

# EE User's Manual

---

Copyright © 2002 Sony Computer Entertainment Inc.  
All Rights Reserved.  
SCE Confidential

© 2002 Sony Computer Entertainment Inc.

Publication date: April 2002

Sony Computer Entertainment Inc.  
1-1, Akasaka 7-chome, Minato-ku  
Tokyo 107-0052 Japan

Sony Computer Entertainment America  
919 East Hillsdale Blvd.  
Foster City, CA 94404, U.S.A.

Sony Computer Entertainment Europe  
30 Golden Square  
London W1F 9LD, U.K.

The *EE User's Manual* is supplied pursuant to and subject to the terms of the Sony Computer Entertainment PlayStation® license agreements.

The *EE User's Manual* is intended for distribution to and use by only Sony Computer Entertainment licensed Developers and Publishers in accordance with the PlayStation® license agreements.

Unauthorized reproduction, distribution, lending, rental or disclosure to any third party, in whole or in part, of this book is expressly prohibited by law and by the terms of the Sony Computer Entertainment PlayStation® license agreements.

Ownership of the physical property of the book is retained by and reserved by Sony Computer Entertainment. Alteration to or deletion, in whole or in part, of the book, its presentation, or its contents is prohibited.

The information in the *EE User's Manual* is subject to change without notice. The content of this book is Confidential Information of Sony Computer Entertainment.

 and PlayStation® are registered trademarks, and GRAPHICS SYNTHESIZER™ and EMOTION ENGINE™ are trademarks of Sony Computer Entertainment Inc. All other trademarks are property of their respective owners and/or their licensors.

## About This Manual

The "EE User's Manual" describes the configuration of the Emotion Engine and the functions of each block.

- Chapter 1 "EE Overview" shows the main specifications and block configuration of the Emotion Engine.
- Chapter 2 "Memory Map / Register Map" shows the physical memory map of the Emotion Engine and the register maps of the Emotion Engine, VPU and Graphics Synthesizer.
- Chapter 3 "INTC: Interrupt Controller" describes the function of the interrupt controller built into the Emotion Engine.
- Chapter 4 "TIMER" describes the functions of the timers built into the Emotion Engine.
- Chapter 5 "DMAC: DMA Controller" describes DMA transfer between main memory, scratchpad RAM, and each sub-unit.
- Chapter 6 "VPU: Vector Operation Processor" describes the VPUs, vector operation units that have the ability to execute four-parallel floating-point operations, and the VIF, an interface unit to the VPU. For details of the operational functions, refer to the "VU User's Manual".
- Chapter 7 "GIF: GS Interface" describes the GIF, a unit that interfaces from the Emotion Engine to the Graphics Synthesizer. See also the "GS User's Manual".
- Chapter 8 "IPU: Image Data Processor" describes the IPU, which can expand compressed animations and texture data.
- Chapter 9 "Appendix" shows the typical processing and data flow in the Emotion Engine.

For the EE Core, the core of the CPU, refer to the separate documents "EE Core User's Manual" and "EE Core Instruction Set Manual".

## Changes Since Release of 5th Edition

Since release of the 5th Edition of the EE User's Manual, the following changes have been made.

Note that each of these changes is indicated by a revision bar in the margin of the affected page.

### Ch. 1: EE Overview

- A correction has been made to section 1.2.8. IPU, on page 17.

### Ch. 4: Timer

- A correction has been made to section 4.1. Timer Overview, on page 34.
- Corrections have been made to the "CLKS" row, the "CMPE" row and the "OVFE" row in the Tn\_MODE register table on page 36.

### Ch. 5: DMAC: DMA Controller

- A correction has been made to section 5.1. DMA Channel, on page 42.
- Information has been added to section 5.3.2. Stall Control, on page 52.
- Figure 5-7 Stall Control, on page 52, has been revised. Information about stall control has been added and the diagram is improved.
- Section 5.7. Suspending and Restarting DMA Transfer, on page 62, has been expanded and clarified.
- Information has been added to the notes for the D\_CTRL register on page 64.
- Information has been added to the "ADDR" row in the D\_RBOR register table on page 68.
- A correction has been made to the "RMSK" row in the D\_RBSR register table on page 69.
- Information has been added to the description of the D\_ENABLEW register on page 71.
- Information has been added to the description of the D\_ENABLER register on page 72.
- Information has been added to the description of the Dn\_CHCR register on page 74.

**Ch. 6: VPU: Vector Operation Processor**

- The figures have been revised in “Activation of Micro Program”, section 6.3.8. Double Buffering, on pages 99, 100 and 101.
- Information regarding an xgkick instruction with the E bit set has been added to “Operation”, in the description of the FLUSH VIFcode on page 112.
- A correction has been made to the “CMD” row in the MPG VIFcode table on page 120.
- A correction has been made to the “CMD” row in the DIRECT VIFcode table on page 121.
- A correction has been made to the “CMD” row in the DIRECTHL VIFcode table on page 122.
- Information has been added to the description of the VIF0\_ITOP/ VIF1\_ITOP VIFcode on page 130.
- A correction has been made to the description of the VIF0\_ITOPS/ VIF1\_ITOPS VIFcode on page 131.

**Ch. 7: GIF: GS Interface**

- A correction has been made to “PATH1” in section 7.1.1. General Data Transfer Path on page 148.
- Information has been added to “A+D(0x0e)” in section 7.3.2. Packing Format on page 155.
- A correction has been made to the “PSE” row in the GIF\_STAT register table on page 164.
- A correction has been made to the description of the GIF\_P3CNT register on page 170.

**Ch. 8: IPU: Image Data Processor**

- A correction has been made to the Motion Code (TBL=10) table on page 204.
- A correction has been made to step 5) in section 8.6.1. CSC: Conversion from RAW8 to RGB32 on page 206.
- Corrections have been made to step 1) in section 8.6.3. CSC: Conversion from RGB16 to INDX4 on page 208.

# Glossary

Term	Definition
EE	Emotion Engine. CPU of the PlayStation 2.
EE Core	Generalized computation and control unit of EE. Core of the CPU.
COP0	EE Core system control coprocessor.
COP1	EE Core floating-point operation coprocessor. Also referred to as FPU.
COP2	Vector operation unit coupled as a coprocessor of EE Core. VPU0.
GS	Graphics Synthesizer. Graphics processor connected to EE.
GIF	EE Interface unit to GS.
IOP	Processor connected to EE for controlling input/output devices.
SBUS	Bus connecting EE to IOP.
VPU (VPU0/VPU1)	Vector operation unit. EE contains 2 VPUs: VPU0 and VPU1.
VU (VU0/VU1)	VPU core operation unit.
VIF (VIF0/VIF1)	VPU data decompression unit.
VIFcode	Instruction code for VIF.
SPR	Quick-access data memory built into EE Core (Scratchpad memory).
IPU	EE Image processor unit.
word	Unit of data length: 32 bits
qword	Unit of data length: 128 bits
Slice	Physical unit of DMA transfer: 8 qwords or less
Packet	Data to be handled as a logical unit for transfer processing.
Transfer list	A group of packets transferred in serial DMA transfer processing.
Tag	Additional data indicating data size and other attributes of packets.
DMAtag	Tag positioned first in DMA packet to indicate address/size of data and address of the following packet.
GS primitive	Data to indicate image elements such as point and triangle.
Context	A set of drawing information (e.g. texture, distant fog color, and dither matrix) applied to two or more primitives uniformly. Also referred to as the drawing environment.
GIFtag	Additional data to indicate attributes of GS primitives.
Display list	A group of GS primitives to indicate batches of images.

(This page is left blank intentionally)

# Contents

1. EE Overview.....	13
1.1. Main Specifications.....	14
1.2. EE Configuration.....	15
1.2.1. EE Core.....	16
1.2.2. FPU.....	16
1.2.3. INTC.....	16
1.2.4. TIMER.....	16
1.2.5. DMAC.....	16
1.2.6. VPU.....	16
1.2.7. GIF.....	16
1.2.8. IPU.....	17
1.2.9. SIF.....	17
2. Memory Map / Register Map.....	19
2.1. Memory Map.....	20
2.1.1. EE Memory Map.....	20
2.1.2. Bus Error.....	20
2.2. Register Map.....	21
2.2.1. EE Register.....	21
2.2.2. VU Memory.....	25
2.2.3. GS Privileged Registers.....	26
3. INTC: Interrupt Controller.....	27
3.1. Overview of Interrupt Processing.....	28
3.2. INTC Registers.....	29
I_STAT : Interrupt status register.....	30
I_MASK : Interrupt mask register.....	31
4. TIMER.....	33
4.1. Timer Overview.....	34
4.2. Timer-Related Registers.....	35
Tn_MODE : Register for setting modes and reading status.....	36
Tn_COUNT : Counter register.....	37
Tn_COMP : Comparison register.....	38
Tn_HOLD : Hold register.....	39
5. DMAC: DMA Controller.....	41
5.1. DMA Channel.....	42
5.2. Transfer Mode.....	43
5.2.1. Physical Transfer Mode and Logical Transfer Mode.....	43
5.2.2. Normal Mode.....	44
5.2.3. Source Chain Mode.....	45
5.2.4. Destination Chain Mode.....	48
5.2.5. Interleave Mode.....	50

5.3. Arbitration of Bus Right.....	51
5.3.1. Arbitration in Order of Priority.....	51
5.3.2. Stall Control.....	51
5.3.3. Priority Setting by Tag.....	54
5.4. Interrupt by DMA.....	55
5.5. MFIFO.....	57
5.6. DMAtag.....	58
Source Chain Tag.....	59
Destination Chain Tag.....	61
5.7. Suspending and Restarting DMA Transfer.....	62
5.8. Common Registers.....	63
D_CTRL : DMA control register.....	64
D_STAT : Interrupt status register.....	65
D_PCR : Priority control register.....	66
D_SQWC : Interleave size register.....	67
D_RBOR : Ring buffer address register.....	68
D_RBSR : Ring buffer size register.....	69
D_STADR : Stall address register.....	70
D_ENABLEW : DMA hold control register.....	71
D_ENABLER : DMA hold state register.....	72
5.9. Channel Registers.....	73
Dn_CHCR : Channel control register.....	74
Dn_MADR : Transfer address register.....	75
Dn_TADR : Tag address register.....	76
Dn_ASR0 / Dn_ASR1 : Tag address save register.....	77
Dn_SADR : SPR transfer address register.....	78
Dn_QWC : Transfer data size register.....	79
5.10. Restrictions.....	80
6. VPU: Vector Operation Processor.....	81
6.1. VPU Overview.....	82
6.1.1. VPU Structure.....	82
6.1.2. VPU0 and VPU1.....	82
6.1.3. VIF.....	83
6.2. VU Mem /MicroMem.....	84
6.2.1. VU Mem0 Memory Map.....	84
6.2.2. MicroMem0 Memory Map.....	85
6.2.3. VU Mem1 Memory Map.....	85
6.2.4. MicroMem1 Memory Map.....	85
6.3. Data Transfer by VIF.....	86
6.3.1. VIF Packet.....	86
6.3.2. VIFcode Structure.....	87
6.3.3. Limitation of Alignment.....	88
6.3.4. Data Write by UNPACK.....	89
6.3.5. Skipping Write and Filling Write.....	94
6.3.6. Write Data Mask.....	95
6.3.7. Addition Decompression Write.....	97

6.3.8. Double Buffering.....	99
6.4. VIFcode Reference.....	102
NOP : No operation.....	103
STCYCL : Sets write recycle.....	104
OFFSET : Sets the double buffer offset.....	105
BASE : Sets the base address of the double buffer.....	106
ITOP : Sets the data pointer.....	107
STMOD : Sets the addition decompression mode.....	108
MSKPATH3 : Sets the PATH3 mask.....	109
MARK : Sets the MARK value.....	110
FLUSHE : Waits for end of the microprogram.....	111
FLUSH : Waits for end of the microprogram.....	112
FLUSHA : Waits for end of the microprogram.....	113
MSCAL : Activates the microprogram.....	114
MSCNT : Activates the microprogram.....	115
MSCALF : Activates the microprogram.....	116
STMASK : Sets the data mask pattern.....	117
STROW : Sets the filling data.....	118
STCOL : Sets the filling data.....	119
MPG : Transfers the microprogram.....	120
DIRECT : Transfers data to the GIF(GS).....	121
DIRECTHL : Transfers data to the GIF(GS).....	122
UNPACK : Transfers data to the VU Mem.....	123
6.5. VPU-Related Register Reference.....	124
VIF0_R[0-3] / VIF1_R[0-3] : Row filling data.....	125
VIF0_C[0-3] / VIF1_C[0-3] : Column filling data.....	126
VIF0_CYCLE / VIF1_CYCLE : Write cycle.....	127
VIF0_MASK / VIF1_MASK : Write mask pattern.....	128
VIF0_MODE / VIF1_MODE : Addition processing mode.....	129
VIF0_ITOP / VIF1_ITOP : Data address.....	130
VIF0_ITOPS / VIF1_ITOPS : Data address.....	131
VIF1_BASE : Double buffer base address.....	132
VIF1_OFST : Double buffer offset.....	133
VIF1_TOP : Data address.....	134
VIF1_TOPS : Data decompression destination address.....	135
VIF0_MARK / VIF1_MARK : Mark value.....	136
VIF0_NUM : Amount of untransferred data.....	137
VIF1_NUM : Amount of untransferred data.....	138
VIF0_CODE / VIF1_CODE : Most recently processed VIFcode.....	139
VIF0_STAT : VIF status register.....	140
VIF0_FBRST : VIF reset register.....	141
VIF0_ERR : Error mask register.....	142
VIF1_STAT : VIF status register.....	143
VIF1_FBRST : VIF reset register.....	145
VIF1_ERR : VIF error mask register.....	146
7. GIF: GS Interface.....	147

7.1. Data Transfer Route .....	148
7.1.1. General Data Transfer Path .....	148
7.1.2. Priority and Timing.....	149
7.1.3. PATH3 Transfer Mode.....	149
7.1.4. PATH3 Operation Control.....	149
7.1.5. GS Privileged Register.....	149
7.2. Data Format.....	150
7.2.1. GS Packet.....	150
7.2.2. GIFtag.....	151
7.2.3. Register Descriptor.....	152
7.3. PACKED Mode.....	153
7.3.1. PACKED Mode Operation .....	153
7.3.2. Packing Format.....	153
7.3.3. Operation Example in PACKED Mode .....	156
7.4. REGLIST Mode.....	159
7.4.1. REGLIST Mode Operation .....	159
7.4.2. Operation Example in REGLIST Mode.....	159
7.5. IMAGE Mode .....	160
7.5.1. IMAGE Mode Operation.....	160
7.5.2. Operation Example in IMAGE Mode.....	160
7.6. GIF Control Register.....	161
GIF_CTRL Register : Control register.....	162
GIF_MODE Register : Mode setting register.....	163
GIF_STAT Register : Status register .....	164
GIF_TAG0 : GIFtag save register.....	165
GIF_TAG1 : GIFtag save register.....	166
GIF_TAG2 : GIFtag save register.....	167
GIF_TAG3 : GIFtag save register.....	168
GIF_CNT : Count register.....	169
GIF_P3CNT : PATH3 count register.....	170
GIF_P3TAG : PATH3 tag register.....	171
8. IPU: Image Data Processor.....	173
8.1. IPU Overview .....	174
8.1.1. Basic Functions .....	174
8.1.2. I/O and Execution of Command.....	175
8.1.3. Processing Example Using IPU.....	176
8.2. Data Format.....	177
8.2.1. BS128.....	177
8.2.2. RGB32.....	177
8.2.3. RGB16.....	178
8.2.4. RAW8.....	179
8.2.5. RAW16.....	180
8.2.6. INDX4.....	181
8.3. IPU Register Set.....	182
IPU_CMD : Decoded-code read register / IPU command register .....	183
IPU_TOP : Bit stream read register.....	184

IPU_CTRL : IPU status register / control register .....	185
IPU_BP : Bit position register .....	186
8.4. IPU Commands .....	187
BCLR : Input FIFO initialization command .....	188
IDEC : Intra decoding command .....	189
BDEC : Block decoding command .....	191
VDEC : Variable-length data decoding command .....	193
FDEC : Fixed-length data decoding command .....	194
SETIQ : IQ table set command .....	195
SETVQ : VQ table set command .....	196
CSC : Color space conversion command .....	197
PACK : Format conversion command .....	198
SETTH : Threshold value setting .....	199
8.5. Bit Stream Symbols .....	200
8.5.1. IDEC Symbols .....	200
8.5.2. BDEC Symbols .....	201
8.5.3. VDEC VLC Table .....	202
8.6. Post Processing .....	205
8.6.1. CSC: Conversion from RAW8 to RGB32 .....	205
8.6.2. Dithering: Conversion from RGB32 to RGB16 .....	207
8.6.3. VQ: Conversion from RGB16 to INDX4 .....	208
9. Appendix .....	209
9.1. Data Flow Model .....	210
9.1.1. Graphics Data Flow .....	210
9.1.2. Graphics Data Flow Example (1) .....	211
9.1.3. Graphics Data Flow Example (2) .....	212
9.1.4. Graphics Data Flow Example (3) .....	213
9.1.5. Image Data Flow Example (1) .....	214
9.1.6. Image Data Flow Example (2) .....	215
9.1.7. Image Data Flow Example (3) .....	216
9.1.8. Data Transfer to VPU1 .....	217

(This page is left blank intentionally)

## **1. EE Overview**

---

## 1.1. Main Specifications

Block	Name	Contents
CPU	CORE	64-bit, conforms to MIPS III(partly to MIPS IV) 2-way superscalar Internal bus 128 bits Internal registers 128 bits x 32
	CACHE	I-Cache 16 KB D-Cache 8 KB D-Scratch pad (SPR) 16 KB
	MMU	32-bit physical/logical address space conversion
	MM Extension	128-bit parallel multimedia instruction set
	Coprocessor	FPU
	VPU0	32-bit single-precision floating-point product sum arithmetic logical unit x 4 32-bit single-precision floating-point division calculator x 1 Data unpacking function Programmable LIW DSP Internal bus (data) 128 bits
Coordinate engine	VPU1	32-bit single-precision floating-point product sum arithmetic logical unit x 5 32-bit single-precision floating-point division calculator x 2 Data unpacking function Programmable LIW DSP Internal bus (data) 128 bits
Image engine	IPU	MPEG2 video layer decoding/bit stream decoding /IDCT/CSC (Color Space Conversion) /Dither VQ (Vector Quantization)
Built-in devices	DMAC	10ch (transfer between MEM and I/O, MEM and SPR) INT1 interrupt
	DRAMC	RDRAM controller
	INTC	INT0 (15 interrupts from devices)
	TIMER	16 bits x 4
	GIF	256-byte FIFO embedded
	SIF	32-bit (address/data multiplex), 128-byte FIFO embedded
Main bus		128 bits

## 1.2. EE Configuration

The block diagram of EE is as shown below.

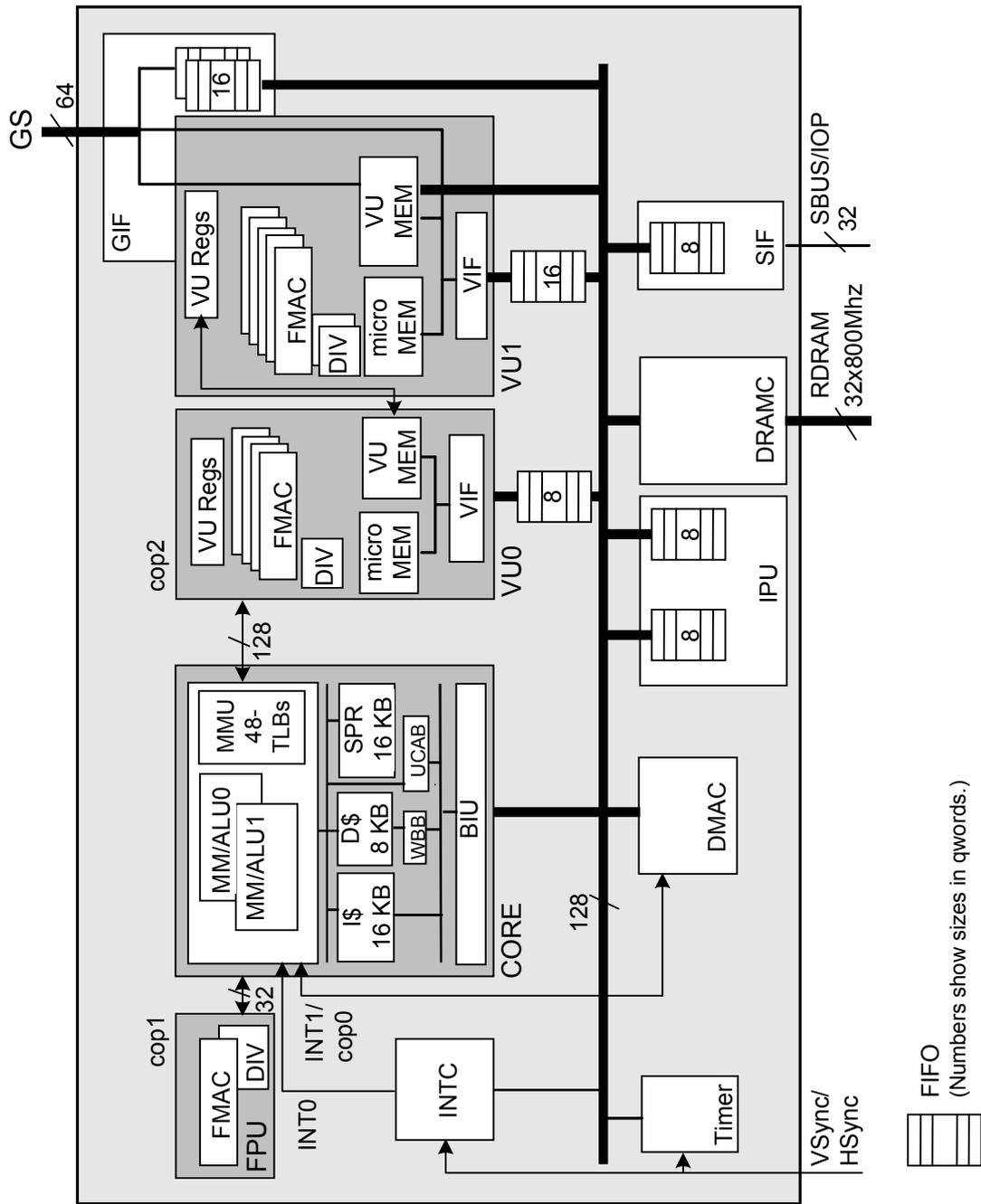


Figure 1-1 EE Block Diagram

### 1.2.1. EE Core

Category	Name	Contents
Cache	I-Cache	16 KB (2-way set associative)
	D-Cache	8 KB (2-way set associative, with line lock)
	D-Scratchpad (SPR)	16 KB
Internal bus	Data bus	128 bits (64 bits x 2)
	System bus	128 bits
Instruction set	64 bits, compatible with MIPS III (partly MIPS IV compatible) 3-operand multiplication/MAC instruction 128-bit multimedia instruction Interrupt enabled/disabled instruction	
MMU	48-entry TLB 32-bit physical/logical address space	

### 1.2.2. FPU

The FPU is a high-speed floating-point coprocessor that performs 32-bit multiplication / division calculations.

### 1.2.3. INTC

The INTC is an interrupt controller to arbitrate interrupts from peripheral devices, except for the DMAC.

### 1.2.4. TIMER

The EE has 4 independent timers that are synchronized to the screen display, based on the bus clock or the V-BLNK.

### 1.2.5. DMAC

The DMAC is an intelligent DMA controller that handles data transfer between main memory and peripheral processors or main memory and the scratch pad memory (SPR) while arbitrating the main bus at the same time. The main features of the DMAC include the Chain mode, which switches the transfer address according to the tag (DMA tag) attached to the data transferred: and the stall control function, which synchronizes two-channel transfer.

### 1.2.6. VPU

The VPU is a vector calculation unit that primarily performs coordinate calculation. It has independent data memory and program memory, and performs micro instruction programs.

The EE has two VPU's: VPU0 and VPU1. VPU0 is connected to the EE Core via a coprocessor connection: primarily executes non-patterned geometric processing, together with the EE Core. VPU1 is an independent processor of the EE Core and it primarily executes patterned geometric processing, and directly outputs a generated display list to GIF.

The VIF is an interface unit to the VPU that expands a compressed vector data string and writes it into VPU data memory.

### 1.2.7. GIF

The GIF is an interface unit to the GS. It converts data according to a tag (GIFtag) specification at the beginning of a display list packet and transfers drawing commands to the GS while mutually arbitrating multiple transfer.

**1.2.8. IPU**

The IPU is an image data processor to expand compressed animations and texture images. It performs MPEG2 bit stream decompression, I-PICTURE macro block decoding, color space conversion and vector quantization.

**1.2.9. SIF**

The SIF is an interface unit to the IOP, which has its own memory and bus to control sound chips, storage devices and other I/O devices.

(This page is left blank intentionally)

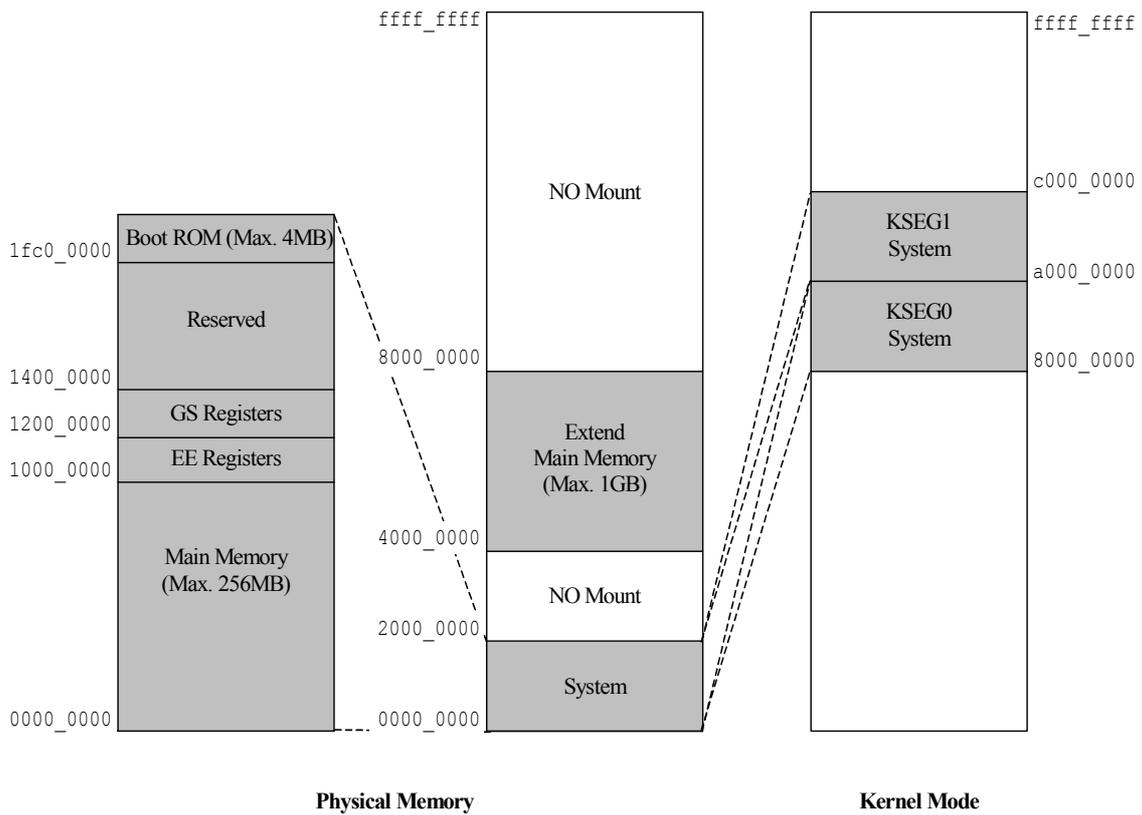
## **2. Memory Map / Register Map**

---

## 2.1. Memory Map

### 2.1.1. EE Memory Map

The physical address space of the EE is 2 GB. KSEG0 and KSEG1 are mapped as shown below.



### 2.1.2. Bus Error

The memory controller asserts a BUSERR\* signal under the following conditions:

- When accessing an EE register area where a register is not allocated (described later in this document)
- When accessing 0x2000\_0000 to 0x3fff\_fff0 and 0x8000\_0000 to 0xffff\_fff0
- When an RDY signal is not returned for 2048 cycles by the bus clock (147.456MHz) from the start of an SBUS access

## 2.2. Register Map

Internal registers are allocated as follows. The following are descriptions of the "Reserved" areas in the table.

"Reserved" Areas	Description
(Reserved)	Write access: Disregarded. Read access: An indeterminate value is returned.
(Reserved) Image of xx	Mapped xx is duplicated.
(Reserved) Bus error	No harm except that an access results in a bus error.

