND8B	0010_0001	Single component 64-bit bitwise AND of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	"old_dst8B" AND "src08B"	global	old_dst8B
Atomic_OR8B	0010_0010	Single component 64-bit bitwise OR of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	"old_dst8B" OR "src08B"	global	old_dst8B

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (Optional)
Atomic_XOR8B	0010_0011	Single component 64-bit bitwise XOR of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	"old_dst8B" XOR "src08B"	global	old_dst8B
Atomic_ MOVE8B	0010_0100	Replacement of the dst with src0.	"src08B"	global	old_dst8B
Atomic_INC8B	0010_0101	Single component 64-bit integer increment of dst back into dst	"old_dst8B + 1"	global	old_dst8B
Atomic_DEC8B	0010_0110	Single component 64-bit integer decrement of dst back into dst	"old_dst8B - 1"	global	old_dst8B
Atomic_ADD8B	0010_0111	Single component 64-bit integer add of operand src0 into dst at 64-bit per component address dstAddress, performed atomically. Insensitive to sign.	"old_dst8B + src08B"	global	old_dst8B
Atomic_SUB8B	0010_1000	Single component 64-bit integer subtraction of operand src0 from dst at 64-bit per component address dstAddress, performed atomically. Insensitive to sign.	"old_dst8B - src08B"	global	old_dst8B
Atomic_ RSUB8B	0010_1001	Single component 64-bit integer subtraction of operand dst from src0 into dst at 64-bit per component address dstAddress, performed atomically. Insensitive to sign.	"src08B - old_dst8B"	global	old_dst8B
Atomic_IMAX8B	0010_1010	Single component 64-bit signed MAX of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	IMAX (old_dst8B, src08B)	global	old_dst8B
Atomic_IMIN8B	0010_1011	Single component 64-bit signed MIN of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	IMIN (old_dst8B, src08B)	global	old_dst8B
Atomic_ UMAX8B	0010_1100	Single component 64-bit unsigned MAX of operand src0 into dst at 64-bit per component address dstAddress, performed atomically.	UMAX (old_dst8B, src08B)	global	old_dst8B
Atomic_	0010_1101	Single component 64-bit unsigned MIN of operand src0 into dst at 64-	UMIN (old_dst8B, src08B)	global	old_dst8B

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (Optional)
UMIN8B		bit per component address dstAddress, performed atomically.			
Atomic_ CMP/WR8B	0010_1110	Single component 64-bit value compare of operand src0 with dst at 64-bit per component address dstAddress.	(src08B == old_dst8B)?	global	old_dst8B
		If the compared values are identical, the single-component 64-bit value in src1 is written to destination memory, else the destination is not changed.	src18B:		
		The entire compare+write operation is performed atomically.	old_dst8B		
Atomic_ PREDEC8B	0010_1111	Single component 64-bit integer decrement of dst back into dst.	"old_dst8B - 1"	global	new_dst8B
Atomic_ CMP/WR16B	0100_1110	Single component 64-bit value compare of operand src0 with dst at 64-bit per component address dstAddress.	(src0_16B == old_dst16B)?	global	old_dst16B
		If the compared values are identical, the single-component 64-bit value in src1 is written to destination memory, else the destination is not changed.	src1_16B:		
		The entire compare+write operation is performed atomically.	old_dst_16B		
Atomic_MAX_Float32	1000_1010	Single component / single precision float MAX of operand src0 into dst dstAddress, performed atomically.	FMAX32(old_dst, src0)	global/SLM	old_dst
Atomic_MIN_Float32	1000_1011	Single component / single precision float MIN of operand src0 into dst dstAddress, performed atomically.	FMIN32(old_dst, src0)	global/SLM	old_dst
Atomic_ CMP/WR_Float32	1000_1110	Single component / single precision value compare of operand src0 with dst at address dstAddress. If the compared values are different, the single-component 32-bit value in src1 is written to destination memory, else the	(src0 == old_dst)?	global/SLM	old_dst

Atomic Operation	Opcode	Description	New Destination Value	Applicable	Return Value (Optional)
		destination is not changed. The entire compare+write operation is performed atomically.			

L3 Cache Error Protection

L3 cache error protection is covered via ECC (SECDED). All accesses are subject to ECC protection where single bit errors are fixed silently. Double bit errors are reported via a register structure and communicated by an interrupt to GFX driver. L3 cache HW is additionally capable of stalling execution upon a double bit error.