### 2.2.1. EE Register

The registers listed in the following table are 32-bit long and only word-accessible, unless specified otherwise. FIFO's are 128-bit long and only qword-accessible, unless specified otherwise.

Module	Address	Name	Contents
TIMER	0x1000_0000	T0_COUNT	Timer 0 counter value
	0x1000_0010	T0_MODE	Timer 0 mode/status
	0x1000_0020	T0_COMP	Timer 0 compare value
	0x1000_0030	T0_HOLD	Timer 0 hold value
	0x1000_0040 - 0x1000_07f0		(Reserved)
	0x1000_0800	T1_COUNT	Timer 1 counter value
	0x1000_0810	T1_MODE	Timer 1 mode/status
	0x1000_0820	T1_COMP	Timer 1 compare value
	0x1000_0830	T1_HOLD	Timer 1 hold value
	0x1000_0840 - 0x1000_0ff0		(Reserved)
	0x1000_1000	T2_COUNT	Timer 2 counter value
	0x1000_1010	T2_MODE	Timer 2 mode/status
	0x1000_1020	T2_COMP	Timer 2 compare value
	0x1000_1030 - 0x1000_17f0		(Reserved)
	0x1000_1800	T3_COUNT	Timer 3 counter value
	0x1000_1810	T3_MODE	Timer 3 mode/status
	0x1000_1820	T3_COMP	Timer 3 compare value
	0x1000_1830 - 0x1000_1ff0		(Reserved)
IPU	0x1000_2000	IPU_CMD	IPU command
	0x1000_2010	IPU_CTRL	IPU control
	0x1000_2020	IPU_BP	IPU input FIFO control
	0x1000_2030	IPU_TOP	First data of bit stream
	0x1000_2040 - 0x1000_2ff0		(Reserved)
GIF	0x1000_3000	GIF_CTRL	GIF control
	0x1000_3010	GIF_MODE	GIF mode setting
	0x1000_3020	GIF_STAT	GIF status
	0x1000_3030		(Reserved)
	0x1000_3040	GIF_TAG0	GIFtag (bits 31-0) immediately before
	0x1000_3050	GIF_TAG1	GIFtag (bits 63-32) immediately before
0x1000_3060	GIF_TAG2	GIFtag (bits 95-64) immediately before	

Module	Address	Name	Contents
	0x1000_3070	GIF_TAG3	GIFtag (bits 127-96) immediately before
	0x1000_3080	GIF_CNT	Transfer status counter
	0x1000_3090	GIF_P3CNT	PATH3 transfer status counter
	0x1000_30a0	GIF_P3TAG	PATH3 tag value
	0x1000_30b0 - 0x1000_37f0		(Reserved)
VIF0	0x1000_3800	VIF0_STAT	Status
	0x1000_3810	VIF0_FBRST	Operation control
	0x1000_3820	VIF0_ERR	Error status
	0x1000_3830	VIF0_MARK	Mark value
	0x1000_3840	VIF0_CYCLE	Data write cycle
	0x1000_3850	VIF0_MODE	ADD mode
	0x1000_3860	VIF0_NUM	Amount of non-transferred data
	0x1000_3870	VIF0_MASK	Write mask pattern
	0x1000_3880	VIF0_CODE	Last processed VIFcode
	0x1000_3890	VIF0_ITOPS	Next ITOP value
	0x1000_38a0		(Reserved)
	0x1000_38b0		(Reserved)
	0x1000_38c0		(Reserved)
	0x1000_38d0	VIF0_ITOP	ITOP value
	0x1000_38e0		(Reserved)
	0x1000_38f0		(Reserved)
	0x1000_3900	VIF0_R0	Filling data R0 (Row register)
	0x1000_3910	VIF0_R1	Filling data R1 (Row register)
	0x1000_3920	VIF0_R2	Filling data R2 (Row register)
	0x1000_3930	VIF0_R3	Filling data R3 (Row register)
	0x1000_3940	VIF0_C0	Filling data C0 (Col register)
	0x1000_3950	VIF0_C1	Filling data C1 (Col register)
	0x1000_3960	VIF0_C2	Filling data C2 (Col register)
	0x1000_3970	VIF0_C3	Filling data C3 (Col register)
	0x1000_3980 - 0x1000_3bf0		(Reserved)
VIF1	0x1000_3c00	VIF1_STAT	Status
	0x1000_3c10	VIF1_FBRST	Operation control
	0x1000_3c20	VIF1_ERR	Error status
	0x1000_3c30	VIF1_MARK	Mark value
	0x1000_3c40	VIF1_CYCLE	Data write cycle
	0x1000_3c50	VIF1_MODE	ADD mode
	0x1000_3c60	VIF1_NUM	Amount of non-transferred data
	0x1000_3c70	VIF1_MASK	Write mask pattern
	0x1000_3c80	VIF1_CODE	Last processed VIFcode
	0x1000_3c90	VIF1_ITOPS	Next ITOP value
	0x1000_3ca0	VIF1_BASE	Base address of double buffer
	0x1000_3cb0	VIF1_OFST	Offset of double buffer
	0x1000_3cc0	VIF1_TOPS	Next TOP value/data write address
	0x1000_3cd0	VIF1_ITOP	ITOP value
	0x1000_3ce0	VIF1_TOP	TOP value
	0x1000_3cf0		(Reserved)
	0x1000_3d00	VIF1_R0	Filling data R0 (Row register)
	0x1000_3d10	VIF1_R1	Filling data R1 (Row register)
	0x1000_3d20	VIF1_R2	Filling data R2 (Row register)
	0x1000_3d30	VIF1_R3	Filling data R3 (Row register)
	0x1000_3d40	VIF1_C0	Filling data C0 (Col register)

Module	Address	Name	Contents
	0x1000_3d50	VIF1_C1	Filling data C1 (Col register)
	0x1000_3d60	VIF1_C2	Filling data C2 (Col register)
	0x1000_3d70	VIF1_C3	Filling data C3 (Col register)
	0x1000_3d80 - 0x1000_3ff0		(Reserved)
FIFO	0x1000_4000	VIF0_FIFO	VIF0 FIFO (write)
	0x1000_4010 - 0x1000_4ff0		(Reserved) Image of 0x1000_4000
	0x1000_5000	VIF1_FIFO	VIF1 FIFO (read/write)
	0x1000_5010 - 0x1000_5ff0		(Reserved) Image of 0x1000_5000
	0x1000_6000	GIF_FIFO	GIF FIFO (write)
	0x1000_6010 - 0x1000_6ff0		(Reserved) Image of 0x1000_6000
	0x1000_7000	IPU_out_FIFO	IPU FIFO (read)
	0x1000_7010	IPU_in_FIFO	IPU FIFO (write)
	0x1000_7020 - 0x1000_7ff0		(Reserved) Image of 0x1000_7000 - 0x1000_7010
DMAC	0x1000_8000	D0_CHCR	Ch0 channel control
	0x1000_8010	D0_MADR	Ch0 memory address
	0x1000_8020	D0_QWC	Ch0 quad word count
	0x1000_8030	D0_TADR	Ch0 tag address
	0x1000_8040	D0_ASR0	Ch0 address stack 0
	0x1000_8050	D0_ASR1	Ch0 address stack 1
	0x1000_8060 - 0x1000_8ff0		(Reserved)
DMAC	0x1000_9000	D1_CHCR	Ch1 channel control
	0x1000_9010	D1_MADR	Ch1 memory address
	0x1000_9020	D1_QWC	Ch1 quad word count
	0x1000_9030	D1_TADR	Ch1 tag address
	0x1000_9040	D1_ASR0	Ch1 address stack 0
	0x1000_9050	D1_ASR1	Ch1 address stack 1
	0x1000_9060 - 0x1000_9ff0		(Reserved)
DMAC	0x1000_a000	D2_CHCR	Ch2 channel control
	0x1000_a010	D2_MADR	Ch2 memory address
	0x1000_a020	D2_QWC	Ch2 quad word count
	0x1000_a030	D2_TADR	Ch2 tag address
	0x1000_a040	D2_ASR0	Ch2 address stack 0
	0x1000_a050	D2_ASR1	Ch2 address stack 1
	0x1000_a060 - 0x1000_aff0		(Reserved)
DMAC	0x1000_b000	D3_CHCR	Ch3 channel control
	0x1000_b010	D3_MADR	Ch3 memory address
	0x1000_b020	D3_QWC	Ch3 quad word count
	0x1000_b030 - 0x1000_b3f0		(Reserved)
DMAC	0x1000_b400	D4_CHCR	Ch4 channel control
	0x1000_b410	D4_MADR	Ch4 memory address
	0x1000_b420	D4_QWC	Ch4 quad word count
	0x1000_b430	D4_TADR	Ch4 tag address
	0x1000_b440 - 0x1000_bff0		(Reserved)

Module	Address	Name	Contents
DMAC	0x1000_c000	D5_CHCR	Ch5 channel control
	0x1000_c010	D5_MADR	Ch5 memory address
	0x1000_c020	D5_QWC	Ch5 quad word count
	0x1000_c030 - 0x1000_c3f0		(Reserved)
DMAC	0x1000_c400	D6_CHCR	Ch6 channel control
	0x1000_c410	D6_MADR	Ch6 memory address
	0x1000_c420	D6_QWC	Ch6 quad word count
	0x1000_c430	D6_TADR	Ch6 tag address
	0x1000_c440 - 0x1000_c7f0		(Reserved)
DMAC	0x1000_c800	D7_CHCR	Ch7 channel control
	0x1000_c810	D7_MADR	Ch7 memory address
	0x1000_c820	D7_QWC	Ch7 quad word count
	0x1000_c830 - 0x1000_cff0		(Reserved)
DMAC	0x1000_d000	D8_CHCR	Ch8 channel control
	0x1000_d010	D8_MADR	Ch8 memory address
	0x1000_d020	D8_QWC	Ch8 quad word count
	0x1000_d030 - 0x1000_d070		(Reserved)
	0x1000_d080	D8_SADR	Ch8 SPR address
	0x1000_d090 - 0x1000_d3f0		(Reserved)
DMAC	0x1000_d400	D9_CHCR	Ch9 channel control
	0x1000_d410	D9_MADR	Ch9 memory address
	0x1000_d420	D9_QWC	Ch9 quad word count
	0x1000_d430	D9_TADR	Ch9 tag address
	0x1000_d440 - 0x1000_d470		(Reserved)
	0x1000_d480	D9_SADR	Ch9 SPR address
	0x1000_d490 - 0x1000_dff0		(Reserved)
DMAC	0x1000_e000	D_CTRL	DMAC control
	0x1000_e010	D_STAT	DMAC status
	0x1000_e020	D_PCR	DMAC priority control
	0x1000_e030	D_SQWC	DMAC skip quad word
	0x1000_e040	D_RBSR	DMAC ring buffer size
	0x1000_e050	D_RBOR	DMAC ring buffer offset
	0x1000_e060	D_STADR	DMA stall address
	0x1000_e070 - 0x1000_eff0		(Reserved)
INTC	0x1000_f000	I_STAT	Interrupt status
	0x1000_f010	I_MASK	Interrupt mask
	0x1000_f020 - 0x1000_f000		(Reserved)
SIF	0x1000_f230	SB_SMFLG	SBUS Sub -> Main communication flag
DMAC	0x1000_f520	D_ENABLER	Acquisition of DMA suspend status
	0x1000_f530 - 0x1000_f580		(Reserved)
	0x1000_f590	D_ENABLEW	DMA suspend control
	0x1000_f5a0 - 0x1000_f5f0		(Reserved)

### 2.2.2. VU Memory

Module	Address	Size	Contents
VU0	0x1100_0000 - 0x1100_0ff0	4KB	Micro Mem0
	0x1100_1000 - 0x1100_3ff0	12KB	(Reserved) Image of 0x1100_0000 - 0x1000_0ff0
	0x1100_4000 - 0x1100_4ff0	4KB	VU Mem0
	0x1100_5000 - 0x1100_7ff0	12KB	(Reserved) Image of 0x1100_4000 - 0x1000_4ff0
VU1	0x1100_8000 - 0x1100_bff0	16KB	Micro Mem1
	0x1100_c000 - 0x1100_ff0	16KB	VU Mem1
	0x1101_0000 - 0x11ff_ff0		(Reserved) Bus error

### 2.2.3. GS Privileged Registers

GS privileged registers must be accessed using LD/SD instructions.

When the EE Core accesses an address between 0x1200\_0000 and 0x13ff\_ffff, GIF recognizes it as an access to a GS privileged register and makes an access to the corresponding GS privileged register. In this case, addresses [12] and [9:4] of the CPU bus are output to GA[6:0].

Module	Address	Name	Contents	
GS Special	0x1200_0000	PMODE	Various PCRTC modes	
	0x1200_0010	SMODE1	Related to Sync	
	0x1200_0020	SMODE2	Related to Sync	
	0x1200_0030	SRFSH	DRAM refresh	
	0x1200_0040	SYNCH1	Related to Sync	
	0x1200_0050	SYNCH2	Related to Sync	
	0x1200_0060	SYNCV	Related to Sync/start	
	0x1200_0070	DISPFB1	Related to display buffer of Rectangular Area 1	
	0x1200_0080	DISPLAY1	Rectangular Area 1 display position etc.	
	0x1200_0090	DISPFB2	Related to display buffer of Rectangular Area 2	
	0x1200_00a0	DISPLAY2	Rectangular Area 2 display position etc.	
	0x1200_00b0	EXTBUF	Rectangular area write buffer	
	0x1200_00c0	EXTDATA	Rectangular area write data	
	0x1200_00d0	EXTWRITE	Rectangular area write start	
	0x1200_00e0	BGCOLOR	Background color	
	0x1200_00f0 - 0x1200_03f0			(Reserved) GS-dependent
	0x1200_0400 - 0x1200_07f0			(Reserved) Image of 0x1200_0000 - 0x1200_03f0
	0x1200_0800 - 0x1200_0fff			(Reserved) Image of 0x1200_0000 - 0x1200_07f0
	0x1200_1000	CSR	Various GS status	
	0x1200_1010	IMR	Interrupt mask	
	0x1200_1020 - 0x1200_1030			(Reserved) GS-dependent
	0x1200_1040	BUSDIR	Host interface switching	
	0x1200_1050 - 0x1200_1070			(Reserved) GS-dependent
	0x1200_1080	SIGLBLID	SIGNALID/LABELID	
	0x1200_1090 - 0x1200_10e0			(Reserved) GS-dependent
	0x1200_1100 - 0x1200_13f0			(Reserved) GS-dependent
	0x1200_1400 - 0x1200_17f0			(Reserved) Image of 0x1200_1000 - 0x1200_13f0
	0x1200_1800 - 0x1200_1fff			(Reserved) Image of 0x1200_1000 - 0x1200_17f0
	0x1200_2000 - 0x1200_ffff			(Reserved)
	0x1201_0000 - 0x13ff_ffff			(Reserved) Bus error

## 3. INTC: Interrupt Controller

---

The INTC (Interrupt Controller) arbitrates interrupt requests from multiple processors and sends an INT0 interrupt to the EE Core.

The INTC controls 15 interrupts.

### 3.1. Overview of Interrupt Processing

An interrupt request is taken into the I\_STAT register at the edge of an interrupt request signal, and the bit corresponding to the interrupt request source is set. Then, the AND between the I\_STAT register and the I\_MASK register is obtained per bit; if any bit is 1, an INT0 interrupt is asserted to the CPU.

The INTC controls the following interrupt request sources.

ID	Name	Cause		Edge
0	INT_GS	GS	External factor	Falling
1	INT_SBUS	SBUS	External factor	Falling
2	INT_VB_ON	V-blank start	External factor	Rising
3	INT_VB_OFF	V-blank end	External factor	Falling
4	INT_VIF0	VIF0	Internal factor	
5	INT_VIF1	VIF1	Internal factor	
6	INT_VU0	VU0	Internal factor	
7	INT_VU1	VU1	Internal factor	
8	INT_IPU	IPU	Internal factor	
9	INT_TIMER0	Timer0	Internal factor	
10	INT_TIMER1	Timer1	Internal factor	
11	INT_TIMER2	Timer2	Internal factor	
12	INT_TIMER3	Timer3	Internal factor	
13	INT_SFIFO	SFIFO	Internal factor	
14	INT_VU0WD	VU0 WatchDog	Internal factor	

The status registers and cause registers for peripherals can check the details of interrupt causes.

Besides the INTC, an interrupt request from the DMAC is connected to the EE Core as an independent signal (INT1\*).

Details of interrupts from each peripheral are as follows.

ID	Name	Interrupt Cause
0	INT_GS	Detection of an interrupt from GS
1	INT_SBUS	Detection of an interrupt from a peripheral on SBUS
2	INT_VB_ON	Start of V-Blank
3	INT_VB_OFF	End of V-Blank
4	INT_VIF0	VIF0's detection of VIFcode with an interrupt bit or an exception VIF0 stalls with occurrence of an interrupt.
5	INT_VIF1	VIF1's detection of VIFcode with an interrupt bit or an exception VIF1 stalls with occurrence of an interrupt.
6	INT_VU0	VU0's execution of a microinstruction with an interrupt bit VU0 stalls with occurrence of an interrupt.
7	INT_VU1	VU1's execution of a microinstruction with an interrupt bit VU1 stalls with occurrence of an interrupt.
8	INT_IPU	IPU's detection of the end of data or an exception IPU stalls with occurrence of an interrupt.
9-12	INT_TIMER [0-3]	Conditions met in timer settings
13	INT_SFIFO	Error detection during SFIFO transfer
14	INT_VU0WD	VU0 in RUN status for a long time continuously ForceBreak is sent to VU0.

## 3.2. INTC Registers

The INTC has the following registers.

Symbol	r/w	Width	Contents
I_STAT	r/w	32 bits	Interrupt status
I_MASK	r/w	32 bits	Interrupt mask set/release





(This page is left blank intentionally)

## 4. TIMER

---

The EE has 4 timers used to generate interrupts as time-out timers or interval timers.

## 4.1. Timer Overview

The EE has 4 independent timers numbered 0, 1, 2 and 3.

Each timer has a 16-bit counter. The bus clock (BUSCLK, 1/16 of BUSCLK and 1/256 of BUSCLK) or the external clock (H-BLNK) performs counting. Since the EE uses the external H-BLNK/V-BLNK as a gate signal, counting operations can be synchronized to the screen image.

Timer interrupts occur when the counter value reaches a certain value specified as the reference value, or when it overflows. The status register flag indicates which of the above has caused the interrupt.

Timer 0 and timer 1 have a hold register for recording the counter value when an SBUS interrupt occurs.

## 4.2. Timer-Related Registers

Each timer has the registers described below.

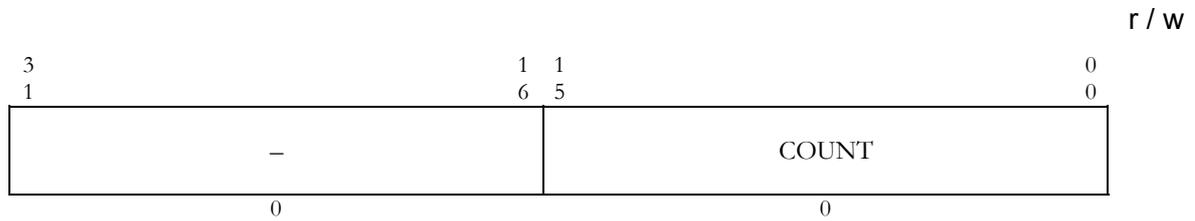
Symbol	r/w	Width	Contents
Tn_MODE (n=0,1,2,3)	r/w	32 bits	Mode setting and status reading
Tn_COUNT (n=0,1,2,3)	r/w	32 bits	Counter value (the upper 16 bits are fixed to 0)
Tn_COMP (n=0,1,2,3)	r/w	32 bits	Compare value (the upper 16 bits are fixed to 0)
Tn_HOLD (n=0,1)	r/w	32 bits	Hold value (the upper 16 bits are fixed to 0)

**Tn\_MODE : Register for setting modes and reading status**

r / w

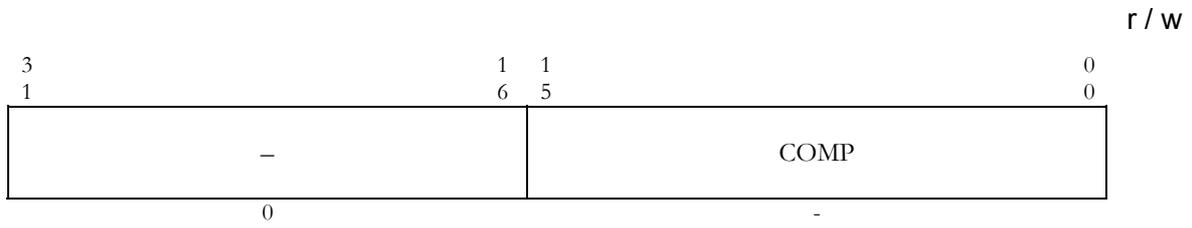
3	1 1 0 0 0 0 0 0 0 0 0 0																																													
1	1 0 9 8 7 6 5 4 3 2 1 0																																													
-	<table border="1" style="border-collapse: collapse; width: 100%;"> <tr> <td style="width: 25px;">O</td><td style="width: 25px;">E</td><td style="width: 25px;">O</td><td style="width: 25px;">C</td><td style="width: 25px;">Z</td><td style="width: 25px;">G</td><td style="width: 25px;">G</td><td style="width: 25px;">G</td><td style="width: 25px;">C</td> </tr> <tr> <td>V</td><td>Q</td><td>V</td><td>M</td><td>R</td><td>A</td><td>A</td><td>A</td><td>L</td> </tr> <tr> <td>F</td><td>U</td><td>F</td><td>P</td><td>E</td><td>T</td><td>T</td><td>T</td><td>K</td> </tr> <tr> <td>F</td><td>F</td><td>E</td><td>E</td><td>T</td><td>M</td><td>S</td><td>E</td><td>S</td> </tr> <tr> <td style="text-align: center;">0</td><td style="text-align: center;">0</td> </tr> </table>	O	E	O	C	Z	G	G	G	C	V	Q	V	M	R	A	A	A	L	F	U	F	P	E	T	T	T	K	F	F	E	E	T	M	S	E	S	0	0	0	0	0	0	0	0	0
O	E	O	C	Z	G	G	G	C																																						
V	Q	V	M	R	A	A	A	L																																						
F	U	F	P	E	T	T	T	K																																						
F	F	E	E	T	M	S	E	S																																						
0	0	0	0	0	0	0	0	0																																						

Name	Pos.	Contents
CLKS	1:0	Clock Selection 00 BUSCLK (147.456MHz) 01 1/16 of the BUSCLK 10 1/256 of the BUSCLK 11 External clock (H-BLNK)
GATE	2	Gate Function Enable 0 Gate function is not used. 1 Gate function is used.
GATS	3	Gate Selection 0 H-BLNK (Disabled when CLKS equals to 11.) 1 V-BLNK
GATM	5:4	Gate Mode 00 Counts while the gate signal is low. 01 Resets and starts counting at the gate signal's rising edge. 10 Resets and starts counting at the gate signal's falling edge. 11 Resets and starts counting at both edges of the gate signal.
ZRET	6	Zero Return 0 The counter keeps counting, ignoring the reference value. 1 The counter is cleared to 0 when the counter value is equal to the reference value.
CUE	7	Count Up Enable 0 Stops counting. 1 Starts/restarts counting.
CMPE	8	Compare-Interrupt Enable 0 A compare-interrupt is not generated. 1 An interrupt is generated when the counter value is equal to the reference value.
OVFE	9	Overflow-Interrupt Enable 0 An overflow interrupt is not generated. 1 An interrupt is generated when an overflow occurs.
EQUF	10	Equal Flag The value is set to 1 when a compare-interrupt occurs. Writing 1 clears the equal flag.
OVFF	11	Overflow Flag The value is set to 1 when an overflow-interrupt occurs. Writing 1 clears the equal flag.

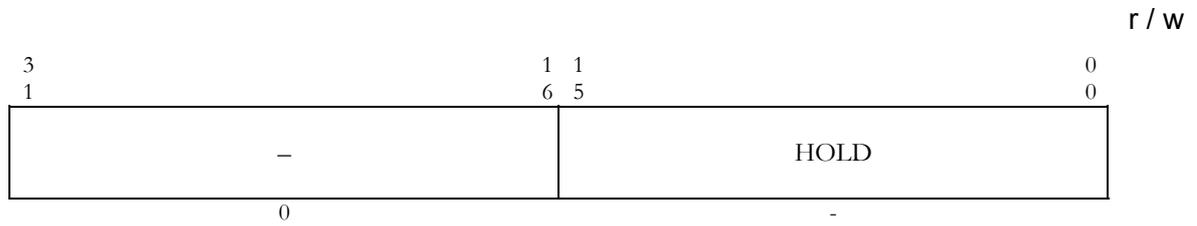
**Tn\_COUNT : Counter register**

Name	Pos.	Contents
COUNT	15:0	Counter Value The counter value is incremented according to the conditions of the clock and the gate signal specified in the Tn_MODE.

### Tn\_COMP : Comparison register



Name	Pos.	Contents
COMP	15:0	Compare Value Reference value to be compared with Tn_COUNT.

**Tn\_HOLD : Hold register**

Name	Pos.	Contents
HOLD	15:0	Hold Value The value of Tn_COUNT is copied when an SBUS interrupt occurs.

(This page is left blank intentionally)

## 5. DMAC: DMA Controller

---

The DMAC intelligently transfers data between main memory and peripheral processors and between main memory and scratchpad RAM (SPR) while performing arbitration of the main bus.

Data is transferred in qword (128-bit) units, and should be aligned on this boundary in memory. Transfer addresses are specified in terms of physical address and are not converted by the TLB. Moreover, in DMA transfers, bus snooping is not performed.

Data is transferred to or from a peripheral processor in 8-qword slices: whenever the transfer of one slice is completed, the channel used for the transfer temporarily releases the bus right. Therefore, two or more data items can be transferred concurrently, in addition to the CPU being able to access main memory during transfer processing.

In some of the channels, Chain mode is available. This mode performs processing such as transfer address switching according to the tag in the transfer data. This allows data to be exchanged between two or more processors through the mediation of the main memory, not the CPU.

A stall control function is available to synchronize data transfer to and from the main memory. This is reinforced for the GIF channel, and the Memory FIFO function, which uses the ring buffer in the main memory is available. Also, priority control is enabled which gives priority to a specific packet over other channels in transferring data.

## 5.1. DMA Channel

The processors to which the DMAC allocates data and the corresponding channels are as follows:

ID	Channel	Dir.	Pr.	P.M.	D.S.	M.F.	S.C.	D.C.	Int.	A.S.
0	VIF0	to	A	Slice	-	-	Yes	-	-	2
1	VIF1	both	C	Slice	Drain	Drain	Yes	-	-	2
2	GIF	to	C	Slice	Drain	Drain	Yes	-	-	2
3	fromIPU	from	C	Slice	Source	-	-	-	-	-
4	toIPU	to	C	Slice	-	-	Yes	-	-	-
5	SIF0	from	C	Slice	Source	-	-	Yes	-	-
6	SIF1	to	C	Slice	Drain	-	Yes	-	-	-
7	SIF2	both	B	Slice	-	-	-	-	-	-
8	fromSPR	from	C	Burst	Source	Source	-	Yes	Yes	-
9	toSPR	to	C	Burst	-	-	Yes	-	Yes	-

Dir.	Direction	both = both directions, from = from peripheral, to = to peripheral
Pr.	Priority Group	A>B>C
P.M.	Physical Mode	
D.S.	DMA Stall	
M.F.	MFIFO	
S.C.	Source Chain	
D.C.	Destination Chain	
Int.	Interleave	
A.S.	Address Stack	No. of addresses stacked

VIF1 channel (ID=1) can switch directions of transfer by making settings for the GS. For details, refer to "4.2.2. Transmission from Local Buffer to Host" in the "GS User's Manual".

The SBUS has three DMA channels (ID=5,6,7) and performs DMA transfer in cooperation with the corresponding SBUS DMA (SDMA).

Channels whose transfer destination is memory (excluding fromSPR) can select main memory or scratchpad memory as a transfer destination. Channels whose transfer source is memory (excluding toSPR) can select main memory or scratchpad memory as a transfer source.

The fromSPR/toSPR channels can transfer data between scratchpad memory and VU memory in Burst Mode by specifying VU Mem0 and VU Mem1 addresses mapped to main memory as a transfer destination/source. In this case, however, logical transfer mode is limited to Normal Mode.

## 5.2. Transfer Mode

### 5.2.1. Physical Transfer Mode and Logical Transfer Mode

Two DMA channel transfer modes are provided: physical transfer mode, which is fixed for each channel, and logical transfer mode, which is selectable on each channel.

Physical transfer mode is categorized into two types: Burst Mode, in which data is transferred in a group, and Slice Mode, in which data is transferred in units of 8-qword slices while arbitrating with other DMA channels. Transfer operations in Burst Mode and Slice Mode vary slightly between two Cycle Stealing states (on and off) regarding arbitration of the bus right with the EE Core.

Physical Mode	Channel	Operation
Burst	toSPR fromSPR	Cycle Stealing off: Transfers all the data at a time with the bus right occupied. Cycle Stealing on: Suspends transfer on each 128-byte boundary and releases the bus to the EE Core when there is a request from the EE Core.
Slice	others	Cycle Stealing off: Transfers data by dividing it into 8-qword slices. Suspends transfer on each slice boundary and performs arbitration. Cycle Stealing on: Suspends transfer and releases the bus to the EE Core in addition to the arbitration on a slice boundary.

Cycle Stealing is set on/off by the RELE bit of the D\_CTRL register.

In Burst Mode with Cycle Stealing on, the transfer is suspended temporarily on a 128-byte boundary in main memory during DMA transfer to allow the EE Core to access main memory. However, when the logical transfer mode is set to Interleave Mode, the transfer is suspended in units of the number of qwords specified in the TW field of the D\_SQWC register and the bus is released.

The DMAC suspends DMA transfer until the number of bus cycles specified in the RCYC field of the D\_CTRL register elapses after the bus right is returned to the EE Core. The next time the DMAC acquires the bus right, the suspended DMA transfer is restarted.

In the logical transfer mode, selection from three modes (Normal, Chain, and Interleave) is made for each channel. Chain Mode is divided between transfer from main/scratchpad memory to a peripheral (Source Chain Mode) and transfer from a peripheral to main/scratchpad memory (Destination Chain Mode). (Scratchpad memory is considered to be a peripheral on the toSPR/fromSPR channel.)

Logical Mode	Operation
Normal	Reads/writes data continuously between a specified address in main/scratchpad memory and a peripheral (general DMA transfer).
Source Chain	Transfers data while switching the transfer address and the transfer mode according to the tag in the transfer packet when transferring data from main/scratchpad memory to a peripheral.
Destination Chain	Transfers data while switching the destination address according to the tag in the transfer packet when transferring data from a peripheral to main/scratchpad memory.
Interleave	Transfers a rectangular area of two-dimensional data between main memory and scratchpad memory.

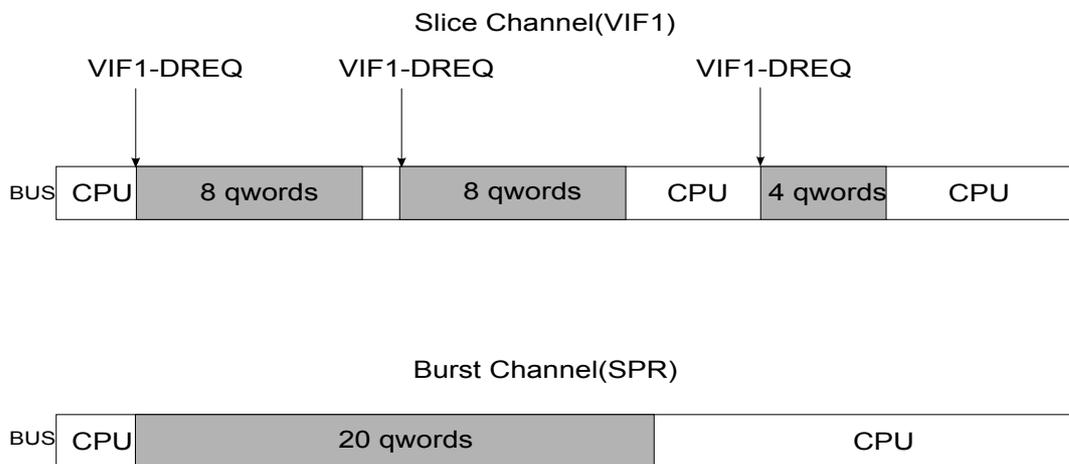
### 5.2.2. Normal Mode

Normal Mode handles general DMA transfers. The transfer address is specified by the Dn\_MADR register, and continuous data of the transfer size is specified by the Dn\_QWC register. The transfer is started by setting the STR bit of the Dn\_CHCR register to 1. The STR bit is cleared to 0 when the transfer is ended.

On the Burst Channel, data of the size specified by the Dn\_QWC register is transferred at one time continuously.

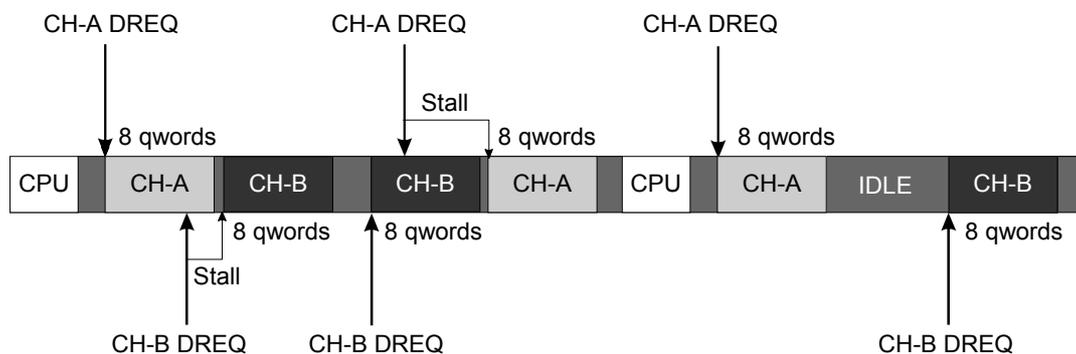
On the Slice Channel, data of the size specified by the Dn\_QWC register is divided into slices (blocks corresponding to 8-qword alignment in main memory). Whenever transfer of a slice ends, the remaining transfer data size and the next slice address are specified in Dn\_QWC and Dn\_MADR respectively, and the next DMA request is awaited.

For example, if 20 qwords of data are to be transferred, transfer on the Slice Channel is executed as follows, in three slices (four slices if the data extends over 8-qword alignment) while waiting for a request from the peripheral. In comparison, data is transferred continuously on the Burst Channel.



**Figure 5-1 Slice Channel and Burst Channel**

The following figure illustrates an example of concurrent DMA transfer in Slice Channel A and B.



**Figure 5-2 Arbitration on the Slice Channel**

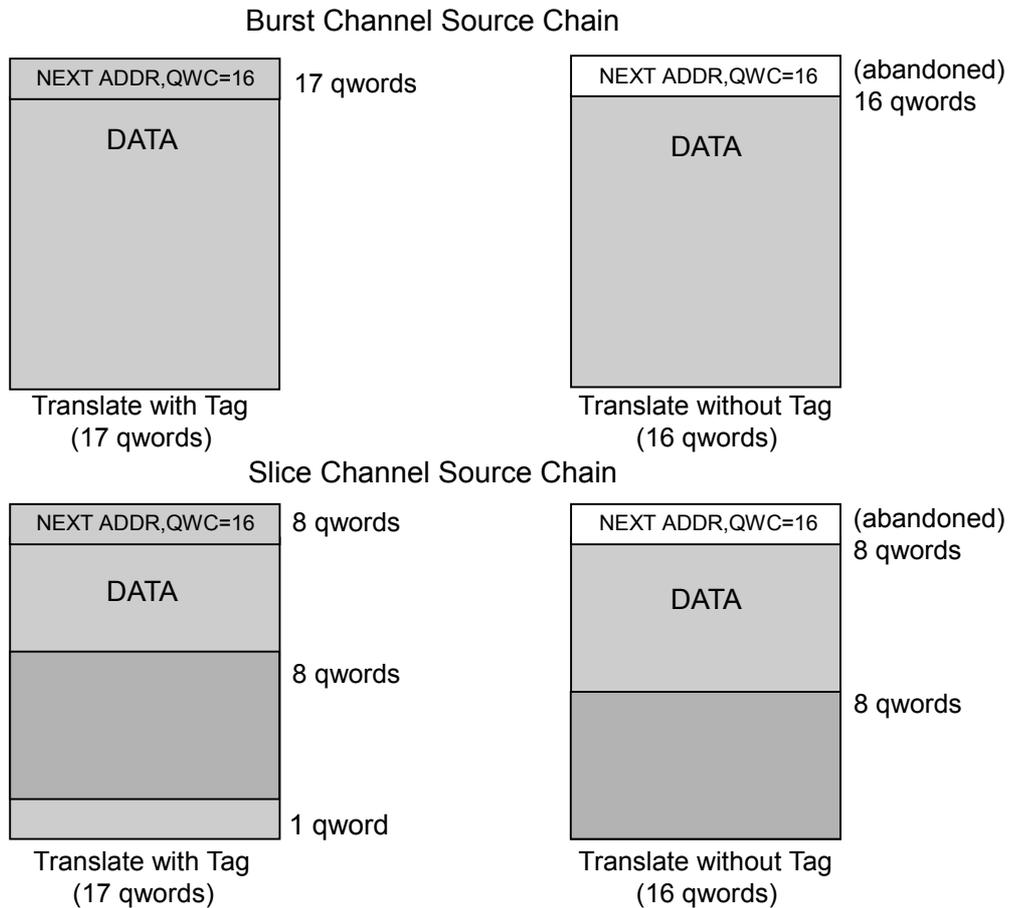
### 5.2.3. Source Chain Mode

Source Chain Mode can perform DMA transfer from memory to a peripheral and specifies the address and data size with the tag (DMAtag) in the transfer packet.

First, the DMAC transfers the data of the size specified by the Dn\_QWC register from the address specified by the Dn\_MADR register. When Dn\_CHCR.TAG is cnt, the DMAC reads 1 qword from the address specified by Dn\_TADR as a tag. The address and size of the data to be transferred next and the address of the tag to be read next are indicated in the tag. These values are stored in Dn\_MADR and Dn\_QWC respectively, the DMAC transfers data accordingly, and updates Dn\_TADR following the transfer.

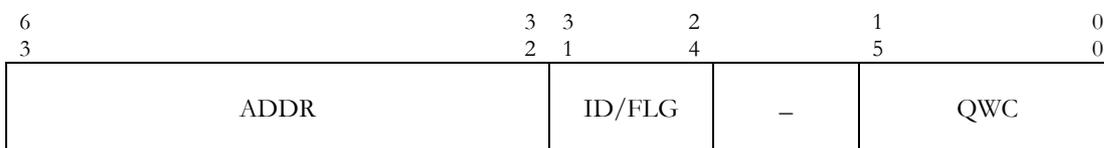
Transfer processing is repeated while following the tag in memory, as described above. A series of transfers is ended by clearing the STR bit to 0 at the end of processing of the tag that contains the end instruction.

The packet is transferred in 8-qword slices on the Slice channel in the same way as the transfer in the Normal mode. (To avoid complexity, this was excluded from the above description.) Whether to transfer the tag itself with the data or not can be specified by the TTE flag of the Dn\_CHCR register. The tag itself is transferred before the DMA transfer activated by the tag. The following figures illustrate each example of transferring data of 16 qwords on the Burst/Slice channel with/without a tag.



**Figure 5-3 Tag Transfers**

The tag is 128 bits (16 bytes) in length, and the lower 64 bits are effective. The format is as follows. ID is the field to show the details of the transfer operation and can specify eight types shown in the table below. For details, refer to "5.6. DMAtag".



### DMA Tags in Source Chain Mode

ID	Transfer data start address	Next tag address	Operation
cnt	Next to tag	Next to transfer data	Transfers the QWC qword following the tag and reads the succeeding qword as the next tag.
next	Next to tag	ADDR	Transfers the QWC qword following the tag and reads the qword of the ADDR field as the next tag.
ref	ADDR	Next to tag	Transfers the QWC qword from the ADDR field and reads the qword following the tag as the next tag.
refs	ADDR	Next to tag	Transfers the QWC qword from the ADDR field while controlling stalls and reads the qword following the tag as the next tag. Effective only on the VIF1, GIF, and SIF1 channels.
refe	ADDR	(None)	Transfers the QWC qword from the ADDR field, clears the Dn_CHCR.STR to 0, and ends transfer.
call	Next to tag	ADDR	Transfers the QWC qword following the tag, pushes the next field into the Dn_ASR register, and reads the qword of the ADDR field as the next tag. Effective only on the VIF0, VIF1, and GIF channels. Addresses can be pushed up to 2 levels.
ret	Next to tag	Dn_ASR	Transfers the QWC qword following the tag and reads the qword of the field popped from the Dn_ASR register as the next tag. Transfers the QWC qword following the tag, clears the Dn_CHCR.STR to 0, and ends transfer when there is no pushed address. Effective only on the VIF0, VIF1, and GIF channels.
end	Next to tag	(None)	Transfers the QWC qword following the tag, clears the Dn_CHCR.STR to 0, and ends transfer.

By using these IDs properly according to the data structure in the memory, data can be transferred efficiently taking advantage of the data structure. The following figures illustrate the above examples.

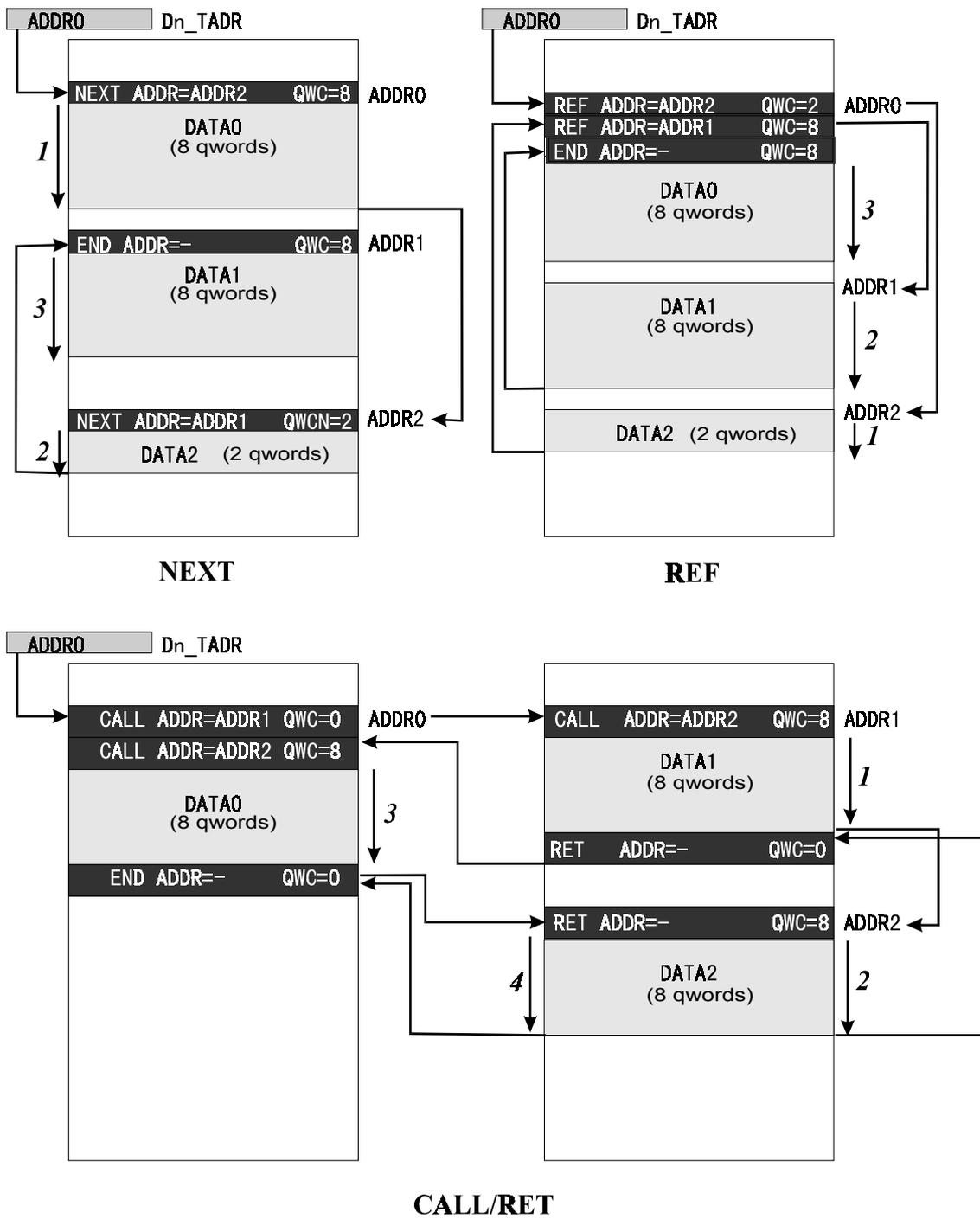


Figure 5-5 Data Structure by DMA Tags

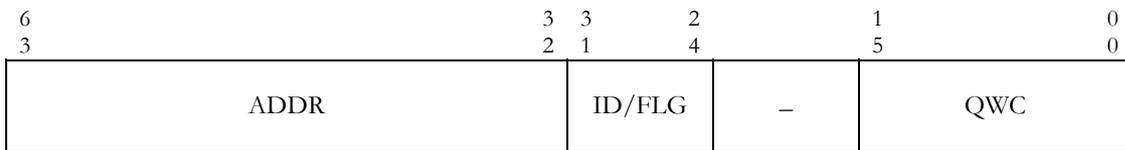
### 5.2.4. Destination Chain Mode

Destination Chain Mode can transfer data from a peripheral to memory. A tag (DMAtag) containing the destination address and the packet length is placed at the start of the transfer packet. As a result, the transfer address of main memory can be controlled on the peripheral side.

First, DMAC transfers data of the size specified by the Dn\_QWC register to the transfer address specified by the Dn\_MADR register. When the transfer ends, DMAC reads the next qword as a tag. The transfer destination address, data size, and address of the tag to be read next are indicated in the tag. DMAC sets the Dn\_MADR register according to this specification, and transfers the specified packet. The tag itself is not transferred to the memory. When the packet with the end instruction is transferred, a series of transfers ends by clearing the STR bit of the Dn\_CHCR register to 0.

Although not mentioned above (to avoid complexity), data is transferred in 8-qword slices on the Slice channel in the same way as the transfer in the Normal mode.

The Destination Chain Tag is 128 bits (16 bytes) in length, and the lower 64 bits are effective. The format is as follows. The ID field shows the details of the transfer operation and can specify three types shown in the table below. For details, refer to "5.6. DMAtag".



ID	Transfer destination address	Next tag	Operation
cnt	ADDR	Data next to transfer data	Transfers the QWC qword following the tag and reads the succeeding qword as the next tag.
cnts	ADDR	Data next to transfer data	Transfers the QWC qword following the tag and reads the succeeding qword as the next tag. Copies the Dn_MADR to the D_STADR successively while transferring the data. Effective only on the SIF0 and fromSPR channels.
end	ADDR	(None)	Transfers the QWC qword following the tag, clears Dn_CHCR.STR to 0, and ends transfer.

The following figures illustrate the above examples.

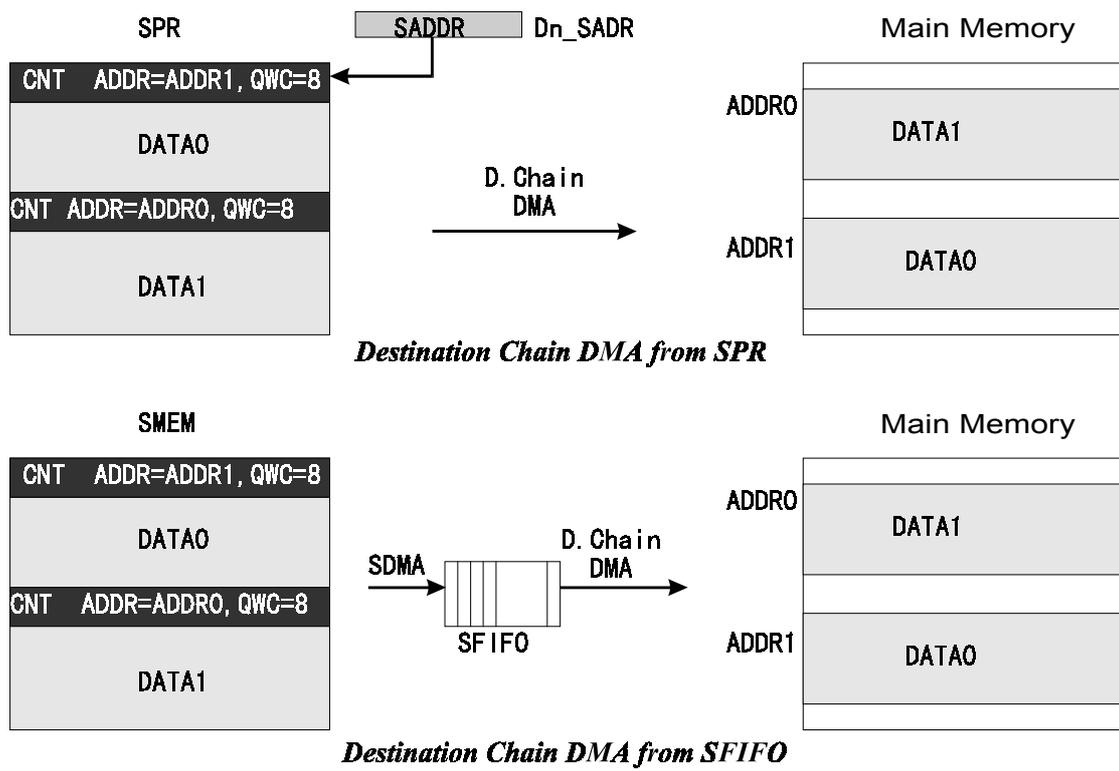


Figure 5-6 Data Transfer by Destination Chain Mode

### 5.2.5. Interleave Mode

Interleave mode can perform DMA transfers between main memory and scratchpad memory. It transfers data so as to cut out a small rectangular area from the two-dimensional data (image data) arranged in memory. Data of the size specified by the TQWC field of the D\_SQWC register is transferred first, and then data of the size specified by the SQWC field is skipped. This process is repeated until the end of the transfer of the data size specified by the Dn\_QWC register. The total transfer data size should be a multiple of D\_SQWC.TQWC. The process by which a small rectangular area of (TW, TH) is cut out from a rectangular area of (FW, FH) is shown as an example below.  $Dn\_QWC=TW*TH$ ,  $D\_SQWC.SQWC=FW-TW$ , and  $D\_SQWC.TQWC=TW$  are specified, and the DMA transfer of  $TW \times TH$  qword is performed from the upper left address in the small rectangular area in the Interleave mode.

Data is transferred in the same way when a small rectangular area of  $TW \times TH$  is read from the scratchpad memory and is transferred so as to be embedded into the rectangular area of (FW, FH) in the main memory.

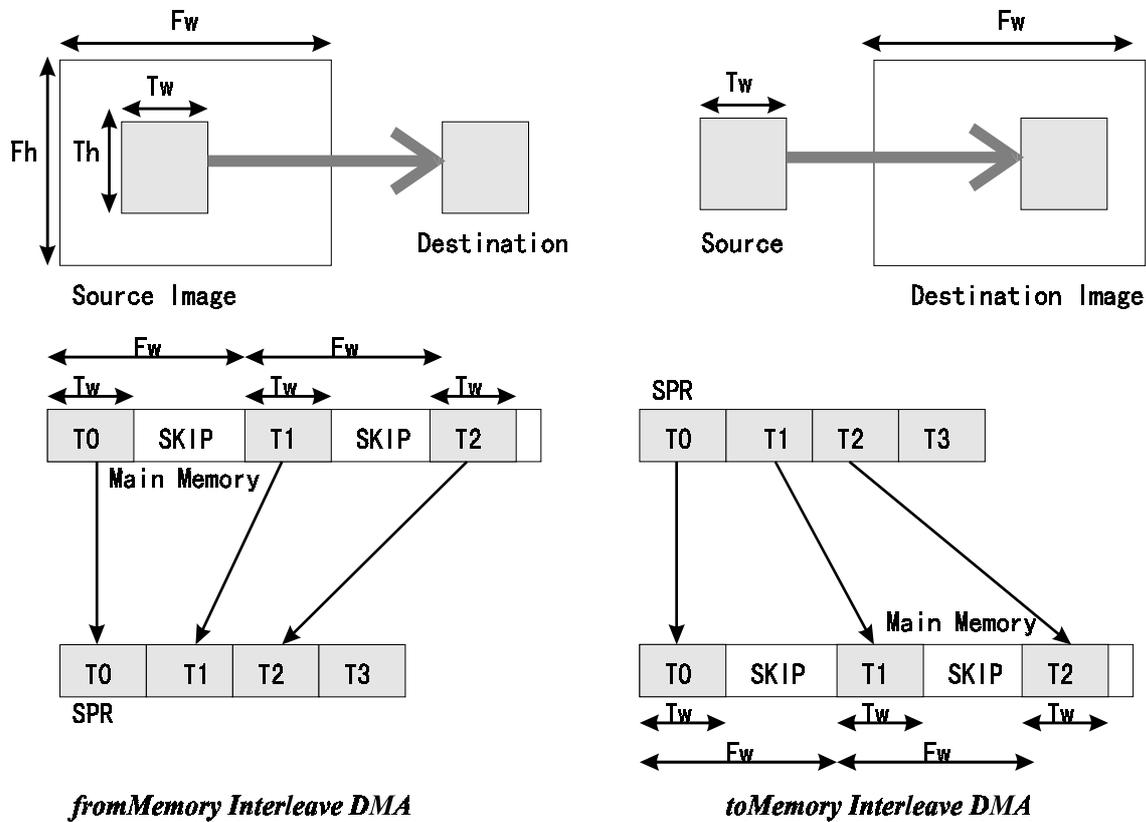


Figure 5-7 Data Transfer in Interleave Mode

## 5.3. Arbitration of Bus Right

Arbitration of the bus right is performed according to the priorities previously assigned to each of the channels. The priorities can also be set by the stall control due to the transfer address and by the tag.

### 5.3.1. Arbitration in Order of Priority

When there are two or more DMA requests, the request from the highest-priority channel is selected. Priorities are assigned as shown in the table below. All channels in Group C are treated equally in terms of priority: the channel selection is made by round robin.

Priority	ID	Channel
A	0	VIF0
B	7	SIF2
C	1	VIF1
C	2	GIF
C	3	fromIPU
C	4	toIPU
C	5	SIF0
C	6	SIF1
C	8	fromSPR
C	9	toSPR

The priority assigned to the bus request from the EE Core is the highest in the Cycle Steal Mode (D\_CTRL.RELE=1) and becomes the lowest in other modes.

The DMA channel releases the bus right temporarily under the following conditions on the way of data transfer, and performs transfer processing according to the priorities assigned to unprocessed DMA requests at that time.

- Slice boundary during the transfer via Slice Channel
- Packet boundary during the transfer in Chain Mode
- Interleave boundary during the transfer in Interleave Mode

### 5.3.2. Stall Control

When data are transferred concurrently from a peripheral to memory and from memory to another peripheral, arbitration between them can be performed via the stall address register (D\_STADR).

At that time, the channel that performs the DMA transfers to memory is called the source channel and the channel which performs the DMA transfer from memory is called the drain channel, and the registers are shown as D<sub>s</sub>\_MADR and D<sub>d</sub>\_MADR respectively. Only one combination of source and drain channels can be selected from the following channels (excluding the combination of SIF0 for source and SIF1 for drain).

Source (Input)
FromIPU (IPU→Memory)
FromSPR (SPR→Memory)
SIF0 (SIF0→Memory)

Drain (Output)
VIF1 (Memory→VIF1)
SIF1 (Memory→SIF1)
GIF (Memory→GIF)

Suppose transfers of SIF0 → Memory and Memory → VIF1 are executed at the same time. The transfer address of SIF0 → Memory is temporarily copied to D\_STADR. And if the transfer address of Memory → VIF1 is larger than this, the transfer is held. As a result, it is guaranteed that the transfer of Memory → VIF1 does not outstrip the transfer of the SIF0 → Memory.

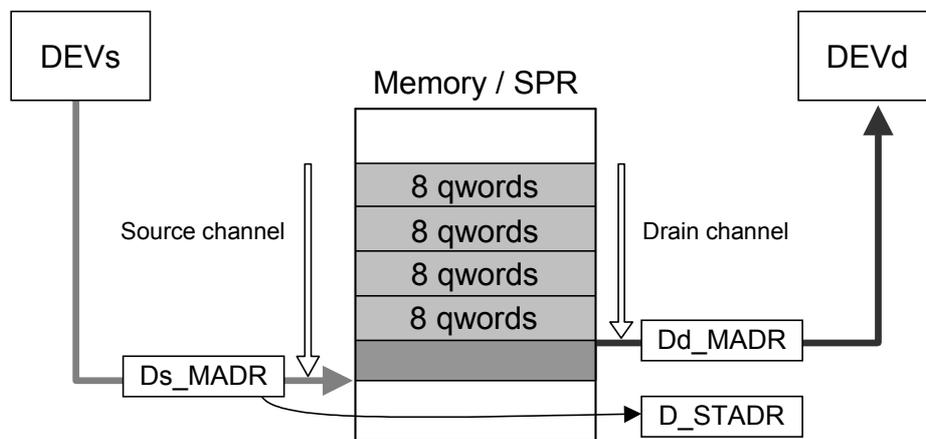
Concretely, stall control is performed according to the following procedure:

- The stall address, source channel, drain channel are stored in the D\_STADR register, the STS field of the D\_CTRL register, and the STD field, respectively.
- When the source channel transfers the packet data targeted for stall control, the values of Ds\_MADR are copied to D\_STADR in succession after the end of transfer. The packets targeted for stall control in the source channel are the transfer data in the Normal Mode and the packet data of ID=cnts in the Destination Chain Mode.
- If the stall condition is met in the first word of the slice when the drain channel is about to transfer the packet data targeted for stall control, the DMA transfer request is disregarded.

The packets targeted for stall control in the drain channel are the transfer data in the Normal Mode and the packet data of ID=refs in the Source Chain Mode. When stall conditions are in effect, DMA transfer in the Slice Mode determines the stall conditions in 8-qword units. DMA transfer via the drain channel stalls when the following stall conditions are established.

$$Dd\_MADR.ADDR + 8 > D\_STADR.ADDR$$

Note that stall conditions are met at the end of the data and that a fraction less than 8 qwords is not transferred to the drain channel, if the data size provided by the source channel is not in 8-qword units.



**Figure 5-9 Stall Control**

Following are some examples of the transfer process in which stall control is performed.

### Normal to Normal

This is an example of transfer processing from ch\_s to ch\_d both in Normal Mode.

This is not shown here, but the D\_CTRL.STD has been cleared to 0 by the termination interrupt of ch\_s.

```

D_CTRL.STS = ch_s;           /* Source Channel Setting */
D_CTRL.STD = ch_d;           /* Drain Channel Setting */

Ds_CHCR.MOD = NORMAL;       /* Source: Normal Mode */
Dd_CHCR.MOD = NORMAL;       /* Drain: Normal Mode */

Ds_MADR = SOURCE_ADDR;      /* Source Setting */
Ds_QWC = SOURCE_QWC;

Dd_MADR = DRAIN_ADDR;       /* Drain Setting */
Dd_QWC = DRAIN_QWC;

Ds_CHCR.STR = 1;            /* Source Start */
Dd_CHCR.STR = 1;            /* Drain Start */

```

### Chain to Normal

This is an example of transfer processing from the ch\_s in Destination Chain Mode to the ch\_d in Normal Mode.

The stall address is updated when the cnts tag is executed on the ch\_s side.

```

D_CTRL.STS = ch_s;           /* Source Channel Setting */
D_CTRL.STD = ch_d;           /* Drain Channel Setting */

Ds_CHCR.MOD = D_CHAIN;      /* Source: D.Chain Mode */
Dd_CHCR.MOD = NORMAL;       /* Drain: Normal Mode */

Dd_MADR = DRAIN_ADDR;       /* Drain Setting */
Dd_QWC = DRAIN_QWC;

D_STADR = STALL_START_ADDR; /* Stall Address Initial Value Setting */

Ds_CHCR.STR = 1;            /* Source Start */
Dd_CHCR.STR = 1;            /* Drain Start */

```

### Normal to Chain

This is an example of transfer processing from the `ch_s` in Normal Mode to the `ch_d` in Source Chain Mode. Transfer normally proceeds without stalls. However, when `refs` and `cnts` tags are executed via `ch_d`, the `Dd_MADR` and the `D_STADR` are compared while transferring the following packet. This results in stalls if the stall conditions have been met.

```
D_CTRL.STS = ch_s;          /* Source Channel Setting */
D_CTRL.STD = ch_d;          /* Drain Channel Setting */

Ds_CHCR.MOD = NORMAL;      /* Source: Normal Mode */
Dd_CHCR.MOD = S_CHAIN;     /* Drain: S.Chain Mode */

Ds_MADR = SOURCE_ADDR;     /* Source Setting */
Ds_QWC = SOURCE_QWC;

Dd_TADR = DRAIN_TADDR;     /* Drain Setting */

Dd_CHCR.STR = 1;           /* Drain Start */
Ds_CHCR.STR = 1;           /* Source Start */
```

### Chain to Chain

This is an example of transfer processing from the `ch_s` in Destination Chain Mode to the `ch_d` in Source Chain Mode.

Transfer normally proceeds without stalls, and when the `cnts` tag is executed via `ch_s`, the `D_STADR` register is updated. When `refs` and `cnts` tags are executed via `ch_d`, the `Dd_MADR` and the `D_STADR` are compared while transferring the following packet. This results in stalls if the stall conditions have been met.

```
Ds_CHCR.MOD = D_CHAIN;     /* Source: D.Chain Mode */
Dd_CHCR.MOD = S_CHAIN;     /* Drain: S.Chain Mode */

Dd_TADR = DRAIN_TADDR;     /* Drain Setting */
D_STADR = STALL_START_ADDR; /* Stall Monitor Buffer Address Setting */

Ds_CHCR.STR = 1;           /* Source Start */
Dd_CHCR.STR = 1;           /* Drain Start */
```

### 5.3.3. Priority Setting by Tag

While a specific packet on a specific channel is transferred, DMA via other channels can temporarily be suspended by controlling the PCE (Priority Control Enable) bit of the DMA tag.

To perform this control, the channel judged stoppable should be specified in advance by setting the CDEn bit to 0 in the `D_PCR` register. This specification becomes valid when the PCE field of the tag is set to 11 and becomes invalid when it is set to 10. As a result, only while a specific packet is transferred, transfers via other channels can be controlled.

However, priorities cannot be controlled individually by this function. The reason is that the SIF0, SIF1, and SIF2 share the FIFO and might cause a deadlock.

## 5.4. Interrupt by DMA

There are four types of interrupts caused by DMA: Channel Interrupt / DMA Stall Interrupt / Memory FIFO Empty Interrupt / BUSERR Interrupt.

The status bits (CIS9-CIS0, SIS, MEIS, BEIS) of the D\_STAT register are set by the following conditions. If one of the results from ANDing these bits and the corresponding mask bits (CIM9-0, SIM, MEIM) becomes 1, an INT1\* interrupt is asserted.

The interrupt status bits are set under the following conditions:

### **CISn: End of transfer**

When transfer of the number of words specified in Dn\_QWC ends in Normal Mode or Interleave Mode:

Dn\_CHCR.STR=0 and D\_STAT.CIS[n]=1 are set.

Transfer processing ends.

### **CISn: End of transfer**

When the tag whose ID is end or refe is read in Chain Mode and transfer of the following packet ends:

Dn\_CHCR.STR=0 and D\_STAT.CIS[n]=1 are set.

Transfer processing ends and bits 31-16 of the end tag are maintained in Dn\_CHCR.

### **CISn: Overflow with call stack**

When the call tag is read in Source Chain Mode on the VIF0/VIF1/GIF channels when

Dn\_CHCR.ASP=10:

Dn\_CHCR.STR=0 and D\_STAT.CIS[n]=1 are set.

Transfer processing ends without transferring the call packet, and the address of the call tag is maintained in Dn\_TADR and bits 31-16 of the call tag are maintained in the Dn\_CHCR.

### **CISn: Underflow with call stack**

When the ret tag is read in Source Chain Mode in the VIF0/VIF1/GIF channels when Dn\_CHCR.ASP=00, and the transfer of the following packet ends:

Dn\_CHCR.STR=0 and D\_STAT.CIS[n]=1 are set.

Transfer processing ends, and bits 31-16 of the ret tag are maintained in Dn\_CHCR.

### **CISn: Undefined tag**

When the tag with undefined ID is read in Chain Mode:

Dn\_CHCR.STR=0 and D\_STAT.CIS[n]=1 are set.

Transfer processing ends. In Source Chain Mode, the read tag address is maintained in Dn\_TADR and bits 31-16 of the tag are maintained in Dn\_CHCR. In Destination Chain Mode, bits 31-16 of the tag are maintained in Dn\_CHCR.

### **CISn: Tag interrupt**

When the tag of IRQ=1 is read in Chain Mode when Dn\_CHCR.TIE=1 and the transfer of the following packet ends:

Dn\_CHCR.STR=0 and D\_STAT.CIS[n]=1 are set.

Transfer processing is interrupted. In Source Chain Mode, the following tag address is maintained in Dn\_TADR and bits 31-16 of the tag are maintained in Dn\_CHCR. In Destination Chain Mode, bits 31-16 of the tag are maintained in Dn\_CHCR.

**BEIS, CISn: Bus error**

When the memory outside the area is about to be DMA-transferred:

Dn\_CHCR.STR=0, D\_STAT.BEIS=1, and D\_STAT.CIS[n]=1 are set.

The transfer is interrupted. In Chain Mode, bits 31-16 of the tag read most recently are maintained in Dn\_CHCR.

**SIS: DMA stall**

When DMA stall occurs when D\_CTRL.STD! =00 and the transfer via the drain channel is held:

D\_STAT.SIS=1 is set.

Dd\_CHCR.STR is not cleared to 0. When the stall condition is removed, the transfer processing is restarted.

However, SIS bit does not change even if the transfer restarts.

**MEIS: MFIFO empty**

When the MFIFO empties when D\_CTRL.MFD! =00 and the transfer via the drain channel is held:

D\_STAT.MEIS=1 is set.

Dd\_CHCR.STR is not cleared to 0, and the MEIS does not change even after the restart of transfer.

The main causes for generating INT1\* interrupts are as follows:

MEIS	SIS	CIS	BEIS	MOD	IRQ	ID	ASP	Cause
1	-	-	-	-	-	-	-	MFIFO Empty
-	1	-	-	-	-	-	-	DMA Stall
-	-	1	-	00	-	-	-	end of Normal
-	-	1	-	01	-	end	-	end of Chain
-	-	1	-	10	-	end	-	end of Interleave
-	-	1	-	01	-	ret	00	end of Chain
-	-	1	-	01	0	call	10	stack overflow
-	-	1	-	01	1	call	10	*1
-	-	1	-	01	-	*3	-	undefined tag-ID
-	-	1	-	01	1	-	-	tag interrupt
-	-	1	1	-	-	-	-	BUSERR*2

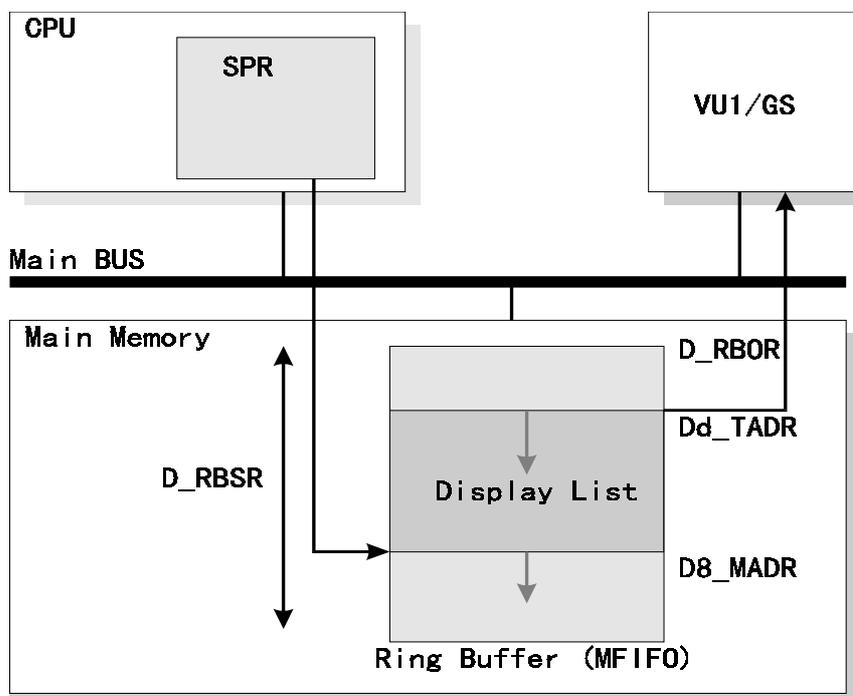
\*1: Whether it is a tag interrupt or a call stack overflow is determined by Dn\_TADR and Dn\_MADR.

\*2: In the case of BEIS=1, if there are two or more channels of CIS=1, the determination is made from Dn\_MADR of each channel.

\*3: Any codes that are not defined as a tag.

## 5.5. MFIFO

In transferring data from scratchpad memory to VIF1/GIF, the FIFO function can be implemented by using a ring buffer in main memory and the DMA tag. This is called MFIFO (MemoryFIFO).



**Figure 5-10 MFIFO**

The ring buffer area is shown with the D\_RBOR register and the D\_RBSR register.

The fromSPR channel (ch-8) becomes the source channel, and the transfer mode is limited to Normal Mode. After checking that there is a space in the ring buffer, data is transferred to the ring buffer in units of the data equivalent to the packet on the drain channel side.

The VIF1 channel (ch-1) or the GIF channel (ch-2) becomes the drain channel. The transfer mode is limited to Source Chain Mode, and the useable tags are limited to cnt, ref, refs, refe, and end.

If  $Dd\_TADR = D8\_MADR$  when the drain channel reads the tag of Dd\_TADR and processes the data, the transfer is held. The Dd\_MADR shows the actual transfer address, and when applying a wraparound to the ring buffer, the Dd\_MADR is also updated to a wraparound-applied address automatically by hardware.

Avoid changing the D\_RBOR/D\_RBSR register values while data is being transferred on the drain channel. This interferes with the operation.

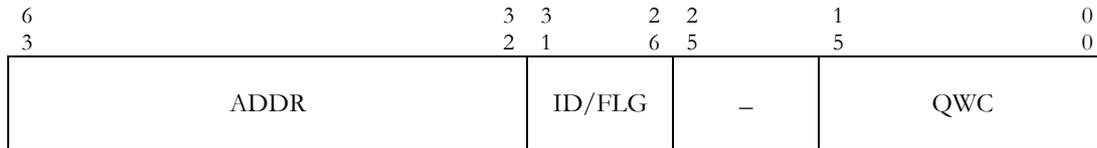
### Restrictions

Do not place ref/refs/refe tag at the end of the packet that is sent from the source channel. Doing so suspends data transfer by establishing the formula  $Dd\_TADR = D8\_MADR$  immediately after the drain channel reads the tag and starts transferring data. (At this time, MFIFO empty interrupt status (MEIS) is not detected.) As a method of avoiding this, ref and refs tags can be followed by a dummy cnt tag (QWC=0) and a refe tag can be replaced with a combination of ref and end tags.

## 5.6. DMAtag

The DMAtag is used in Chain Mode to control the destination/source memory address, transfer size, etc. There are two types: Source Chain Tag and Destination Chain Tag.

Both types have the format roughly outlined below:





ID	Transfer data address	Next tag address	Operation
cnt	Next to tag	Next to transfer data	Transfers the QWC qword following the tag and reads the succeeding qword as the next tag.
next	Next to tag	ADDR	Transfers the QWC qword following the tag and reads the qword of the ADDR field as the next tag.
ref	ADDR	Next to tag	Transfers the QWC qword from the ADDR field and reads the qword following the tag as the next tag.
refs	ADDR	Next to tag	Transfers the QWC qword from the ADDR field while controlling stalls and reads the qword following the tag as the next tag. Effective only on the VIF1, GIF, and SIF1 channels.
refe	ADDR	(None)	Transfers the QWC qword from the ADDR field, clears the Dn_CHCR.STR to 0, and ends transfer.
call	Next to tag	ADDR	Transfers the QWC qword following the tag, pushes the next field into the Dn_ASR register, and reads the qword of the ADDR field as the next tag. Effective only on the VIF0, VIF1, and GIF channels
ret	Next to tag	Dn_ASR	Transfers the QWC qword following the tag and reads the qword of the field popped from the Dn_ASR register as the next tag. Transfers the QWC qword following the tag, clears the Dn_CHCR.STR to 0, and ends transfer when there is no pushed address. Effective only on the VIF0, VIF1, and GIF channels.
end	Next to tag	(None)	Transfers the QWC qword following the tag, clears the Dn_CHCR.STR to 0, and ends the transfer.



## 5.7. Suspending and Restarting DMA Transfer

Note the following when suspending or restarting the DMA transfer in process for debugging and MFIFO switching purposes.

- A DMA transfer may not stop properly just by rewriting the STR bit of the Dn\_CHCR register to 0. At this time, other bits cannot be accessed. Access the Dn\_CHCR register after suspending the DMAC (via the D\_ENABLER/D\_ENABLEW register) by performing the following procedure.
  1. Disable interrupts.
  2. Read D\_ENABLER.
  3. Perform OR between the result of step 2 and 0x00010000.
  4. Write the data obtained from the OR operation to D\_ENABLEW.  
< DMA transfer is suspended. >
  5. Perform two or more uncached read operations.  
(Reading Dn\_CHCR and D\_CTRL is appropriate.)
  6. Set the STR bit of the Dn\_CHCR register (or the DMAE bit of the D\_CTRL register) to 0.  
(Access other bits as necessary.)
  7. Write the data obtained from the read operation in step 2 to D\_ENABLEW.  
< DMA transfer restarts. >
  8. Recover from the interrupt-disabled state.
- The upper 16 bits of the Dn\_CHCR register maintain bits 31 to 16 of the DMA tag in process. They must be set correctly again when the transfer restarts.

## 5.8. Common Registers

The following common registers perform control extending over all the channels.

Name	r/w	Width	Contents
D_CTRL	r/w	32 bits	DMA Control Register
D_STAT	r/w	32 bits	Interrupt Status Register
D_PCR	r/w	32 bits	Priority Control Register
D_SQWC	r/w	32 bits	Interleave Size Register
D_RBOR	r/w	32 bits	Ring Buffer Address Register
D_RBSR	r/w	32 bits	Ring Buffer Size Register
D_STADR	r/w	32 bits	Stall Address Register
D_ENABLER	r/-	32 bits	DMA Hold Status Register
D_ENABLEW	-/w	32 bits	DMA Hold Control Register

**D\_CTRL : DMA control register**

r / w

3	1 1	0 0 0 0 0 0 0 0							
1	1 0	8 7 6 5 4 3 2 1 0							
-			RCYC	S T D	S T S	M F D	R E L E	D M A E	
0			-	-	-	-	-	-	0

Name	Pos.	Contents
DMAE	0	DMA enable 0 Disables all DMAs 1 Enables all DMAs
RELE	1	Release signal enable 0 Cycle Stealing off 1 Cycle Stealing on
MFD	3:2	Memory FIFO drain channel 00 Does not use MFIFO function 01 Reserved 10 VIF1 channel (ch-1) 11 GIF channel (ch-2)
STS	5:4	Stall Control source channel 00 Non-specified (Does not update D_STADR) 01 SIF0 channel (ch-5) 10 fromSPR channel (ch-8) 11 fromIPU channel (ch-3)
STD	7:6	Stall Control drain channel 00 Does not perform stall control 01 VIF1 channel (ch-1) 10 GIF channel (ch-2) 11 SIF1 channel (ch-6)
RCYC	10:8	Release Cycle 000 8 001 16 010 32 011 64 100 128 101 256

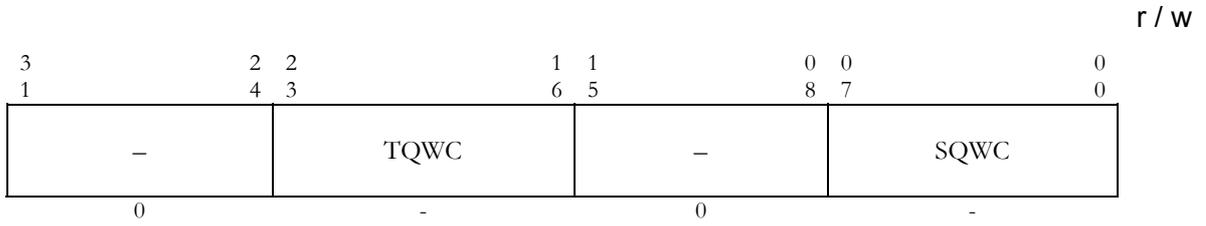
The RCYC field sets the period to release the bus to the EE Core when Cycle Stealing is on.

For the procedure for suspending and restarting a DMA transfer, see "5.7. Suspending and Restarting DMA Transfer".





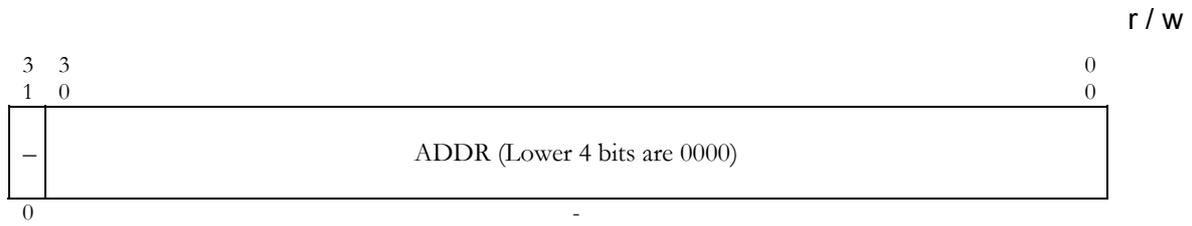
### D\_SQWC : Interleave size register



Name	Pos.	Contents
SQWC	7:0	Skip quadword counter Size of the part not transferred (qword)
TQWC	23:16	Transfer quadword counter Size of the part transferred (qword)

This register specifies the size of a small rectangular area to be transferred in Interleave mode. For details, refer to "5.2.5. Interleave Mode".

## D\_RBOR : Ring buffer address register



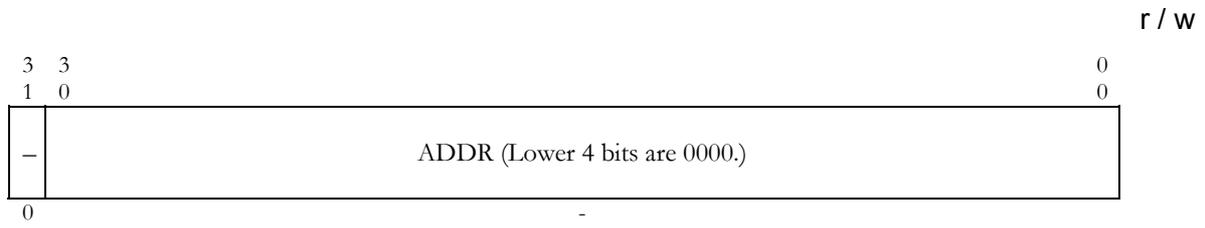
Name	Pos.	Contents
ADDR	30:0	Ring buffer offset address (qword alignment)

This register sets the starting address of the MFIFO ring buffer. The starting address of the ring buffer must be aligned on the boundary of the ring buffer size specified by the D\_RBSR register. That is, the setting is made so that D\_RBOR.ADDR&D\_RBSR.RMSK becomes 0.

Changing the value of this register while data is being transferred on the drain channel may cause the operation to be unstable.



## D\_STADR : Stall address register

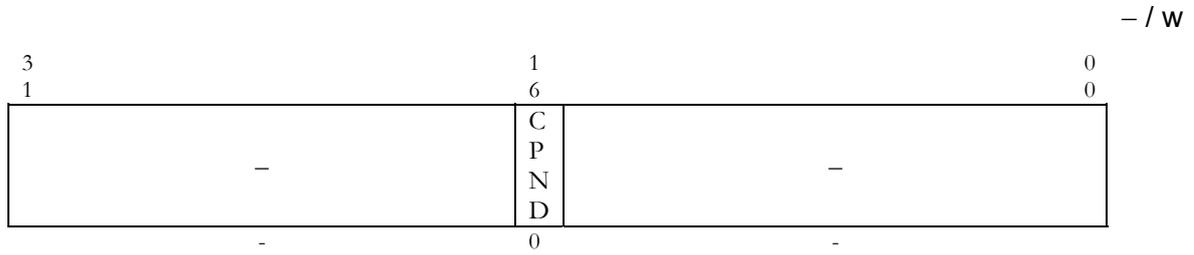


Name	Pos.	Contents
ADDR	30:0	Stall address (qword alignment)

This register maintains the stall control address.

When D\_CTRL.STS != 00, the value of the Dn\_MADR of the source channel is copied whenever a transfer is performed and the stall is controlled in comparison with the Dn\_MADR of the drain channel specified by the D\_CTRL.STD. For details, refer to "5.3.2. Stall Control".

**D\_ENABLEW : DMA hold control register**

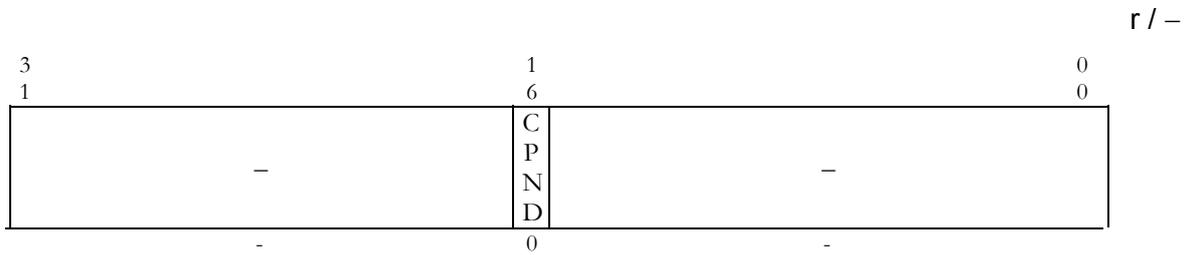


Name	Pos.	Contents
CPND	16	DMA transfer is held 0 Enables all channel transfer(restarts) 1 Holds all channel transfer(suspends)

This register controls suspension and restarting of the DMA transfer. When D\_ENABLEW.CPND=1, the DMAC transfer is suspended. If D\_ENABLEW.CPND=0, the DMAC operation is restarted.

To clear the STR bit of each channel (DnCHCR.STR) to discontinue the transfer, stop DMAC before starting the operation. For details of the procedure, see "5.7. Suspending and Restarting DMA Transfer".

The contents of the D\_ENABLEW register are readable with the D\_ENABLER register.

**D\_ENABLER : DMA hold state register**

Name	Pos.	Contents
CPND	16	DMA transfer hold state 0 All channel transfer enabled 1 All channel transfer being held (suspended)

This register indicates status of the DMA transfer hold state (enabled or temporarily suspended). For details, see "5.7. Suspending and Restarting DMA Transfer".

## 5.9. Channel Registers

The registers allocated to each of the channels are as follows.

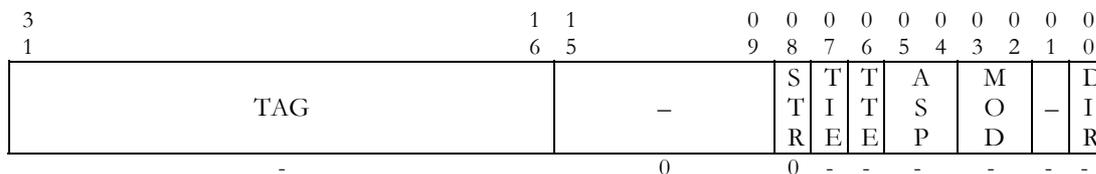
Name	r/w	Width	Contents
Dn_CHCR	r/w	32 bits	Channel Control Register
Dn_MADR	r/w	32 bits	Transfer Address Register
Dn_QWC	r/w	32 bits	Transfer Data Size Register
Dn_TADR	r/w	32 bits	Tag Address Register
Dn_ASR0, Dn_ASR1	r/w	32 bits	Tag Address Save Register
Dn_SADR	r/w	32 bits	SPR Transfer Address Register

Dn\_TADR, Dn\_ASR[0-1], and Dn\_SADR exist only for some channels.

ID	Channel	TADR	ASR0/1	SADR
0	VIF0	Yes	Yes	No
1	VIF1	Yes	Yes	No
2	GIF	Yes	Yes	No
3	fromIPU	No	No	No
4	toIPU	Yes	No	No
5	SIF0	No	No	No
6	SIF1	Yes	No	No
7	SIF2	No	No	No
8	fromSPR	No	No	Yes
9	toSPR	Yes	No	Yes

## Dn\_CHCR : Channel control register

r / w



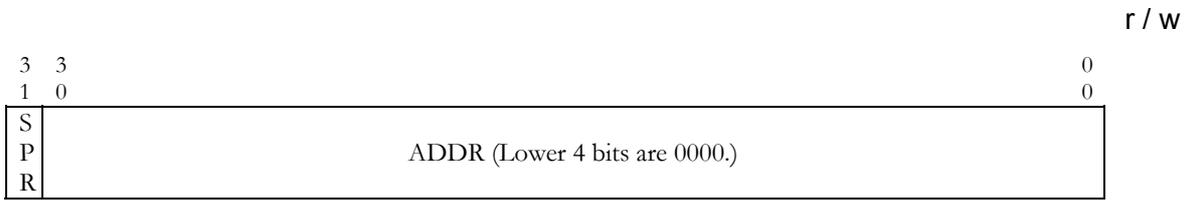
Name	Pos.	Contents
DIR	0	Direction 0 to Memory 1 from Memory Effective only on VIF1(ch-1) and SIF2(ch-7) channels
MOD	3:2	Mode 00 Normal 01 Chain 10 Interleave
ASP	5:4	Address stack pointer 00 No address pushed by call tag 01 1 address pushed 10 2 addresses pushed
TTE	6	Tag transfer enable 0 Does not transfer DMAtag itself. 1 Transfers DMAtag. Effective only in Source Chain mode
TIE	7	Tag interrupt enable 0 Disables IRQ bit of DMAtag. 1 Enables IRQ bit of DMAtag.
STR	8	Start 0 Stops DMA. Maintains 0 while stopping. 1 Starts DMA. Maintains 1 while operating.
TAG	31:16	DMAtag Maintains bits 31 to 16 (IRQ/ID/PCE) of the DMAtag read most recently in the Chain Mode.

The fields other than the STR can be written only while DMA is stopped.

The DMA transfer in process may not stop properly just by rewriting the STR bit to 0. Access the STR bit after stopping the DMAC via the D\_ENABLEW register. The contents of the TAG field must be set correctly again when the transfer restarts. For details of the procedure, see "5.7. Suspending and Restarting DMA Transfer".

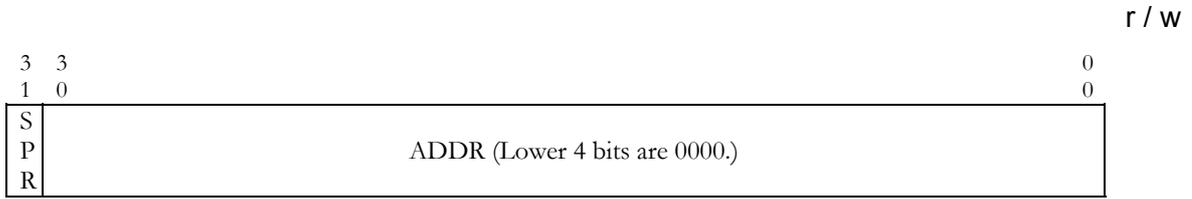


## Dn\_TADR : Tag address register



Name	Pos.	Contents
ADDR	30:0	Next tag address Address of the next tag (qword alignment)
SPR	31	Next memory/SPR Selection 0     Memory address 1     SPR address

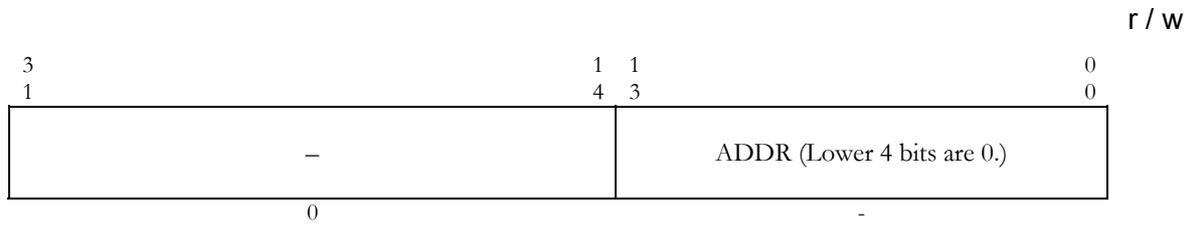
## Dn\_ASRO / Dn\_ASR1 : Tag address save register



Name	Pos.	Contents
ADDR	30:0	Tag memory address Tag address pushed by call tag (qword alignment)
SPR	31	Memory/SPR Selection 0     Memory address 1     SPR address

This is the stack register, which saves the tag address when the call tag is read. Since there are two registers, nesting of calls can be up to 2 levels.

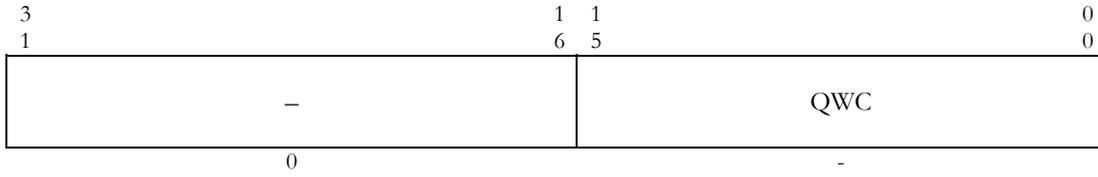
### Dn\_SADR : SPR transfer address register



Name	Pos.	Contents
ADDR	13:0	SPR address Transfer address of scratchpad memory (qword alignment)

This register specifies the scratchpad memory address on the fromSPR channel (ch-8) and the toSPR channel (ch-9).

### **Dn\_QWC : Transfer data size register**



Name	Pos.	Contents
QWC	15:0	QuadWord Counter Transfer data size (qword)

In DMA transfer in Normal or Interleave mode, performing DMA Kick (writing 1 to Dn\_CHCR.STR) with Dn\_QWC set to 0 will result in unsuccessful transfer.

## 5.10. Restrictions

### **Restrictions on concurrent transfers on the ch-1 and ch-9 channels**

A stall may occur, when transferring data concurrently on the toVPU1 (ID = 1) channel with D1\_CHCR.TTE set to 1 (which transfers DMAtag) and on the toSPR (ID = 9) channel with D9\_CHCR.TTE set to 0 (which does not transfer DMAtag) both in source chain mode.

Therefore, do not start DMA transfer in the above settings.

### **Restrictions on size specification when starting transfer**

In DMA transfer in Normal or Interleave mode, performing DMA Kick (writing 1 to Dn\_CHCR.STR) with Dn\_QWC set to 0 will result in unsuccessful transfer. Avoid performing such a process.

## 6. VPU: Vector Operation Processor

---

The EE has two built-in VPUs, which are floating-point vector processors that perform high-speed matrix operations, coordinate conversion, transparency perspective conversion, etc.

This chapter explains the structure of the VPUs and the methods of controlling them.

For details of the structure and instructions of the VU, which serves as the center of the VPU, refer to the separate volume "VU User's Manual".

## 6.1. VPU Overview

### 6.1.1. VPU Structure

The VPU consists of the VU, a floating-point vector processor unit, and the VIF, a DMA transfer data interface unit.

The floating-point vector processor unit, VU, has four floating-point product sum ALUs (FMACs) and an independent floating-point division/square root calculation unit (FDIV) built into it. By using the floating-point registers structured in 128-bit units, and the built-in data memory (VU Mem), the VU can execute floating-point vector operations of 32-bit single precision floating-point number x four elements concurrently. Moreover, the VU has 16-bit integer registers and an integer ALU (IALU) and can calculate the address of built-in data memory, loop counter, etc. In addition, the VU has a built-in instruction memory (MicroMem), and can execute programs of 64-bit length microinstructions.

The VIF can efficiently decompress DMA-transferred packets of different data lengths in the VU Mem, which is structured in 128-bit (32 bits x 4) vector units. In addition, the VIF has the ability to start a VU microinstruction program.

### 6.1.2. VPU0 and VPU1

Two VPUs are built into the EE: VPU0 and VPU1.

VPU0 is tightly coupled as a coprocessor (COP2) of the EE Core. The VU (VU0) of VPU0 can execute macroinstructions as a coprocessor, besides operating with microprograms. The CPU shares the floating-point registers and integer registers of VU0 as coprocessor-registers.

VPU1 works chiefly as a preprocessor of the GS. It is designed to process three-dimensional object data geometrically and generate GS commands. The converted/generated GS commands are sent directly to the GS through the GIF. In addition to the shared VU calculation unit, VPU1 has an independent elementary function unit (EFU) with one FMAC and one FDIV embedded and can process elementary function operations necessary for coordinate conversion etc. independently of other calculation units.

The main differences between VPU0 and VPU1 are as follows:

Function	VPU0	VPU1
Coprocessor connection	COP2	No
Macroinstruction	Yes	No
Built-in data memory (VU Mem)	4 KB	16 KB
Built-in instruction memory (MicroMem)	4 KB	16 KB
Elementary function unit (EFU)	No	Yes
GS direct output path (GIF)	No	Yes

### 6.1.3. VIF

The VIF is the interface unit that decompresses the data transferred by DMA in packets and transfers them to the memory in the VPU. Besides the data transfer between main memory and VU Mem or MicroMem, data can be transferred directly from the VIF to the GIF by bypassing the memory in the VPU.

The VIF is designed to set the decompression method and transfer destination address of the data, which follows the code, according to the 32-bit VIFcode included in the VIF packet. It is possible to perform VU processing independently of the CPU by embedding in the DMA packet chain the VIF packet of vector data, the VIF packet of the microinstruction program to process the vector data, and the VIF packet to issue the instruction to activate the microinstruction program.

The data types the VIF can decompress and transfer to the VU Mem are one- to four-dimensional vectors consisting of 8-bit/16-bit/32-bit elements and a four dimensional vector of 16-bit color type with RGBa: 5.5.5.1. Moreover, the VIF can transfer the microinstruction code to be transferred to the MicroMem. In VIF1, the GS data to be transferred to the GS via the GIF can also be transferred.

Two 128-bit vector data registers and some control registers are built into the VIF for data decompression purposes. By using these registers, initialization of the VU Mem, supplementation processing of the decompressed data, double buffering by the address offset (only in VIF1), etc. can be executed. Moreover, the VIF has an addition function for the input vector data and the built-in vector data register and can process the difference data and the offset data.

## 6.2. VU Mem /MicroMem

The VU Mem is a built-in VU data memory, and the MicroMem is a built-in VU instruction memory. VU Mem0 (4 Kbytes) and MicroMem0 (4 Kbytes) are built into VU0, and VU Mem1 (16 Kbytes) and MicroMem1 (16 Kbytes) are built into VU1. The memory map is shown below.

### 6.2.1. VU Mem0 Memory Map

VU Mem0 is 4 Kbytes (256 qwords) in size and is mapped as follows. Although VU Mem0 is also mapped to main memory as shown in the right-hand part of the figure, it is accessible from the EE Core only when both VU0 and VIF0 are stopped.

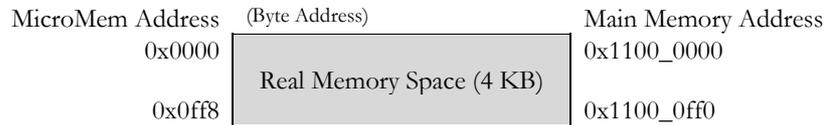
The VU1 registers, which are mapped to the address 0x4000 and after, are not mapped to main memory.

VU Mem Address	(Byte Address)	Main Memory Address
0x0000	Real Memory Space (4 KB)	0x1100_4000
0x0fff		0x1100_4fff
0x1000		0x1100_5000
0x1fff		0x1100_5fff
0x2000	Mirrored Real Memory Space	0x1100_6000
0x2fff		0x1100_6fff
0x3000	Mirrored Real Memory Space	0x1100_7000
0x3fff		0x1100_7fff
0x4000		N/A
0x41f0	32 128-bit VU1 VF Registers	N/A
0x4200	16 128-bit VU1 VI Registers (Lower 16 bits are effective.)	N/A
0x42f0		N/A
0x4300	VU1 Status Flag	N/A
0x4310	VU1 MAC Flag	N/A
0x4320	VU1 Clipping Flag	N/A
0x4340	VU1 R Register	N/A
0x4350	VU1 I Register	N/A
0x4360	VU1 Q Register	N/A
0x4370	VU1 P Register	N/A
0x43a0	VU1 TPC	N/A

The VU Mem is specified in qword alignment, and effective data addresses are divisible by 16 without exception. Therefore, the address divided by 16 is used in the instruction for the VIF.

### 6.2.2. MicroMem0 Memory Map

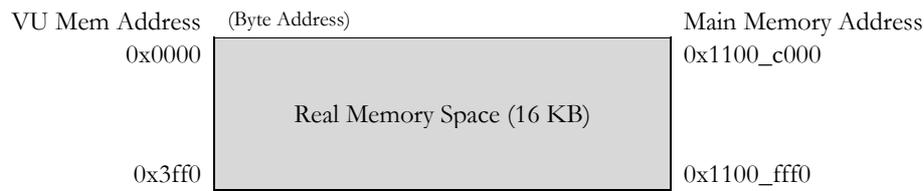
MicroMem0 is 4 Kbytes (512 dwords) in size and is mapped as follows. Although the MicroMem0 is also mapped to main memory as shown in the right-hand part of the figure, it is accessible from the EE Core only when VU0 and VIF0 are stopped.



The MicroMem is specified in 64-bit alignment, and effective addresses must be divisible by 8. Therefore, the address divided by 8 is used in the instruction for the VIF.

### 6.2.3. VU Mem1 Memory Map

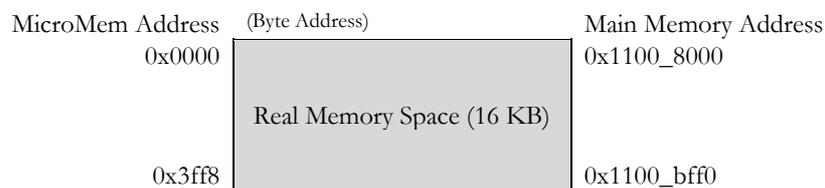
VU Mem1 is 16K bytes (1K qwords) in size and is mapped as follows. VUMem1 is accessible from the EE Core only when VU1, VIF1, and GIF are stopped.



The VU Mem is specified in qword alignment, and effective data addresses must be divisible by 16. Therefore, the address divided by 16 is used in the instruction for the VIF.

### 6.2.4. MicroMem1 Memory Map

MicroMem1 is 16 Kbytes (2048 dwords) in size and is mapped as follows. Although the MicroMem1 is also mapped to main memory as shown in the right-hand part of the figure, it is accessible from the EE Core only when VU1, VIF1, and GIF are stopped.



The MicroMem is specified in 64-bit alignment, and effective addresses must be divisible by 8. Therefore, the address divided by 8 is used in the instruction for the VIF.

## 6.3. Data Transfer by VIF

### 6.3.1. VIF Packet

Data transferred to the VIF is a DMA packet with 128-bit alignment as shown in the following figures. According to the 32-bit VIFcode in the transferred data, the VIF decompresses the following data and writes the memory and register in the VU. However, the DMAtag may be or may not be transferred with the data, depending on the DMA transfer mode. This causes a difference in the contents of the DMA packet. The VIFcode and the following data string are called the VIF packet. Two or more VIF packets exist in the DMA packet, as shown in the figures below.

#### DMA packet example (When DMAtag is transferred)

←MSB	128 bits			LSB→
data	VIFcode0	DMAtag		
data	data	VIFcode2	VIFcode1	
data	data	data	data	
data	data	data	data	
data	data	data	data	
data	data	data	VIFcode3	
data	data	VIFcode4	data	
data	data	data	data	
VIFcode5	data	data	data	

#### DMA packet example (When DMAtag is not transferred)

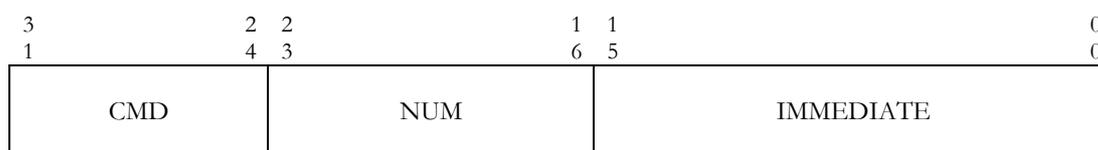
←MSB	128 bits			LSB→
--	--	DMAtag		
VIFcode2	VIFcode1	data	VIFcode0	
data	data	data	data	
data	data	data	data	
data	data	data	data	
data	VIFcode3	data	data	
VIFcode4	data	data	data	
data	data	data	data	
data	data	data	data	
--	--	VIFcode5	data	

#### VIF packets included in the above DMA packets

data	VIFcode0		
VIFcode1			
data	data	.....data × 12	VIFcode2
data	data	.....data × 2	VIFcode3
data	data	.....data × 3	VIFcode4
VIFcode5			

### 6.3.2. VIFcode Structure

The VIFcode is 32 bits in length, consisting of a CMD field (8 bits), NUM field (8 bits), and IMMEDIATE field (16 bits), as shown in the figure below. In the normal data decompression transfer instruction (VIFcode UNPACK), the data type, the number of data vectors decompressed, and the VU Mem address of the decompression destination are shown in the lower 4 bits of the CMD field, the NUM field, and the IMMEDIATE field respectively.



#### CMD Field

The CMD field directs the operation of the VIF, that is, the decompression method of the following data. The list is shown below. For details, refer to "6.4. VIFcode Reference".

Bit	Name	Function	VIF packet length
i0000000	NOP	No operation	1
i0000001	STCYCL	Sets CYCLE register value	1
i0000010	OFFSET	Sets OFFSET register value (VIF1 only)	1
i0000011	BASE	Sets BASE register value (VIF1 only)	1
i0000100	ITOP	Sets ITOPS register value	1
i0000101	STMOD	Sets MODE register value	1
i0000110	MSKPATH3	Masks GIF PATH3 transfer (VIF1 only)	1
i0000111	MARK	Sets MARK register value	1
i0010000	FLUSHE	Waits for end of micro program	1
i0010001	FLUSH	Waits for end of micro program and end of GIF (PATH1 /PATH2) transfer (VIF1 only)	1
i0010011	FLUSHA	Waits for end of micro program and end of GIF transfer (VIF1 only)	1
i0010100	MSCAL	Activates micro programs	1
i0010111	MSCNT	Executes micro program continuously	1
i0010101	MSCALF	Activates micro programs (VIF1 only)	1
i0100000	STMASK	Sets value to MASK register	1+1
i0110000	STROW	Sets value to Row register	1+4
i0110001	STCOL	Sets value to Col register	1+4
i1001010	MPG	Loads micro program	1+NUM x 2
i1010000	DIRECT	Transfers data to GIF (via PATH2) (VIF1 only)	1+IMMEDIATE x 4
i1010001	DIRECTHL	Transfers data to GIF (via PATH2) (VIF1 only)	1+IMMEDIATE x 4
i11mvnvl	UNPACK	Decompresses data and writes to VU Mem	(Described later)

The most significant bit of the CMD field is the interrupt control bit (i). When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding VIFcode MARK) stalls.

**NUM Field**

The NUM field shows the amount of data written to the VU Mem or MicroMem (Data is in 128-bit units and microinstruction code is in 64-bit units). When NUM=0, it is considered to be 256.

Note that it is not the amount of data in the VIF packet. NUM pieces of data can be written even when the amount of data in the packet is 0 depending on the CYCLE register setting. For details, refer to "6.3.5. Skipping Write and Filling Write".

**IMMEDIATE Field**

The contents of the IMMEDIATE field vary according to the CMD field. For details, refer to "6.4. VIFcode Reference".

**6.3.3. Limitation of Alignment**

The microinstruction code that follows the VIFcode MPG is 64 bits long, and needs to be specified in 64-bit alignment. Therefore, it is necessary to insert the VIFcode NOP properly so that the VIFcode MPG is allocated in the following two patterns.

**MPG Allocation Pattern (1)**

127	96	95	64	63	32	31	0
VIFcode MPG		-	-		-		
CODE2				CODE1			

(The remaining data patterns are omitted.)

**MPG Allocation Pattern (2)**

127	96	95	64	63	32	31	0
CODE1				VIFcode MPG		-	
CODE3				CODE2			

(The remaining data patterns are omitted.)

The data that follows the VIFcode DIRECT/DIRECTHL is 128 bits long, and needs to be specified in 128-bit alignment. Therefore, it is necessary to insert the VIFcode NOP properly so that the VIFcode DIRECT/DIRECTHL is allocated as follows.

**DIRECT/DIRECTHL Allocation Pattern**

127	96	95	64	63	32	31	0
VIFcode DIRECT/DIRECTHL		-	-		-		
DATA1							
DATA2							

(The remaining data patterns are omitted.)

### 6.3.4. Data Write by UNPACK

The most general data transfer is the data transfer to the VU Mem, which uses the VIFcode UNPACK. As shown by the name of UNPACK, the transfer data following the VIFcode is decompressed to the vector data of 32 bits x 4 elements before written to the VU Mem. (Data is written after additional processing such as blocking, supplementation, mask, or addition. They are described later in this document.)

The decompression format, or the format of transfer data following the VIFcode, can be specified from the 11 formats below by the lower 4 bits (vn, vl) of the CMD field.

CMD	vn	vl	Format	No. of elements (Dimension)	Data length	Alignment
i11m0000	00	00	S-32	1	32 bits	32 bits
i11m0001	00	01	S-16	1	16 bits	32 bits
i11m0010	00	10	S-8	1	8 bits	32 bits
i11m0100	01	00	V2-32	2	32 bits	32 bits
i11m0101	01	01	V2-16	2	16 bits	32 bits
i11m0110	01	10	V2-8	2	8 bits	32 bits
i11m1000	10	00	V3-32	3	32 bits	32 bits
i11m1001	10	01	V3-16	3	16 bits	32 bits
i11m1010	10	10	V3-8	3	8 bits	32 bits
i11m1100	11	00	V4-32	4	32 bits	32 bits
i11m1101	11	01	V4-16	4	16 bits	32 bits
i11m1110	11	10	V4-8	4	8 bits	32 bits
i11m1111	11	11	V4-5	4	5+5+5+1 bits	32 bits

\* The i bit shows the presence of the interrupt after processing is performed. See "6.3.2. VIFcode Structure".

\* The m bit shows the presence of the supplementation and mask processing. See "6.3.6. Write Data Mask".

In 8-bit and 16-bit data decompression, the USN bit in the IMMEDIATE field specifies whether the compression is signed (sign-extended) or unsigned (0-padded). Signed decompression is performed when USN=0, and unsigned decompression is performed when USN=1.

A fraction less than 32 bits may be generated occasionally depending on the format and amount of data. In that case, padding to the end is required so that the whole data length can be a multiple of 32 bits.

Decompression processing in each format is shown in the following tables, which are examples of generation of vector data of 128 bits x 3.

#### S-32 Format

Transfer data

32 bits	32 bits	32 bits	32 bits
S3	S2	S1	VIFcode

Decompression data

W (32 bits)	Z (32 bits)	Y (32 bits)	X (32 bits)
S1	S1	S1	S1
S2	S2	S2	S2
S3	S3	S3	S3

**S-16 Format**

Transfer data

16 bits	16 bits	16 bits	16 bits	32 bits
pad	S3	S2	S1	VIFcode

Decompression data

W (32 bits)		Z (32 bits)		Y (32 bits)		X (32 bits)	
Extended ←	S1						
Extended ←	S2						
Extended ←	S3						

**S-8 Format**

Transfer data

8bits	8bits	8bits	8bits	32 bits
pad	S3	S2	S1	VIFcode

Decompression data

W (32 bits)		Z (32 bits)		Y (32 bits)		X (32 bits)	
Extended ←	S1						
Extended ←	S2						
Extended ←	S3						

**V2-32 Format**

Transfer data

32 bits	32 bits	32 bits	32 bits
X2	Y1	X1	VIFcode
	Y3	X3	Y2

Decompression data

W (32 bits)		Z (32 bits)		Y (32 bits)		X (32 bits)	
Indeterminate	Indeterminate	Indeterminate	Indeterminate	Y1	X1	Indeterminate	Indeterminate
Indeterminate	Indeterminate	Indeterminate	Indeterminate	Y2	X2	Indeterminate	Indeterminate
Indeterminate	Indeterminate	Indeterminate	Indeterminate	Y3	X3	Indeterminate	Indeterminate

**V2-16 Format**

Transfer data

16 bits	32 bits					
Y3	X3	Y2	X2	Y1	X1	VIFcode

Decompression data

W (32 bits)	Z (32 bits)	Y (32 bits)	X (32 bits)
Indeterminate	Indeterminate	Extended ←	Y1
Indeterminate	Indeterminate	Extended ←	Y2
Indeterminate	Indeterminate	Extended ←	Y3
		Extended ←	X1
		Extended ←	X2
		Extended ←	X3

**V2-8 Format**

Transfer data

16 bits	8bits	8bits	8bits	8bits	8bits	8bits	32 bits
pad	Y3	X3	Y2	X2	Y1	X1	VIFcode

Decompression data

W (32 bits)	Z (32 bits)	Y (32 bits)	X (32 bits)
Indeterminate	Indeterminate	Extended ←	Y1
Indeterminate	Indeterminate	Extended ←	Y2
Indeterminate	Indeterminate	Extended ←	Y3
		Extended ←	X1
		Extended ←	X2
		Extended ←	X3

**V3-32 Format**

Transfer data

32 bits	32 bits	32 bits	32 bits
Z1	Y1	X1	VIFcode
X3	Z2	Y2	X2
		Z3	Y3

Decompression data

W (32 bits)	Z (32 bits)	Y (32 bits)	X (32 bits)
Indeterminate	Z1	Y1	X1
Indeterminate	Z2	Y2	X2
Indeterminate	Z3	Y3	X3

**V3-16 Format**

Transfer data

16 bits	32 bits						
Z2	Y2	X2	Z1	Y1	X1	VIFcode	
					pad	Z3	Y3 X3

Decompression data

W (32 bits)		Z (32 bits)		Y (32 bits)		X (32 bits)	
Indeterminate	Extended ←	Z1	Extended ←	Y1	Extended ←	X1	
Indeterminate	Extended ←	Z2	Extended ←	Y2	Extended ←	X2	
Indeterminate	Extended ←	Z3	Extended ←	Y3	Extended ←	X3	

**V3-8 Format**

Transfer data

	8bits	32 bits										
pad	Z3	Y3	X3	Z2	Y2	X2	Z1	Y1	X1	VIFcode		

Decompression data

W (32 bits)		Z (32 bits)		Y (32 bits)		X (32 bits)	
Indeterminate	Extended ←	Z1	Extended ←	Y1	Extended ←	X1	
Indeterminate	Extended ←	Z2	Extended ←	Y2	Extended ←	X2	
Indeterminate	Extended ←	Z3	Extended ←	Y3	Extended ←	X3	

**V4-32 Format**

Transfer data

32 bits	32 bits	32 bits	32 bits
Z1	Y1	X1	VIFcode
Z2	Y2	X2	W1
Z3	Y3	X3	W2
			W3

Decompression data

W (32 bits)		Z (32 bits)		Y (32 bits)		X (32 bits)	
W1	Z1	Y1	X1				
W2	Z2	Y2	X2				
W3	Z3	Y3	X3				

**V4-16 Format**

Transfer data

16 bits	32 bits						
	X2	W1	Z1	Y1	X1	VIFcode	
		W3	Z3	Y3	X3	W2	Z2

Decompression data

W (32 bits)		Z (32 bits)		Y (32 bits)		X (32 bits)	
Extended ←	W1	Extended ←	Z1	Extended ←	Y1	Extended ←	X1
Extended ←	W2	Extended ←	Z2	Extended ←	Y2	Extended ←	X2
Extended ←	W3	Extended ←	Z3	Extended ←	Y3	Extended ←	X3

**V4-8 Format**

Transfer data

8bits	32 bits													
W3	Z3	Y3	X3	W2	Z2	Y2	X2	W1	Z1	Y1	X1	VIFcode		

Decompression data

W (32 bits)		Z (32 bits)		Y (32 bits)		X (32 bits)	
Extended ←	W1	Extended ←	Z1	Extended ←	Y1	Extended ←	X1
Extended ←	W2	Extended ←	Z2	Extended ←	Y2	Extended ←	X2
Extended ←	W3	Extended ←	Z3	Extended ←	Y3	Extended ←	X3

**V4-5 Format**

Transfer data

16 bits	16 bits	16 bits	16 bits	32 bits	
pad	RGBA3	RGBA2	RGBA1	VIFcode	

However, the format of RGBAn is as follows.

16 bits							
1	1	1	0	0	0	0	0
5	4	0	9	5	4	0	0
A	B	G	R				

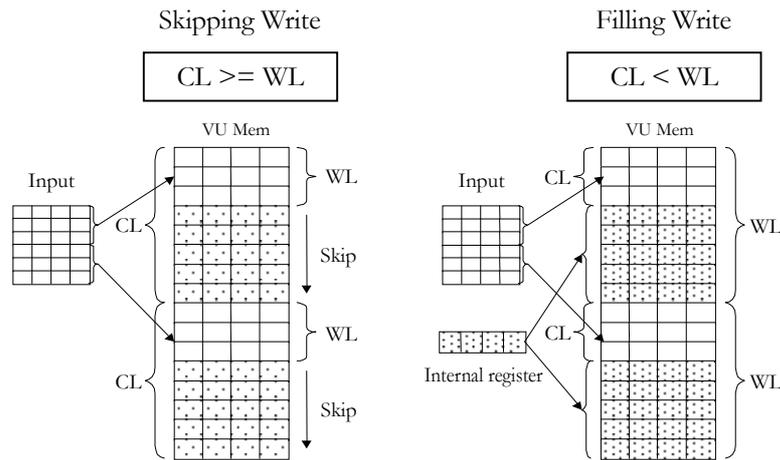
Decompression data

W (32 bits)			Z (32 bits)			Y (32 bits)			X (32 bits)		
0	A1	0000000	0	B1	000	0	G1	000	0	R1	000
0	A2	0000000	0	B2	000	0	G2	000	0	R2	000
0	A3	0000000	0	B3	000	0	G3	000	0	R3	000

### 6.3.5. Skipping Write and Filling Write

Since the VU has a narrow work area and a low degree of flexibility in programming, the type of data transferred to the VU Mem and the data arrangement in the memory should be considered for the convenience of the VU as much as possible. Therefore, the VIF has the ability to appropriately arrange the decompressed data in the VU Mem. The CYCLE register (VIFn\_CYCLE) controls it.

The WL and CL fields in the CYCLE register set the block size and cycle to write data to the VU Mem. The figure below shows the relation between them.



When  $CL \geq WL$ , data is not written in the space between WL and CL ("Skipping write"). The writing cycle of WL is executed every CL interval, so the amount of data and the VU Mem address of the decompression destination are as follows:

Amount of read data	num
Amount of write data	num
Decompression destination address	if (FLG == 0) $ADDR + CL \times (num/WL) + (num \% WL)$ if (FLG == 1) $VIFn\_TOPS + ADDR + CL \times (num/WL) + (num \% WL)$

(/ shows integer division and % shows surplus.)

On the other hand, when  $CL < WL$ , the space is filled with the value of an internal register according to specifications of the MASK register (VIFn\_MASK) ("Filling write"). The write cycle of WL is performed at WL intervals, and the data read is suspended after CL and restarted after WL. While the read is suspended, the supplemental data is written according to the specification of the MASK register. The amount of data and the VU Mem address of the decompression destination are as follows:

Amount of read data	$CL \times (num/WL) + \text{limit}(num \% WL, CL)$ $\text{int limit}(\text{int } a, \text{int } \text{max}) \{ \text{return}( a > \text{max} ? \text{max} : a); \}$
Amount of write data	num
Decompression destination address	if (FLG == 0) ADDR + num if (FLG == 1) VIF1_TOPS + ADDR + num

(/ shows integer division and % shows surplus.)

### 6.3.6. Write Data Mask

When writing the decompressed data to the VU Mem, it is possible to mask the data partially or write the internal register data instead. The MASK register specifies this. The MASK register has 16 2-bit mask specifiers as follows.

								1	1						0
								6	5						0
m15	m14	m13	m12	m11	m10	m9	m8	m7	m6	m5	m4	m3	m2	m1	m0

These specifiers are used as shown in the following matrix. When the m bit (bit 28) of the CMD field is 1, the masking of the row corresponding to the write cycle is selected. Then, the writing of each field of the 128-bit vector data is controlled.

Write cycle = 1	m3	m2	m1	m0
Write cycle = 2	m7	m6	m5	m4
Write cycle = 3	m11	m10	m9	m8
Write cycle >= 4	m15	m14	m13	m12
	↓	↓	↓	↓
	W Field	Z Field	Y Field	X Field
	bits 127-96	bits 95-64	bits 63-32	bits 31-0

The write cycle is controlled by the WL and CL fields of the CYCLE register. For details, refer to "6.3.5. Skipping Write and Filling Write".

Writing is processed according to the value of mask specifier m[n] as follows: When the m bit of the CMD field is 0, the result is the same as when all the mask specifiers are 0.

**m[n] = 0:**

The decompressed input data is written as is.

S-32/S-16/S-8

	W Field bits 127-96	Z Field bits 95-64	Y Field bits 63-32	X Field bits 31-0
Input data	S	S	S	S
Mask specifier	0	0	0	0
Write data	S	S	S	S

V2-32/V2-16/V2-8

	W Field bits 127-96	Z Field bits 95-64	Y Field bits 63-32	X Field bits 31-0
Input data	Indeterminate	Indeterminate	Y	X
Mask specifier	0	0	0	0
Write data	Indeterminate	Indeterminate	Y	X

V3-32/V3-16/V3-8

	W Field bits 127-96	Z Field bits 95-64	Y Field bits 63-32	X Field bits 31-0
Input data	Indeterminate	Z	Y	X
Mask specifier	0	0	0	0
Write data	Indeterminate	Z	Y	X

V4-32/V4-16/V4-8

	W Field bits 127-96	Z Field bits 95-64	Y Field bits 63-32	X Field bits 31-0
Input data	W	Z	Y	X
Mask specifier	0	0	0	0
Write data	W	Z	Y	X

**m[n] = 1:**

Instead of the decompressed input data, the value of the corresponding Row register is written.

V4-32/V4-16/V4-8

	W Field bits 127-96	Z Field bits 95-64	Y Field bits 63-32	X Field bits 31-0
Input data	W	Z	Y	X
Mask specifier	1	1	1	1
Write data	VIF <sub>n</sub> _R3	VIF <sub>n</sub> _R2	VIF <sub>n</sub> _R1	VIF <sub>n</sub> _R0

Ditto for other formats.

**m[n] = 2:**

The value of the Col register corresponding to the write cycle is output.

V4-32/V4-16/V4-8

	W Field bits 127-96	Z Field bits 95-64	Y Field bits 63-32	X Field bits 31-0
Input data	W	Z	Y	X
Mask specifier	2	2	2	2
Write data	VIF <sub>n</sub> _C0	VIF <sub>n</sub> _C0	VIF <sub>n</sub> _C0	VIF <sub>n</sub> _C0
Write data	VIF <sub>n</sub> _C1	VIF <sub>n</sub> _C1	VIF <sub>n</sub> _C1	VIF <sub>n</sub> _C1
Write data	VIF <sub>n</sub> _C2	VIF <sub>n</sub> _C2	VIF <sub>n</sub> _C2	VIF <sub>n</sub> _C2
Write data	VIF <sub>n</sub> _C3	VIF <sub>n</sub> _C3	VIF <sub>n</sub> _C3	VIF <sub>n</sub> _C3

Cycle = 1  
 Cycle = 2  
 Cycle = 3  
 Cycle >= 4

Ditto for other formats.

**m[n] = 3:**

Write is masked.

### 6.3.7. Addition Decompression Write

By the setting of the MODE register (VIFn\_MODE), it is possible to perform addition decompression processing in which the Row register value is added to the decompressed input data and written to the VU Mem, except when the input data is in V4-5 format. The offset mode, in which a constant value is added to a series of input data, and the difference mode, in which a series of input data is totaled, are available as follows.

Mode	Processing
00	No addition processing (normal)
01	Offset mode (Input data + Row -> VU Mem)
02	Difference mode (Input data + Row -> Row -> VU Mem)

Processing to add the Row register value is performed only when the m bit of the CMD field is 0 or the mask specifier of the corresponding field is 0, that is, when input data is written as is.

The Row register update in the difference mode is performed in the same way. Since the input data becomes indeterminate in the W and Z fields in the V2-32/V2-16/V2-8 format and in the W field in the V3-32/V3-16/V3-8 format, it is necessary to prevent the Row register update by setting the corresponding mask specifiers to a value other than 0.

#### Example 1: V4-32 Format, Offset Mode, without Filling-Mask

Input data	W	Z	Y	X
Mask specifier	0	0	0	0
	VIFn_R3	VIFn_R2	VIFn_R1	VIFn_R0
Row register	R3	R2	R1	R0
	↓	↓	↓	↓
Write data	W+R3	Z+R2	Y+R1	X+R0
	VIFn_R3	VIFn_R2	VIFn_R1	VIFn_R0
Row register	R3	R2	R1	R0

#### Example 2: V4-32 Format, Difference Mode, without Filling-Mask

Input data	W	Z	Y	X
Mask specifier	0	0	0	0
	VIFn_R3	VIFn_R2	VIFn_R1	VIFn_R0
Row register	R3	R2	R1	R0
	↓	↓	↓	↓
Write data	W+R3	Z+R2	Y+R1	X+R0
	VIFn_R3	VIFn_R2	VIFn_R1	VIFn_R0
Row register	W+R3	Z+R2	Y+R1	X+R0

**Example 3: V4-32 Format, Difference Mode, Filling with Col Register**

Input data	W	Z	Y	X
Mask specifier	2	0	0	0
	VIFn_R3	VIFn_R2	VIFn_R1	VIFn_R0
Row register	R3	R2	R1	R0
	↓	↓	↓	↓
Write data	VIFn_Cx	Z+R2	Y+R1	X+R0
	VIFn_R3	VIFn_R2	VIFn_R1	VIFn_R0
Row register	R3	Z+R2	Y+R1	X+R0

**Example 4: V2-32 Format, Difference Mode, Filling with Col Register**

Input data	Indeterminate	Indeterminate	Y	X
Mask specifier	2	2	0	0
	VIFn_R3	VIFn_R2	VIFn_R1	VIFn_R0
Row register	R3	R2	R1	R0
	↓	↓	↓	↓
Write data	VIFn_Cx	VIFn_Cx	Y+R1	X+R0
	VIFn_R3	VIFn_R2	VIFn_R1	VIFn_R0
Row register	R3	R2	Y+R1	X+R0

### 6.3.8. Double Buffering

In VPU1, it is possible to perform double buffering, by which the VIF writes data without destroying the data in use by the VU, by setting two buffer areas in the VU Mem. In this way, data transfer to the VU Mem and processing by the micro program can be executed concurrently.

Double buffering is outlined as follows: The BASE/OFFSET/TOPS shows the VIF1\_BASE/VIF1\_OFST/VIF1\_TOPS register respectively. Moreover, the DBF flag shows the DBF flag of the VIF1\_STAT register.

#### 0a) Preparation for Double Buffers

First, the BASE is set by the VIFcode BASE. This becomes the buffer address of one buffer. Next, the OFFSET is set by the VIFcode OFFSET. The BASE+OFFSET becomes the address of the other buffer. At the same time, the BASE value is transferred to the TOPS and the DBF flag is set to 0.

#### 0b) Micro Program Transfer

The microprogram is transferred by means of the VIFcode MPG. This microprogram should be programmed to obtain the address of the data transferred from the VIF in every transfer by the XTOP instruction.

#### 1) Data Transfer

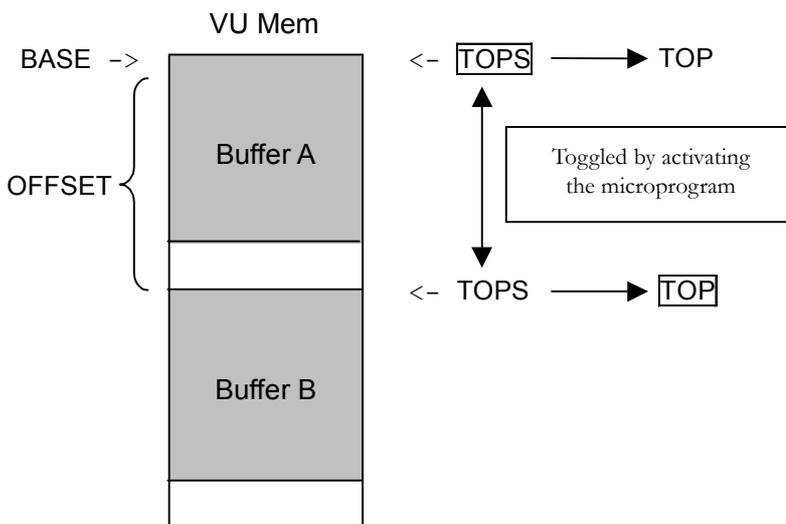
Data is transferred to the VU Mem by the VIFcode UNPACK. It is important to decompress the data to the TOPS-based address by setting the FLG bit (MSB of the IMMEDIATE field) to 1.

#### 2) Activation of Micro Program

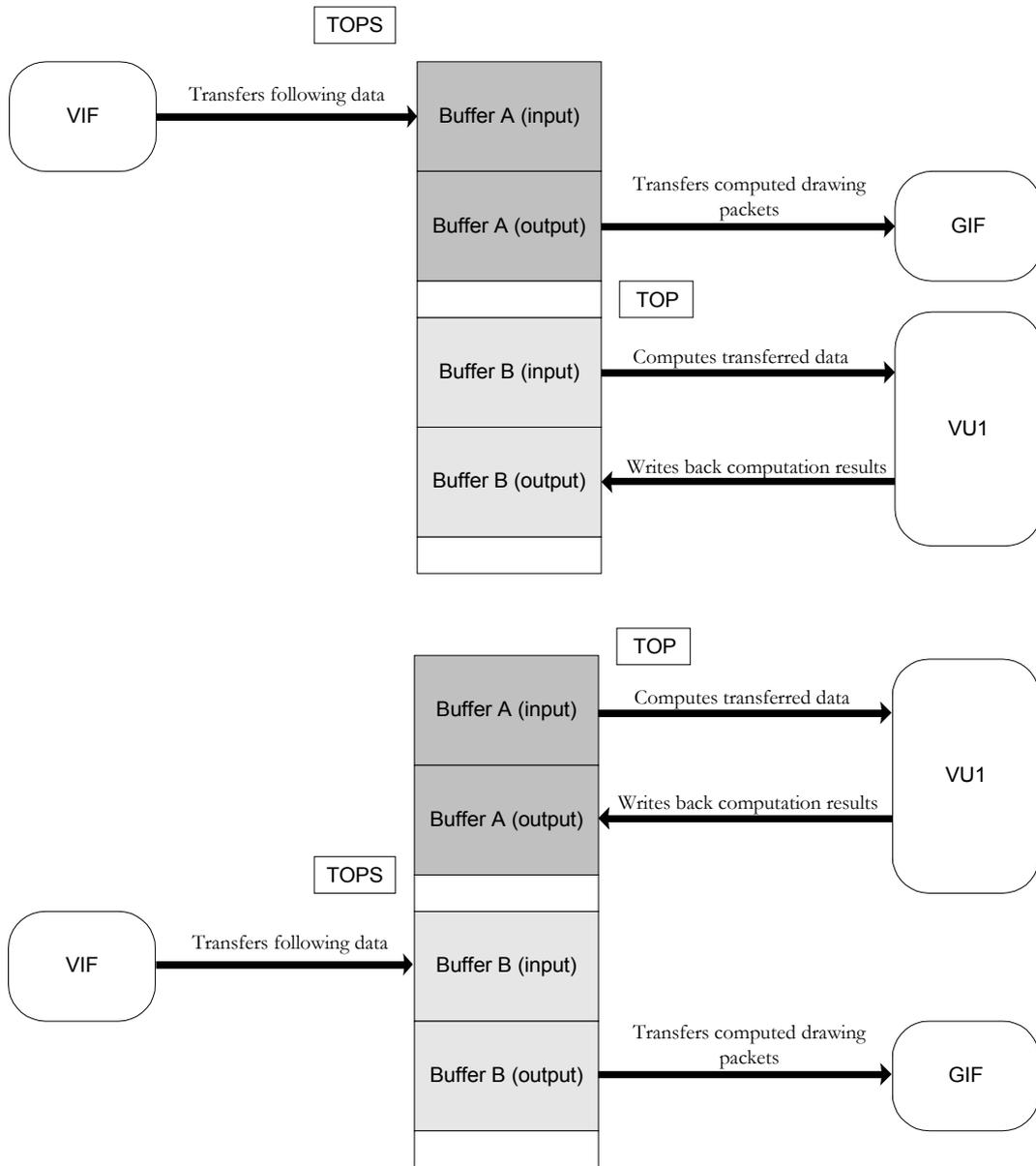
The microprogram previously transferred by the VIFcode MSCAL/MSCALF is activated. At this time, the value of the TOPS register is transferred to the TOP register, the other buffer address is set to the TOPS register in preparation for the following data transfer, and the DBF flag is reversed.

By repeating 1) and 2) above, the already transferred data can be processed by the micro program while alternately transferring the data to the two buffers.

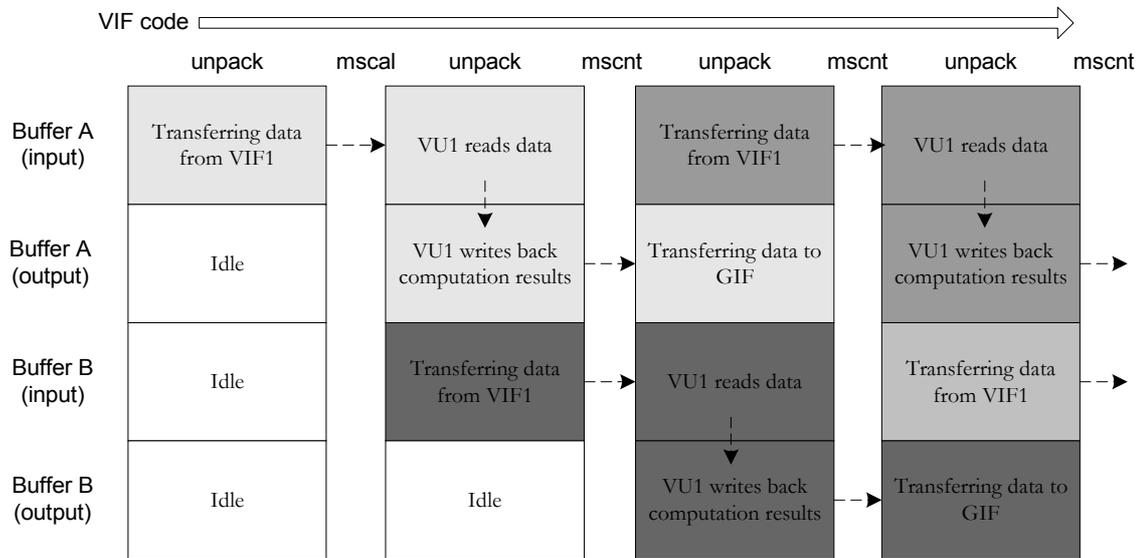
Double buffer structure is as follows:



The actual use status of a double buffer is as follows:



The VU Mem1 status while double buffering is as follows:



## 6.4. VIFcode Reference

The following is the description of the required parameters and the VIF operation for each VIFcode defined in the CMD field.

### Legends

3	2 2	1 1	0
1	4 3	6 5	0
CMD —	NUM —	IMMEDIATE —	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
CMD field bit pattern	VIF operation	Effective VIF	Operation on the i bit interrupt	Subpacket length including the VIFcode (words)

**NOP : No operation**

3 1	2 2 4 3	1 1 6 5	0 0
CMD i0000000	NUM -	IMMEDIATE -	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0000000	No operation	VIF0/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

**NUM Field**

Arbitrary

**IMMEDIATE Field**

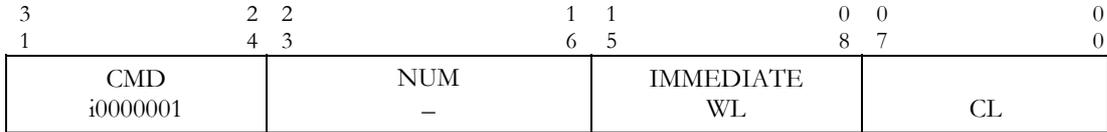
Arbitrary

**Operation**

No operation is performed.

NOP is used to adjust the data alignment in the VIF packet.

## STCYCL : Sets write recycle



CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0000001	Sets CYCLE register value.	VIF0/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

Bits 15-8: Value to be written to the WL Field

Bits 7-0: Value to be written to the CL Field

### Operation

STCYCL writes the value of the IMMEDIATE field to the VIFn\_CYCLE register.

For the CYCLE register function, refer to "6.3.5. Skipping Write and Filling Write".

**OFFSET : Sets the double buffer offset**

3 1	2 2 4 3	1 1 6 5	1 0 0 9	0 0
CMD i0000010	NUM -	-	OFFSET	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0000010	Sets OFFSET register value.	-/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and processing of the following VIFcode (excluding the VIFcode MARK) stalls.

**NUM Field**

Arbitrary

**IMMEDIATE Field**

Bits 9-0: Value to be written to the VIF1\_OFST register

**Operation**

OFFSET writes the lower 10 bits of the IMMEDIATE field to the VIF1\_OFST register.

At the same time, the DBF flag of the VIF1\_STAT register is cleared to 0, and the VIF1\_BASE register value is set to the VIF1\_TOPS. That is, the pointer for the double buffer points to the BASE.

Refer to "6.3.8. Double Buffering".

**BASE : Sets the base address of the double buffer**

3 1	2 2 4 3	1 1 6 5	1 0 0 9	0 0
CMD i0000011	NUM -	-	BASE	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0000011	Sets BASE register value.	-/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

**NUM Field**

Arbitrary

**IMMEDIATE Field**

Bits 9-0: Value to be written to the VIF1\_BASE register

**Operation**

BASE writes the lower 10 bits of the IMMEDIATE field to the VIF1\_BASE register.

These bits become the base address of the double buffers.

Refer to "6.3.8. Double Buffering".

## ITOP : Sets the data pointer

3 1	2 2 4 3	1 1 6 5	1 0 0 9	0 0
CMD i0000100	NUM -	-	ADDR	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0000100	Sets ITOPS register value.	VIF0/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

Bits 9-0: Value to be written to the VIFn\_ITOPS register

### Operation

ITOP writes the lower 10 bits of the IMMEDIATE field to the VIFn\_ITOPS register.

This value is read from the VU by the XITOP instruction. For details, refer to " VIF0\_ITOP / VIF1\_ITOP " and " VIF0\_ITOPS / VIF1\_ITOPS ".

## STMOD : Sets the addition decompression mode

3 1	2 2 4 3	1 1 6 5	0 0 0 2 1 0
CMD i0000101	NUM -	-	M O D E

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0000101	Sets MODE register value	VIF0/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

Bits 1-0: Value to be written to the VIFn\_MODE register

### Operation

STMOD writes the lower 2 bits of the IMMEDIATE field to the VIFn\_MODE register.

This becomes the addition decompression mode setting. For details, refer to "6.3.7. Addition Decompression Write".

### MSKPATH3 : Sets the PATH3 mask

3 1	2 2 4 3	1 1 6 5	0 0
CMD i0000110	NUM -	M A S K	-

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0000110	Masks the GIF PATH3.	-/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

#### NUM Field

Arbitrary

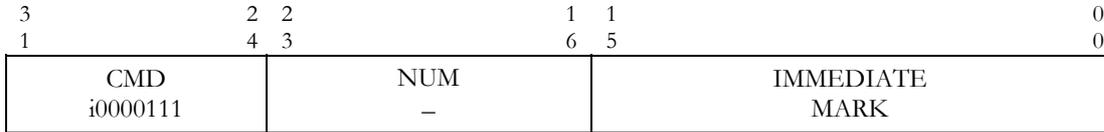
#### IMMEDIATE Field

Bits 15 : 1    Masked (Transfer disabled)  
 0            Transfer enabled

#### Operation

MSKPATH3 enables/disables transfer processing via the GIF PATH3. The setting of this register applies to the next data block or later.

## MARK : Sets the MARK value



CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0000111	Sets MARK register value	VIF0/VIF1	No stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

Arbitrary MARK value

### Operation

MARK writes the value of the IMMEDIATE field to the VIFn\_MARK register.

By properly setting the MARK value, it becomes possible to use this value for synchronization with the EE Core and debugging.

**FLUSHE : Waits for end of the microprogram**

3 1	2 2 4 3	1 1 6 5	0 0
CMD i0010000	NUM -	IMMEDIATE -	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0010000	Waits for end of micro program.	VIF0/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

**NUM Field**

Arbitrary

**IMMEDIATE Field**

Arbitrary

**Operation**

FLUSHE waits for the state in which the microprogram in VU0/VU1 has been ended.

## FLUSH : Waits for end of the microprogram

3 1	2 2 4 3	1 1 6 5	0 0
CMD i0010001	NUM -	IMMEDIATE -	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0010001	Waits for end of both micro program and GIF (PATH1 /PATH2) transfer	-/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

Arbitrary

### Operation

FLUSH waits for the state in which transfers to the GIF from PATH1 and PATH2 have been ended after end of microprogram in VU1.

In some cases, however, it may not wait for a transfer via PATH1 activated by an XGKICK instruction with the E bit set. If an XGKICK instruction with the E bit set is executed and the transfer is suspended when a transfer via PATH1 by the preceding XGKICK instruction is still in process, the waiting condition of the VIFcode FLUSH is cancelled at the completion of the first transfer via PATH1. As a result, FLUSH does not wait for the end of the transfer via PATH1 by the XGKICK instruction with the E bit.

**FLUSHA : Waits for end of the microprogram**

3 1	2 2 4 3	1 1 6 5	0 0
CMD i0010011	NUM -	IMMEDIATE -	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0010011	Waits for end of both micro program and GIF transfer	-/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

**NUM Field**

Arbitrary

**IMMEDIATE Field**

Arbitrary

**Operation**

FLUSHA waits for the state in which there is no transfer request from PATH3 after the end of micro program in VU1 and end of transfer to the GIF from PATH1 and PATH2.

## MSCAL : Activates the microprogram

3 1	2 2 4 3	1 1 6 5	0 0
CMD i0010100	NUM -	IMMEDIATE EXECADDR	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0010100	Activates micro program.	VIF0/VIF1	Stall	1

When  $i=1$ , the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

The starting address divided by 8

### Operation

MSCAL waits for the end of the microprogram under execution and activates the micro program with the value of the IMMEDIATE field as the starting address.

## MSCNT : Activates the microprogram

3 1	2 2 4 3	1 1 6 5	0 0
CMD i0010111	NUM -	IMMEDIATE -	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0010111	Continuously executes micro programs.	VIF0/VIF1	Stall	1

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

Arbitrary

### Operation

MSCNT waits for the end of the microprogram under execution and executes the next microprogram from the address following the most recently ended one in the PC (program counter).

## MSCALF : Activates the microprogram

3 1	2 2 4 3	1 1 6 5	0 0
CMD i0010101	NUM -	IMMEDIATE EXECADDR	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0010101	Activates microprograms	VIF0/VIF1	Stall	1

When  $i=1$ , the VIF generates an interrupt after the end of processing, and processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

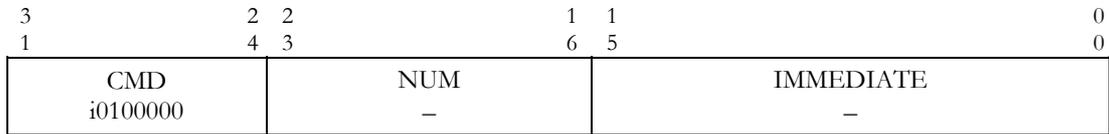
### IMMEDIATE Field

The starting address divided by 8

### Operation

MSCALF waits for the end of both the microprogram and the GIF(PATH1/PATH2) transfer and executes the microprogram with the value of the IMMEDIATE field as the starting address.

## STMASK : Sets the data mask pattern



CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0100000	Sets MASK register value.	VIF0/VIF1	Stall	1+1

When  $i=1$ , the VIF generates an interrupt after the end of processing, and processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

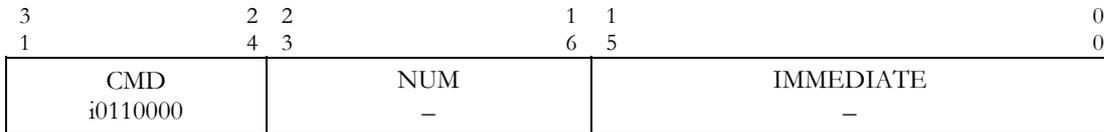
Arbitrary

### Operation

STMASK stores the next 1 word of data in the VIF<sub>n</sub>\_MASK register.

This data becomes the mask pattern of the write data. For details, refer to "6.3.6. Write Data Mask".

## STROW : Sets the filling data



CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0110000	Sets Row register value.	VIF0/VIF1	Stall	1+4

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

Arbitrary

### Operation

STROW stores the following 4 words of data in the VIF<sub>n</sub>\_R0 - VIF<sub>n</sub>\_R3 registers.

These are used as filling data when decompressed by the VIFcode UNPACK. For details, refer to "6.3.5. Skipping Write and Filling Write".

## STCOL : Sets the filling data

3 1	2 2 4 3	1 1 6 5	0 0
CMD i0110001	NUM -	IMMEDIATE -	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i0110001	Sets Col register value.	VIF0/VIF1	Stall	1+4

When  $i=1$ , the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

Arbitrary

### Operation

STCOL stores the following 4 words of data in the VIF<sub>n</sub>\_C0 - VIF<sub>n</sub>\_C3 registers.

These are used as filling data when decompressed by the VIFcode UNPACK. For details, refer to "6.3.5. Skipping Write and Filling Write".

## MPG : Transfers the microprogram

3 1	2 2 4 3	1 1 6 5	0 0
CMD i1001010	NUM SIZE	IMMEDIATE LOADADDR	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i1001010	Loads microprograms.	VIF0/VIF1	Stall	1+ NUM ×2

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Number of microprogram instructions (in 64-bit units)  
 NUM is considered to be 256 when 0 is specified.

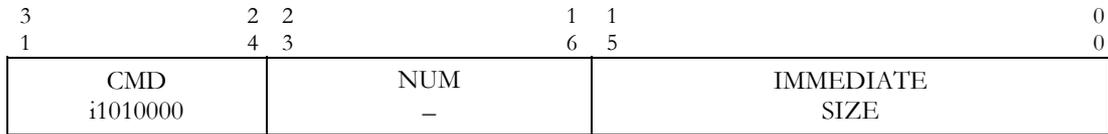
### IMMEDIATE Field

Load destination address divided by 8.

### Operation

MPG waits for the end of the microprogram and transfers the following NUM pieces of 64-bit data (microinstruction code) to the MicroMem address shown by the IMMEDIATE field.

## DIRECT : Transfers data to the GIF(GS)



CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i1010000	Transfers data to GIF (via PATH2).	-/VIF1	Stall	1+ IMMEDIATE ×4

When i=1, the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

Amount of transfer data (in 128-bit units)

IMMEDIATE is considered to be 65536 when 0 is specified.

### Operation

DIRECT transfers the following IMMEDIATE pieces of 128-bit data to the GS via GIF PATH2. It is necessary to put the appropriate GIFtag to the data.

## DIRECTHL : Transfers data to the GIF(GS)

3 1	2 2 4 3	1 1 6 5	0 0
CMD i1010001	NUM -	IMMEDIATE SIZE	

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i1010001	Transfers data to GIF (via PATH2)	-/VIF1	Stall	1+IMMEDIATE×4

When  $i=1$ , the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Arbitrary

### IMMEDIATE Field

Amount of transfer data (in 128-bit units)

IMMEDIATE is considered to be 65536 when 0 is specified.

### Operation

DIRECTHL transfers the following IMMEDIATE pieces of 128-bit data to the GS via GIF PATH2. The appropriate GIFtag must be attached to the data.

Interruption of PATH3 IMAGE mode data does not occur during the data transfer via PATH2 by DIRECTHL. Moreover, when the IMAGE mode data is being transferred via PATH3, DIRECTHL stalls until the end of the transfer.

## UNPACK : Transfers data to the VU Mem

3 1	2 2 4 3	1 1 1 6 5 4	0 9	0 0
CMD i11mavnvl	NUM SIZE	F L G	U S N	- ADDR

CMD	Function	VIF0/VIF1	Interrupt	Subpacket length
i11mavnvl	Decompresses data and writes to VU Mem.	VIF0/VIF1	Stall	(See below.)

When  $i=1$ , the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

### NUM Field

Amount of data written to the VU Mem (in 128-bit units)

### IMMEDIATE Field

Name	Pos.	Contents
ADDR	9:0	VU Mem address in transfer destination (address divided by 16.)
USN	14	Sign of 8-bit/16-bit data 1 Unsigned (Decompresses by padding 0 to the upper field.) 0 Signed (Decompresses by sign-extension.)
FLG	15	Address mode (VIF1 only) 1 Adds VIF1_TOPS register to ADDR. 0 Does not use VIF1_TOPS register.

When the FLG bit is 1 in VPU1, the address obtained by adding the VIF1\_TOPS register value to the ADDR field value becomes the transfer destination.

### Operation

UNPACK decompresses the following data in the format shown in the lower 4 bits (vn, vl) of the CMD field and transfers it to the VU Mem. The transfer destination address is the IMMEDIATE field value in VPU0, and is the value obtained by adding the VIF1\_TOPS register and IMMEDIATE field values according to the specification by the FLG bit in VPU1.

For the vn/vl specification and data decompression format, refer to "6.3.4. Data Write by UNPACK".

When  $i=1$ , the VIF generates an interrupt after the end of processing, and the processing of the following VIFcode (excluding the VIFcode MARK) stalls.

The length of the VIF packet is obtained by the following formula. The operator // means the division in which the quotient is rounded up when there is a remainder.

$$WL \leq CL: \quad 1 + (((32 \gg vl) \times (vn + 1)) \times num // 32)$$

$$WL > CL: \quad 1 + (((32 \gg vl) \times (vn + 1)) \times n // 32)$$

$$\text{However, } n = CL \times (num / WL) + \text{limit}(num \% WL, CL)$$

$$\text{int limit(int a, int max)}$$

$$\{ \text{return}(a > \text{max} ? \text{max} : a); \}$$

## 6.5. VPU-Related Register Reference

### VIF Register

The following VIF registers are mapped also to the address space of the EE Core.

When accessing from the EE Core, these registers are read-only, except for the MARK register. All of them are accessible only when the VIF is stopped.

Symbol	r/w	Width	Contents
VIFn_R[0-3]	r	32 bits	Filling data (Row register)
VIFn_C[0-3]	r	32 bits	Filling data (Col register)
VIFn_CYCLE	r	32 bits	Data write cycle
VIFn_MASK	r	32 bits	Write mask pattern
VIFn_MODE	r	32 bits	Addition mode
VIFn_ITOP	r	32 bits	ITOP value
VIFn_ITOPS	r	32 bits	Following ITOP value
VIF1_BASE	r	32 bits	Double buffer base address
VIF1_OFST	r	32 bits	Double buffer offset
VIF1_TOP	r	32 bits	TOP value
VIF1_TOPS	r	32 bits	Next TOP value/data write address
VIFn_MARK	r/w	32 bits	Mark value
VIFn_NUM	r	32 bits	Amount of unwritten data
VIFn_CODE	r	32 bits	Most recently processed VIFcode

### VIF Control Register

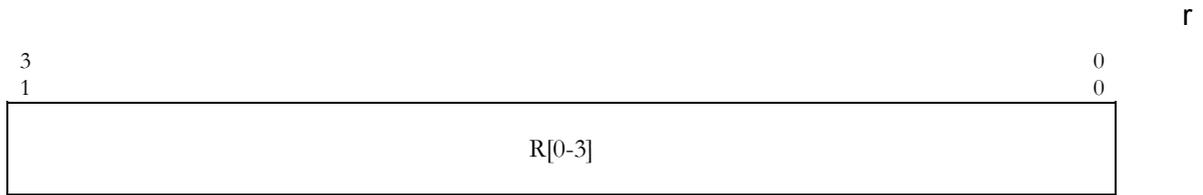
The following VIF control registers are mapped to the address space of the EE Core. These are accessible only from the EE Core.

Symbol	r/w	Width	Contents
VIF0_STAT	r	32 bits	VIF0 status
VIF0_FBRST	w	32 bits	VIF0 operation control
VIF0_ERR	r/w	32 bits	VIF0 error status
VIF1_STAT	r	32 bits	VIF1 status
VIF1_FBRST	w	32 bits	VIF1 operation control
VIF1_ERR	r/w	32 bits	VIF1 error status

### VU Control Register

All the VU0 control registers and some of the VU1 control registers are allocated to the Coprocessor 2 control registers (CCR [2,n]). The remaining VU1 control registers are mapped to VU0 VU Mem. For details, refer to a separate document "VU User's Manual".

## VIF0\_R[0-3] / VIF1\_R[0-3] : Row filling data



Name	Pos.	Contents
R	31:0	Row data for filling when decompressing

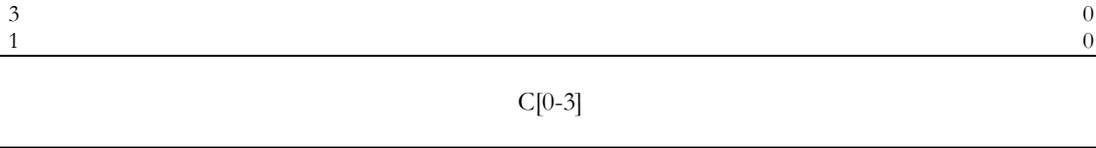
This register maintains the row data for filling when doing data decompression when the mask specifier = 1, and the offset data during addition decompression. For details, refer to "6.3.5. Skipping Write and Filling Write" and "6.3.7. Addition Decompression Write".

### Related VIFcode

STROW

**VIF0\_C[0-3] / VIF1\_C[0-3] : Column filling data**

r



Name	Pos.	Contents
C	31:0	Column data for filling when decompressing

This register maintains the column data for filling in data decompression when the mask specifier = 2. For details, refer to "6.3.5. Skipping Write and Filling Write".

**Related VIFcode**

STCOL

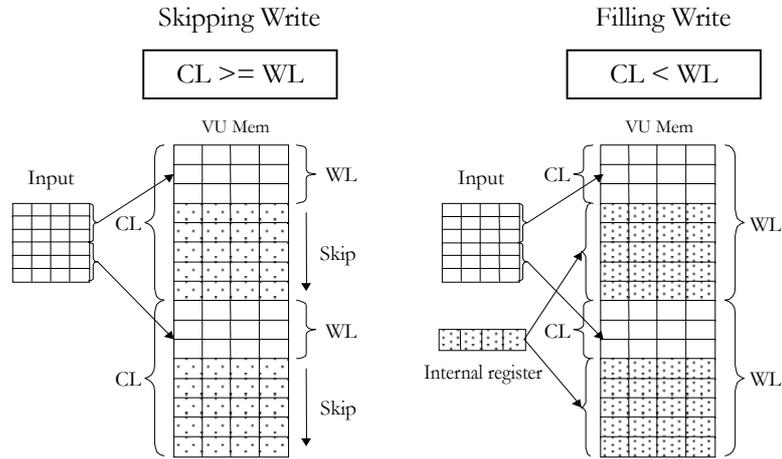
## VIF0\_CYCLE / VIF1\_CYCLE : Write cycle

r

3	1 1	0 0	0
1	6 5	8 7	0
-		WL	CL

Name	Pos.	Contents
CL	7:0	Cycle length Cycle/Block size
WL	15:8	Write cycle length Block size/Cycle

This register controls the method of writing the data decompressed by the VIFcode UNPACK to the VU Mem, and sets the block size and cycle. The figure below shows the relation between CL and WL.



For details, refer to "6.3.5. Skipping Write and Filling Write".

### Related VIFcode

STCYCL



## VIF0\_MODE / VIF1\_MODE : Addition processing mode

r

3		0 0 0
1		2 1 0
-		M O D

Name	Pos.	Contents
MOD	1:0	Addition mode 00 No addition processing 01 Offset mode (Row+dV -> VU Mem) 10 Difference mode (Row+dV -> Row -> VU Mem) 11 Undefined

This register specifies the addition processing when writing the data decompressed by the VIFcode UNPACK to the VU Mem. For details, refer to "6.3.7. Addition Decompression Write".

### Related VIFcode

STMOD

## VIF0\_ITOP / VIF1\_ITOP : Data address



Name	Pos.	Contents
ITOP	9:0	ITOP value

This register notifies the VU of the data address.

When activating a microprogram by the VIFcode MSCAL/MSCALF/MSCNT, the VIFn\_ITOPS register value is transferred to this register and becomes readable by the XITOP instruction in the microprogram.

This register is normally used to notify the microprogram of the address of the data transferred to the VU Mem.

Since the data memory capacity for VU0 is 4 Kbytes, only the lower 8 bits are zero-extended and read in the XITOP instruction of VU0.

### Related VIFcode

MSCAL, MSCALF, MSCNT, (ITOP)

## VIF0\_ITOPS / VIF1\_ITOPS : Data address



Name	Pos.	Contents
ITOPS	9:0	ITOPS value

This register notifies the VU of the data address.

The value can be set by the VIFcode ITOP.

Afterward, when activating a microprogram by the VIFcode MSCAL/MSCALF/MSCNT, the value is transferred to the VIFn\_ITOP register and becomes readable by the XITOP instruction in the microprogram.

This register is normally used to notify the microprogram of the address of the data transferred to the VU Mem.

### Related VIFcode

ITOP, (MSCAL/MSCALF/MSCNT)

## VIF1\_BASE : Double buffer base address



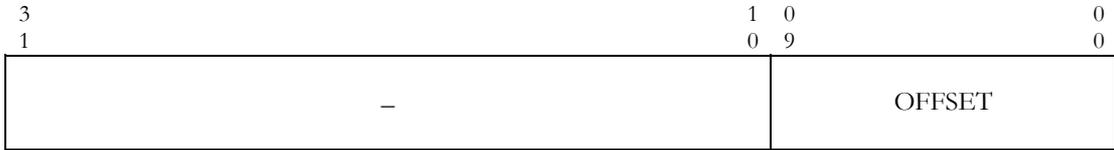
Name	Pos.	Contents
BASE	9:0	Data buffer base address

This register maintains the base address in a data buffer structured as a double buffer.  
 The value can be set by the VIFcode BASE.  
 For details, refer to "6.3.8. Double Buffering".

### Related VIFcode

BASE

## VIF1\_OFST : Double buffer offset



r

Name	Pos.	Contents
OFFSET	9:0	Data buffer offset

This register specifies the offset value in a data buffer structured as a double buffer.

The value can be set by the VIFcode OFFSET.

At this time, the VIF1\_BASE register value is transferred to the VIF1\_TOP register.

For details, refer to "6.3.8. Double Buffering".

### Related VIFcode

OFFSET

## VIF1\_TOP : Data address



Name	Pos.	Contents
TOP	9:0	Data buffer address

This register notifies the VU of the data address.  
 It becomes a pointer to the buffer that is not being used by the VIF side, in double buffering.  
 When executing a microprogram by the VIFcode MSCAL/MSCALF/MSCNT, the VIF1\_TOPS register value is transferred to this register and becomes readable by the XTOP instruction in the microprogram.

### Related VIFcode

MSCAL, MSALF, MSCNT, (BASE, OFFSET)

## VIF1\_TOPS : Data decompression destination address



Name	Pos.	Contents
TOPS	9:0	Data decompression buffer address

This register maintains the decompression destination address when data is decompressed by the VIFcode UNPACK. In double buffering, this becomes a pointer to the buffer being used by the VIF side.

When setting the value to the VIF1\_OFST register by the VIFcode OFFSET, the VIF1\_BASE register value is set.

When executing a microprogram by the VIFcode MSCAL/MSCALF/MSCNT, the value of this register is transferred to the VIF1\_TOP register. Then, if the DBF flag of the VIF1\_STAT register is 0, the VIF1\_BASE value is set again, and if it is 1, the VIF1\_BASE+VIF1\_OFST value is set again. Since the DBF flag value is reversed at this time, the value of this register will be switched whenever the microprogram is executed.

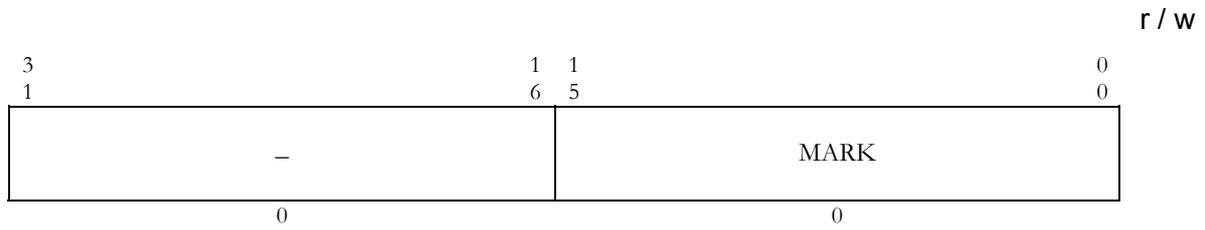
For details, refer to "6.3.8. Double Buffering".

Furthermore, when the FLG (the MSB in the IMMEDIATE field) is set to 0 by the VIFcode UNPACK, the value of this register is not referred to and the data is decompressed to the address specified by the IMMEDIATE field.

### Related VIFcode

MSCAL, MSCALF, MSCNT, (OFFSET, BASE)

**VIF0\_MARK / VIF1\_MARK : Mark value**



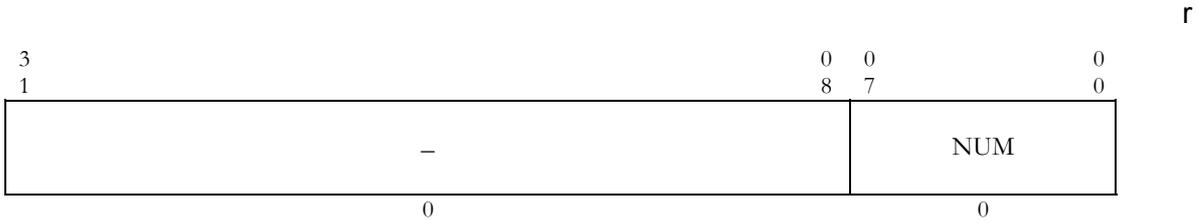
Name	Pos.	Contents
MARK	15:0	Mark value most recently set

This register maintains the mark value that has been most recently set. The value can be set by the VIFcode MARK. Unlike other registers, this register can be written from the EE Core as well. When writing from the EE Core, the MRK field of the VIFn\_STAT register is cleared to 0.

**Related VIFcode**

MARK

**VIF0\_NUM : Amount of untransferred data**

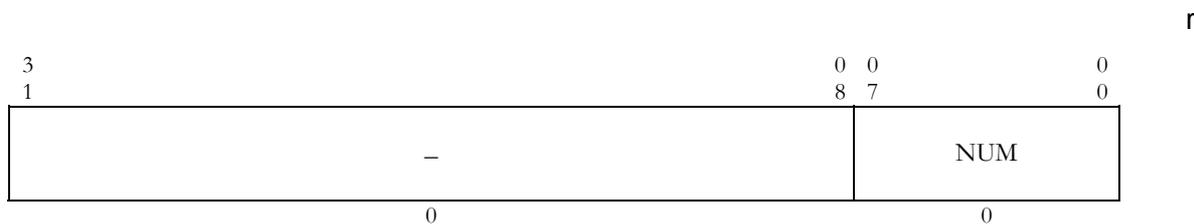


Name	Pos.	Contents
NUM	7:0	Amount of untransferred data

This register shows the amount of untransferred data when transferring data with the VIFcode MPG/UNPACK.  
 The value of the NUM field is set as the initial value, and is decremented whenever the data for one address is written to the MicroMem/VU Mem.

**Related VIFcode**  
 MPG, UNPACK

## VIF1\_NUM : Amount of untransferred data



Name	Pos.	Contents
NUM	7:0	Amount of untransferred data

This register shows the amount of untransferred data when transferring data with the VIFcode MPG/UNPACK/DIRECT/DIRECTHL.

With the VIFcode MPG, the initial setting is made by the NUM field value, and is decremented whenever one instruction (64 bits) is written to the MicroMem.

With the VIFcode UNPACK, the initial setting is made by the NUM field value, and is decremented whenever 1 qword (128 bits) is written to the VU Mem.

With the VIFcode DIRECT/DIRECTHL, the initial setting is made by the IMMEDIATE field value, and is decremented whenever 128-bit data is transferred to the GIF.

### Related VIFcode

MPG, UNPACK, DIRECT, DIRECTHL

**VIF0\_CODE / VIF1\_CODE : Most recently processed VIFcode**

r

3 1	2 2 4 3	1 1 6 5	0 0
CMD	NUM	IMMEDIATE	

Name	Pos.	Contents
IMMEDIATE	15:0	VIFcode IMMEDIATE processed most recently
NUM	23:16	VIFcode NUM processed most recently
CMD	31:24	VIFcode CMD processed most recently

This register maintains the VIFcode last processed or in process when the VIF is stopped.

**Related VIFcode**

Whole VIFcode

**VIF0\_STAT : VIF status register**

r

3	2	2	2	2	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0												
1	8	7	4	3	4	3	2	1	0	9	8	7	6	5	3	2	1	0																	
-				F																	E	E	I	V	V	V	M					V	V		
-				Q																	R	R	N	I	F	S	-	R					E	P	
-				C																	1	0	T	S	S	S		K					W	S	
0				0																	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Name	Pos.	Contents
VPS	1:0	VIF0 status 00 Idle 01 Waits for the data following VIFcode. 10 Decoding the VIFcode. 11 Decompressing/transferring the data following VIFcode.
VEW	2	VIF0 E-bit wait 0 Not-wait 1 Wait (VU0 is executing a microprogram.)
MRK	6	VIF0 MARK detect 0 Not-detect 1 Detect
VSS	8	Stop by STOP 0 Not-stall 1 Stall
VFS	9	Stop by ForceBreak 0 Not-stall 1 Stall
VIS	10	Stop by the i bit 0 Not-stall 1 Stall
INT	11	Interrupt bit detected flag 0 Not-detect 1 Detect (The i bit has been set and the VIFcode has been detected.)
ER0	12	DMAtag Mismatch error 0 No error 1 Error (DMAtag has been detected in the VIF packet.)
ER1	13	VIFcode error 0 No error 1 Error (Undefined VIFcode has been detected.)
FQC	27:24	Amount of effective data in the VIF0-FIFO 0000 0 qword (empty) 0001 1 qword :: 1000 8 qwords (full) Others Reserved

This register shows the VIF0 status.

The MRK field is set when VIFcode MARK is detected and cleared to 0 when the EE Core writes to the VIF0\_MARK register.

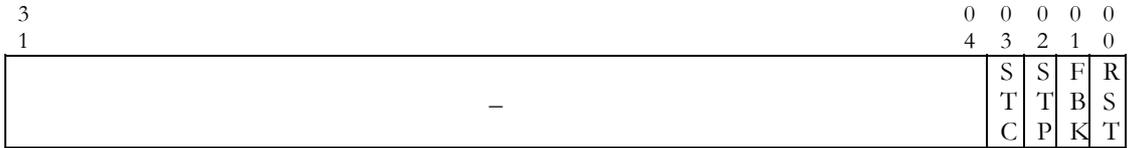
VSS, VFS, VIS, INT, ER0, and ER1 are cleared to 0 by setting the STC bit of the VIF0\_FBRST register to 1.

The ER0 bit cannot be used appropriately due to a failure in determining DMAtag Mismatch error.

Mask the DMAtag Mismatch error detection feature by setting VIF0\_ERR.ME0 to 1.

### VIF0\_FBRST : VIF reset register

W

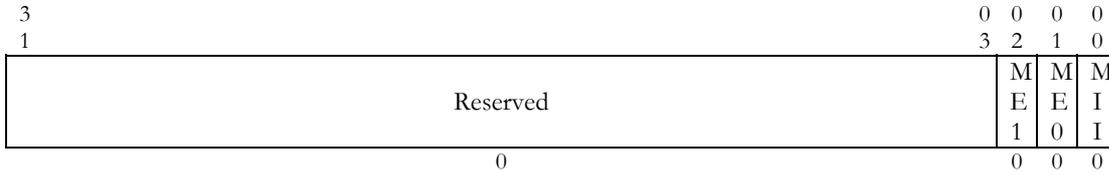


Name	Pos.	Contents
RST	0	Reset Resets the whole VIF0 (including VIF0-FIFO) when 1 is written.
FBK	1	ForceBreak Causes a ForceBreak to the VIF0 when 1 is written. (Stall occurrence)
STP	2	STOP Stops after end of VIFcode in process when 1 is written. (Stall occurrence)
STC	3	Stall Cancel Cancels the VIF0 stall and clears the VSS, VFS, VIS, INT, ER0, and ER1 of the VIF0_STAT register to 0 when 1 is written.

This register stops/restarts/resets the VIF0 processing.

### VIF0\_ERR : Error mask register

r / w

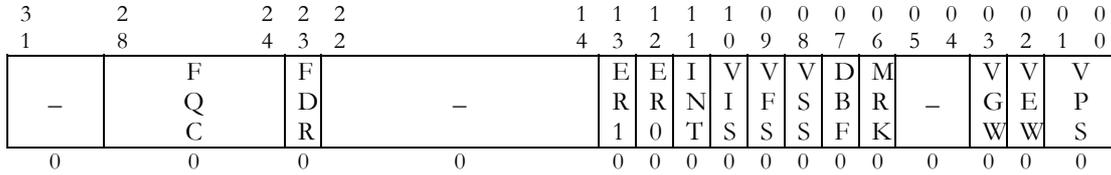


Name	Pos.	Contents
MII	0	Masks an interrupt by the i bit of VIFcode. 0 Unmask (i bit interrupt enable) 1 Mask (i bit interrupt disable)
ME0	1	DMAtag Mismatch error mask 0 Unmask (stalls when an error occurs.) 1 Mask (ignores DMAtag Mismatch error.)
ME1	2	VIFcode error mask 0 Unmask (stalls when an error occurs.) 1 Mask (considered as VIFcode NOP.)

This register sets the VIF0 error mask and reads the setting status.  
 Due to a failure in determining DMAtag Mismatch error, setting ME0 bit to 0 may generate an error even though a correct packet has been transmitted. Set the ME0 bit to 1 when using it. In V3\_16 format, set the ME1 bit to 1.

### VIF1\_STAT : VIF status register

r



Name	Pos.	Contents
VPS	1:0	VIF1 status 00 Idle 01 Waits for the data following VIFcode 10 Decoding the VIFcode 11 Decompressing/transferring the data following VIFcode
VEW	2	VIF1 E-bit wait 0 Not-wait 1 Wait (VU1 is executing a microprogram)
VGW	3	Status waiting for end of GIF transfer 0 Not-wait 1 Wait (Stopped status with one of FLUSH/FLUSHA, DIRECT/DIRECTHL and MSCALF commands when GIF! = idle.)
MRK	6	VIF1 MARK detect 0 Not-detect 1 Detect
DBF	7	Double buffer flag Cleared to 0 by OFFSET and reversed by MSCAL/MSCALF/MSCNT. 0 TOPS=BASE 1 TOPS=BASE+OFFSET
VSS	8	Stop by STOP 0 Not-stall 1 Stall
VFS	9	Stop by ForceBreak 0 Not-stall 1 Stall
VIS	10	VIF1 interrupt stall 0 Not-stall 1 Stall
INT	11	Interrupt by the i bit 0 Not-detect 1 Detect
ER0	12	DMAtag Mismatch error 0 No error 1 Error (DMAtag has been detected in the VIF packet.)
ER1	13	VIFcode error 0 Not-detect 1 Detect (Undefined VIFcode has been detected.)
FDR	23	VIF1-FIFO transfer direction 0 Main memory/SPRAM -> VIF1 1 VIF1 -> Main memory/SPRAM

Name	Pos.	Contents
FQC	28:24	Amount of effective data in the VIF1-FIFO 00000 0 qword (empty) 00001 1 qword ∴ 10000 16 qwords (full) Others Reserved

This register shows the VIF1 status.

Only the FDR field is readable/writable.

VSS, VFS, VIS, INT, ER0, and ER1 are cleared to 0 by writing 1 to the STC bit of the VIF1\_FBRST register.

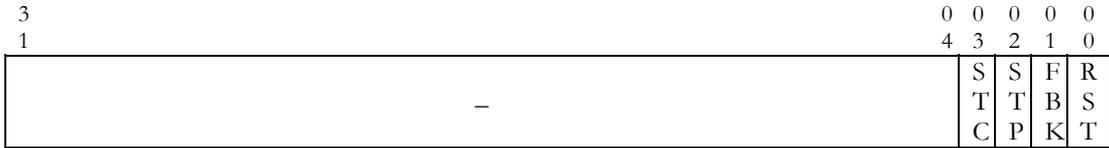
The MRK field is set when the VIFcode MARK is detected and cleared to 0 when the VIF1\_MARK register is written from the EE Core.

The ER0 bit cannot be used appropriately due to a failure in determining DMAtag Mismatch error.

Mask the DMAtag Mismatch error detection feature by setting VIF0\_ERR.ME0 to 1.

### VIF1\_FBRST : VIF reset register

W

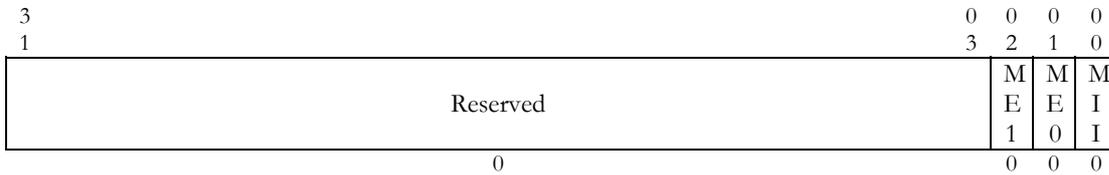


Name	Pos.	Contents
RST	0	Reset Resets the whole VIF1 (including VIF1-FIFO) when 1 is written.
FBK	1	ForceBreak Causes a ForceBreak to the VIF1 when 1 is written. (Stall occurrence)
STP	2	Stop Stops after end of VIFcode in process when 1 is written. (Stall occurrence)
STC	3	Stall Cancel Cancels the VIF1 stall and clears the VSS, VFS, VIS, INT, ER0, and ER1 of the VIF1_STAT register to 0 when 1 is written.

This register stops/restarts/resets the VIF1 processing.

### VIF1\_ERR : VIF error mask register

r / w



Name	Pos.	Contents
MII	0	Masks an interrupt by the i bit of the VIFcode. 0 Unmask (i bit interrupt enable) 1 Mask (i bit interrupt disable)
ME0	1	DMAtag Mismatch error mask 0 Unmask (stalls when an error occurs.) 1 Mask (ignores a DMAtag Mismatch error.)
ME1	2	VIFcode error mask 0 Unmask (stalls when an error occurs.) 1 Mask (considered as VIFcode NOP)

This register sets the VIF1 error mask and reads the setting status.

Due to a failure in determining DMAtag Mismatch error, setting ME0 bit to 0 may generate an error even though a correct packet has been transmitted. Set the ME0 bit to 1 when using it. In V3\_16 format, set the ME1 bit to 1.

## 7. GIF: GS Interface

---

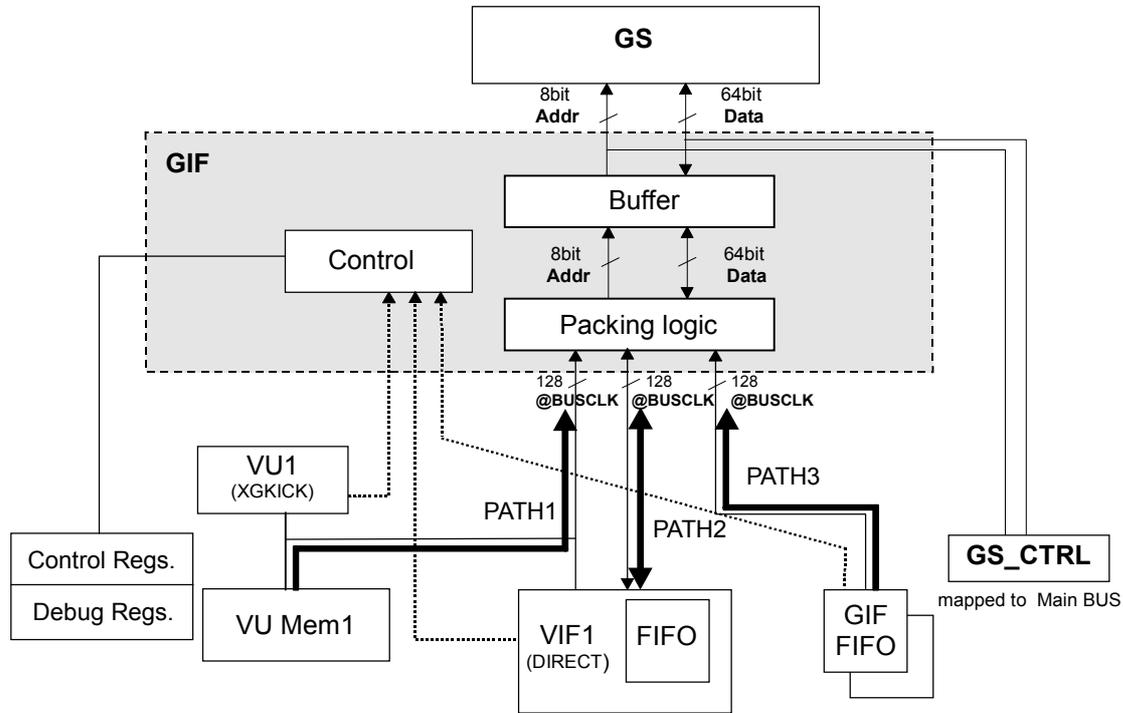
The GIF is an interface between the EE and the GS.

Data sent from the EE to the GS is usually generated in VU1 (simple stream) or the EE Core and VU0 (complex stream).

Although the data are generated and transferred in parallel, they are arbitrated by the GIF, and can be allocated to two contexts of the GS.

## 7.1. Data Transfer Route

The schematic representation of the GIF is shown as follows.



### 7.1.1. General Data Transfer Path

The GIF has three kinds of general data transfer paths.

#### PATH1

PATH1 is the data transfer path from VPU1 data memory (VU Mem1) to the GS.

When VU1 executes the XGKICK instruction, a GS packet (described later in this document) stored in the address specified with an argument is transferred via this path.

If an XGKICK instruction is executed before the end of the transfer operation via PATH1, it stalls. An XGKICK instruction executed during the transfer via PATH2 and PATH3 does not stall, and one instruction is queued. However, when the next XGKICK instruction is executed while the instruction is queued, stalls are generated.

#### PATH2

PATH2 is the data transfer path between the FIFO in the VPU1 VIF and the GIF. This path is used when executing the DIRECT/DIRECTHL instruction in the VIF, transferring data from the GS to main memory by using the image data transfer function of the GS, etc.

#### PATH3

PATH3 is the direct data transfer path from the EE main bus to the GIF. This path is used when transferring data from main memory or scratchpad memory to the GS on DMA channel 2.

### 7.1.2. Priority and Timing

The three general data transfer paths are prioritized as PATH1>PATH2>PATH3. Whenever transfer of a GS packet (described later in this document) in a path ends, transfer requests from other paths are checked. If there is a request, transfer processing is performed according to priority.

During transfer arbitration, an idle state of 1 cycle (@147.456MHz) occurs. Note that outputs are not consecutive even if transfer requests are made consecutively.

In addition, in the cases where the PRIM field in the GIFtag is not output to the PRIM register, an idle state of 1 cycle is generated.

### 7.1.3. PATH3 Transfer Mode

When transferring data of the IMAGE mode (described later in this document) via PATH3, the continuous transfer mode and the intermittent transfer mode can be specified.

In the intermittent transfer mode, transfer requests from other paths are checked whenever 1 slice (8 qwords) is transferred. If there is a transfer request via PATH1 or a transfer request to use PATH2 by the DIRECT instruction, priority in transfer is given to them. (Priority is not given to the transfer request via PATH2 by the DIRECTHL instruction.) If there is no transfer request from other paths, transfer processing via PATH3 is executed next.

When using the intermittent transfer mode, it is possible to transfer large sized image data by splitting into smaller parts at intervals of the transfer via PATH1.

The transfer mode is specified in the IMT field of the GIF\_STAT register. The continuous transfer mode is entered when the IMT field value is 0, and the intermittent transfer mode is entered when the IMT field value is 1.

### 7.1.4. PATH3 Operation Control

The transfer operation via PATH3 can be masked (suppressed) if necessary. There are two kinds of mask factors:

- MSKPATH3 command of the VIF
- M3R field of the GIF\_MODE register

These factors are independent of each other. The mask set by the VIF can be cancelled only by the VIF, and the mask set by the GIF\_MODE register can be cancelled only by the GIF\_MODE register.

When either mask is in effect, the arbitration switch to PATH3 is not performed. This means the transfer via PATH3 becomes possible only when both the masks have been cancelled.

### 7.1.5. GS Privileged Register

The privileged register of the GS is directly mapped to the I/O space of the EE Core, and is not accessible via the GIF regardless of the state of the general data transfer path.

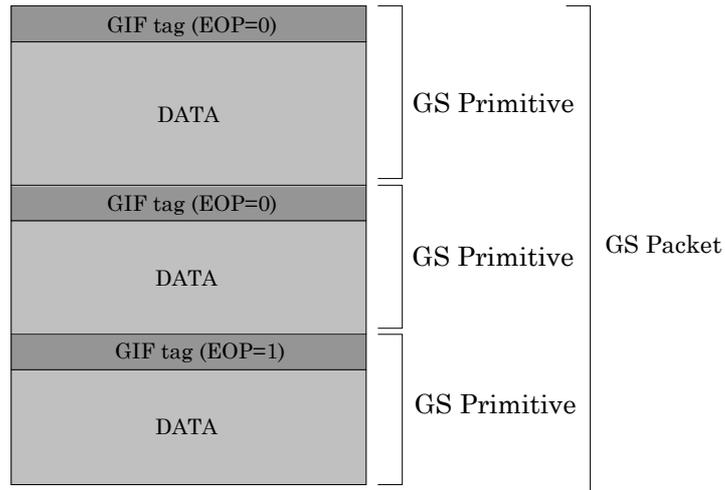
The GIF monitors access to the privileged register. When the transfer direction switch register (BUSDIR) is accessed, the GIF sets the data transfer direction in the GIF according to the contents of the output data.

When the privileged register is accessed, and there is no response from the GS because of a hardware failure, a bus error is generated after a stall for 1024 cycles.

## 7.2. Data Format

### 7.2.1. GS Packet

The basic unit of data transferred by the GIF is a GS primitive, consisting of a leading GIFtag and following data. Transfer processing is performed in units of GS packets, consisting of two or more GS primitives. The last GS primitive in a GS packet is indicated by the termination information (EOP=1) in the GIFtag.



This structure is common to all the general data transfer paths (PATH1, PATH2, and PATH3). For PATH2 and PATH3, however, the VIFcode and DMATag are put in front of the GS packet. It is necessary to align the GIFtag and data on a 128-bit boundary in memory.

### 7.2.2. GIFtag

The GIFtag has a 128-bit fixed length, and specifies the size and structure of the subsequent data and the data format (mode). The structure of the GIFtag is as follows:

1		0	0	0	0	0	0	0	0	0	0
2		6	6	6	5	5	5	4	4	1	1
7		4	3	0	9	8	7	7	6	5	4

REGS (max 16)	N	F	PRIM	P	-	E	NLOOP
	R	L		R		O	
	E	G		E		P	
	G	G					

Name	Pos.	Contents
NLOOP	14:0	Repeat count (GS primitive data size) PACKED mode    NREG x NLOOP(qword) REGLIST mode    NREG x NLOOP(dword) IMAGE mode       NLOOP(qword)
EOP	15	Termination information (End Of Packet) 0    With following primitive 1    Without following primitive (End of GS packet)
PRE	46	PRIM field enable 0    Ignores PRIM field 1    Outputs PRIM field value to PRIM register
PRIM	57:47	Data to be set to the PRIM register of GS
FLG	59:58	Data format 00    PACKED mode 01    REGLIST mode 10    IMAGE mode 11    Disable (Same operation with the IMAGE mode)
NREG	63:60	Register descriptor Number of register descriptors in REGS field When the value is 0, the number of descriptors is 16.
REGS	127:64	Register descriptor (4 bits x 16 max.) Details are described later.

The value of the NLOOP field shows the data size of GS primitive, but the unit varies depending on the data format. Moreover, when NLOOP is 0, the GIF does not output anything, and values other than the EOP field are disregarded.

### 7.2.3. Register Descriptor

The register descriptor specified in the REGS field specifies to which register of the GS the input data following the GIFtag should be output and also specifies the effective area of the input data (packing format). The value and meaning of the register descriptor are as follows.

Register descriptor	Meaning	Output destination register address	Significant part
0x00	PRIM	0x00	Described later
0x01	RGBAQ	0x01	Described later
0x02	ST	0x02	Described later
0x03	UV	0x03	Described later
0x04	XYZF2	0x04/0x0c (Switched by ADC of Input data)	Described later
0x05	XYZ2	0x05/0x0d (Switched by ADC of Input data)	Described later
0x06	TEX0_1	0x06	Lower 64 bits
0x07	TEX0_2	0x07	Lower 64 bits
0x08	CLAMP_1	0x08	Lower 64 bits
0x09	CLAMP_2	0x09	Lower 64 bits
0x0a	FOG	0x0a	Described later
0x0b	-	-	Reserved
0x0c	XYZF3	0x0c	Lower 64 bits
0x0d	XYZ3	0x0d	Lower 64 bits
0x0e	A+D	(Specified by upper 64 bits)	Lower 64 bits
0x0f	NOP	(Not output)	(Not output)

Up to 16 register descriptors can be specified for one GIFtag. When two or more register descriptors are specified, they are applied in order starting from the least significant bit. For more concrete information, refer to "7.3.1. PACKED Mode Operation" and "7.4.1. REGLIST Mode Operation".

### 7.3. PACKED Mode

In the PACKED mode, the input data of each qword is interpreted and packed according to the register descriptor in the GIFtag, and is output to the address specified in the same way by the register descriptor.

#### 7.3.1. PACKED Mode Operation

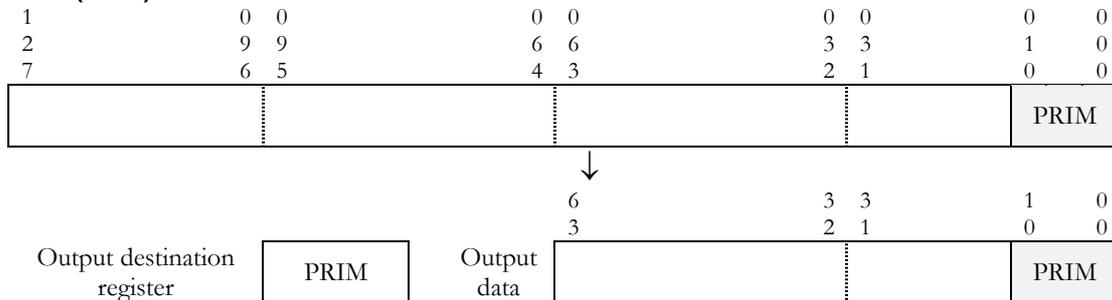
The actual operation in PACKED mode is as follows:

1. The PRIM field value of the GIFtag is output to the PRIM register. This operation is performed only when the value of the PRE field is 1. When the value is 0, the operation becomes idle for 1 cycle (@147.456MHz).
2. The first qword data is packed according to the least significant bits of the register descriptor (bits 67:64), and is output.
3. The second qword data is packed according to the next register descriptor (bits 71: 68), and is output. This process is repeated NREGS.
4. Steps 2 and 3 are repeated NLOOP times.

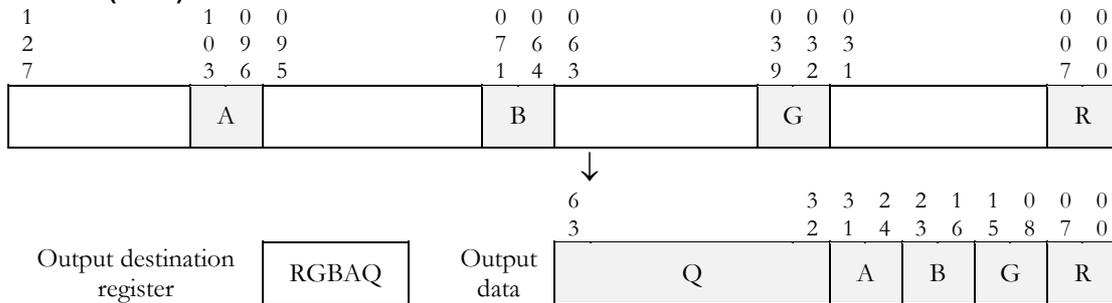
#### 7.3.2. Packing Format

The packing format for input data is specified by the register descriptor. Following are the packing formats corresponding to each register descriptor and their output destination registers.

##### PRIM(0x00)

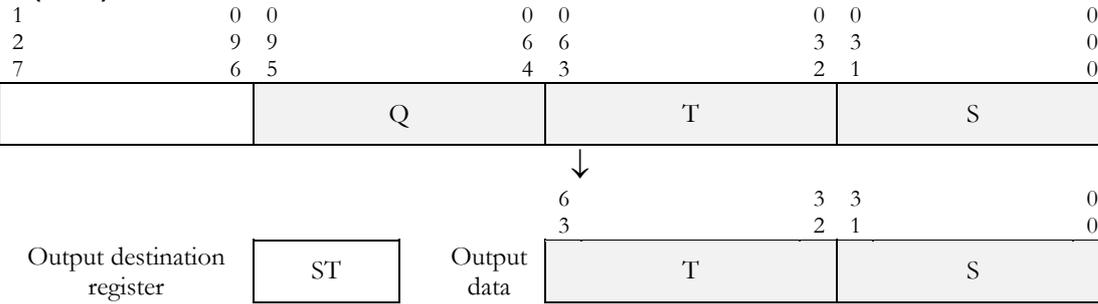


##### RGBAQ(0x01)



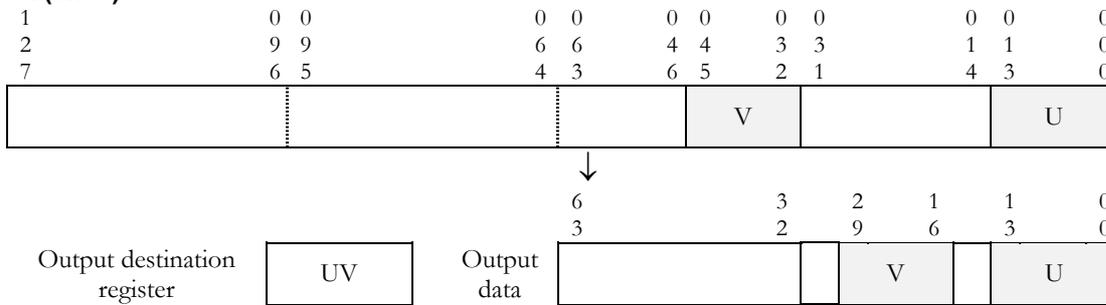
For the Q value, the value most recently stored in the internal register by the ST register descriptor is used. The initial value is 1.0, and the Q value is initialized whenever the GIFtag is read.

**ST(0x02)**



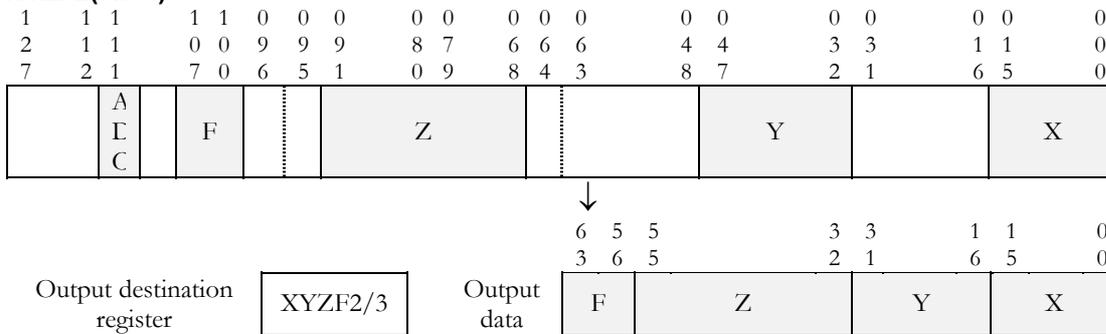
S, T, and Q are 32-bit single precision floating-point numbers (conforming to the VU format). The Q value is stored in the internal register of the GIF, and is output by the following RGBAQ register descriptor.

**UV(0x03)**



U and V are 14-bit unsigned fixed-point numbers (fractional part: 4 bits).

**XYZF2(0x04)**



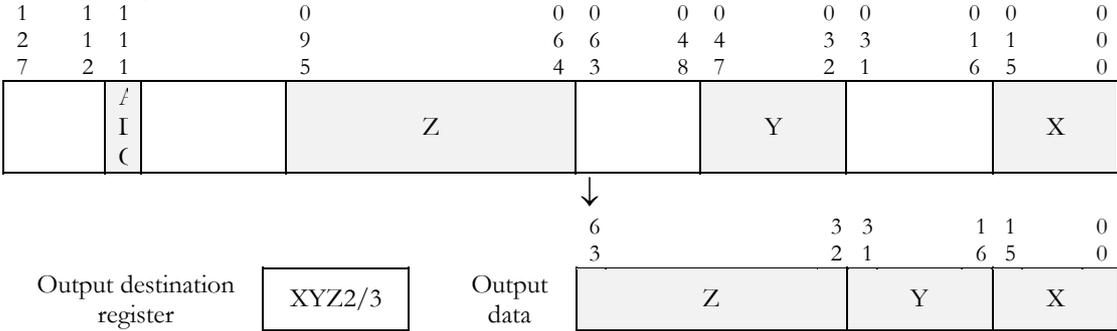
X and Y are 16-bit signed fixed-point numbers (fractional part: 4 bits)

Z is a 24-bit unsigned integer, and F is an 8-bit unsigned integer. The upper 28 bits are retrieved from each 32-bit input data.

ADC is the flag to switch the output address, and is output to XYZF2 (0x04) with 0 and is output to XYZF3 (0x0c) with 1.

That is, 0 is specified when performing Drawing Kick and 1 is specified when not performing Drawing Kick.

**XYZ2(0x05)**



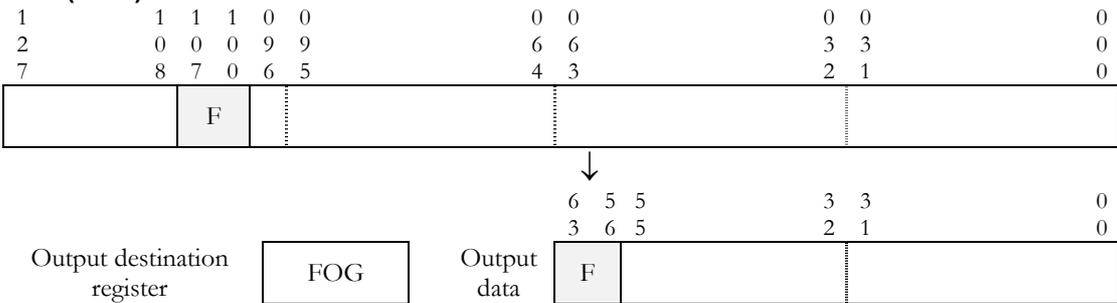
X and Y are 16-bit signed fixed-point numbers (fractional part: 4 bits).

Z is a 32-bit unsigned integer.

ADC is the flag to switch the output address, and is output to XYZ2 (0x05) with 0 and is output to XYZ3 (0x0d) with 1.

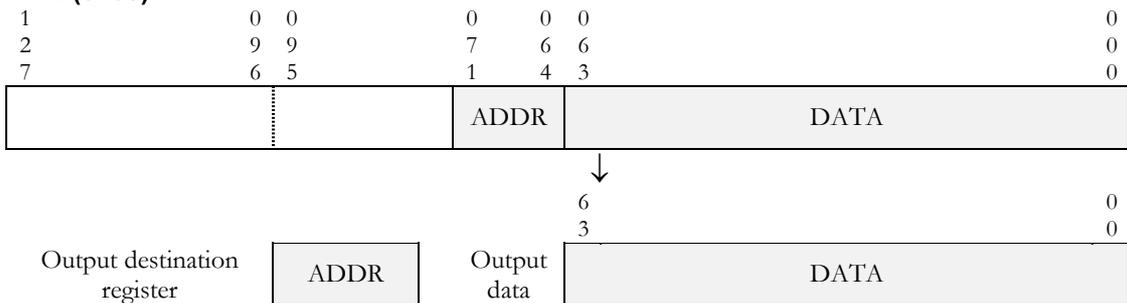
That is, 0 is specified when performing Drawing Kick and 1 is specified when not performing Drawing Kick.

**FOG(0x0a)**



F is an 8-bit unsigned integer. The integer part of the fixed-point value of the fractional part 4 bits is retrieved.

**A+D(0x0e)**

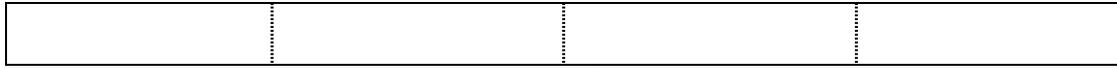


Data is output to the register specified by the ADDR.

When transferring data via PATH1 and PATH3 in parallel, do not apply the A+D packing format to the data on PATH3. This may cause the GS to hang on rare occasions.

**NOP(0x0f)**

1	0 0	0 0	0 0	0
2	9 9	6 6	3 3	0
7	6 5	4 3	2 1	0



↓  
Not output.

**7.3.3. Operation Example in PACKED Mode**

The following are examples of the GS packet input data string and output data to the GS in the PACKED mode.

**Address/Data Direct Specification**

GS packet			GIFtag	Output data
GIFtag			NREG=1 REGS={A+D} PRE=0 NLOOP=3	→ TEX0, ALPHA, TEST
--	TEX0	TEX0-DATA		
--	ALPHA	ALPHA-DATA		
--	TEST	TEST-DATA		

**Independent Triangle, Flat-NoTexture**

This is an example in which 2 triangles are drawn with flat shading and without texture. The type and attribute of the primitive are specified in the PRIM field of the GIFtag.

GS packet				GIFtag	Output data
GIFtag				NREG=4 REGS={ RGBAQ, XYZF2, XYZF2, XYZF2, XYZF2 } PRE=1 NLOOP=2	→ PRIM, RGBAQ, XYZF2, XYZF2, XYZF2, RGBAQ, XYZF2, XYZF2, XYZF2
A0	B0	G0	R0		
F0	Z0	Y0	X0		
F1	Z1	Y1	X1		
F2	Z2	Y2	X2		
A0	B0	G0	R0		
F0	Z0	Y0	X0		
F1	Z1	Y1	X1		
F2	Z2	Y2	X2		

**Independent Triangle, Flat-NoTexture(PRIM Field Unused)**

This is an example in which 2 triangles are drawn with flat shading and without texture. The PRIM field of the GIFtag is not used, and the type and attribute of the primitive are specified in the data.

GS packet				GIFtag		→	Output data	
GIFtag				NREG=4				PRIM, RGBAQ, XYZF2, XYZF2, XYZF2, PRIM, RGBAQ, XYZF2, XYZF2, XYZF2
--		PRIM		REGS={				
A0	B0	G0	R0	PRIM,				
F0	Z0	Y0	X0	RGBAQ,				
F1	Z1	Y1	X1	XYZF2,				
F2	Z2	Y2	X2	XYZF2,				
--		PRIM		XYZF2				
A0	B0	G0	R0	}				
F0	Z0	Y0	X0	PRE=0				
F1	Z1	Y1	X1	NLOOP=2				
F2	Z2	Y2	X2					

**Independent Triangle, Gouraud-NoTexture**

This is an example in which one triangle is drawn with Gouraud shading and without texture.

GS packet				GIFtag		→	Output data	
GIFtag				NREG=6				PRIM, RGBAQ, XYZF2, RGBAQ, XYZF2, RGBAQ, XYZF2
A0	B0	G0	R0	REGS={				
F0	Z0	Y0	X0	RGBAQ,				
A1	B1	G1	R1	XYZF2,				
F1	Z1	Y1	X1	RGBAQ,				
A2	B2	G2	R2	XYZF2,				
F2	Z2	Y2	X2	RGBAQ,				
				XYZF2				
				}				
				PRE=1				
				NLOOP=1				

**Independent Triangle, with Texture, Independent Texture**

This is an example in which one triangle is drawn by specifying the texture with the TEX0\_1 register.

GS packet				GIFtag		→	Output data	
GIFtag				NREG=10				PRIM, TEX0_1, ST, RGBAQ, XYZF2, ST, RGBAQ, XYZF2, ST, RGBAQ, XYZF2
--		TEX0-DATA		REGS={				
--	Q0	T0	S0	TEX0_1,				
A0	B0	G0	R0	ST,				
F0	Z0	Y0	X0	RGBAQ,				
--	Q1	T1	S1	XYZF2,				
A1	B1	G1	R1	ST,				
F1	Z1	Y1	X1	RGBAQ,				
--	Q2	T2	S2	XYZF2,				
A2	B2	G2	R2	ST,				
F2	Z2	Y2	X2	RGBAQ,				
				XYZF2				
				}				
				PRE=1				
				NLOOP=1				

**Independent Triangle, with Texture, Shared Texture**

GS packet				GIFtag	→	Output data
GIFtag						
--	TEX0-DATA					
GIFtag				PRE=1 NLOOP=1		
--	Q0	T0	S0			
A0	B0	G0	R0	NREG=9 REGS={ ST, RGBAQ, XYZF2, ST, RGBAQ, XYZF2 } PRE=0 NLOOP=1		
F0	Z0	Y0	X0			
--	Q1	T1	S1			
A1	B1	G1	R1			
F1	Z1	Y1	X1			
--	Q2	T2	S2			
A2	B2	G2	R2			
F2	Z2	Y2	X2			

**TriangleStrip, with Texture, Shared Texture**

GS packet				GIFtag	→	Output data
GIFtag						
--	TEX0-DATA					
GIFtag				PRE=1 NLOOP=1		
--	Q0	T0	S0			
A0	B0	G0	R0	NREG=3 REGS={ ST, RGBAQ, XYZF2 } PRE=0 NLOOP=3		
F0	Z0	Y0	X0			
--	Q1	T1	S1			
A1	B1	G1	R1			
F1	Z1	Y1	X1			
--	Q2	T2	S2			
A2	B2	G2	R2			
F2	Z2	Y2	X2			

## 7.4. REGLIST Mode

In REGLIST mode, the data following the GIFtag is considered to be data strings of 64 bits x 2 and output as is without packing by setting the register descriptor value as the output destination address.

When the packed data can be prepared in advance, the size of the GS packet can be reduced by outputting the data in this mode.

### 7.4.1. REGLIST Mode Operation

The actual operation in the REGLIST mode is as follows:

1. The lower 64 bits of the first qword data are output according to the register descriptor in the least significant position (bits 67:64).
2. The upper 64 bits of the first qword data are output according to the next register descriptor (bits 71:68).
3. The lower 64 bits of the following qword data are output according to the following register descriptor (bits 75:72).
4. The operation is continued for NREGS times.
5. Steps 1 to 4 are repeated NLOOP times.

When the register descriptor value is A+D, it is processed as NOP. The number of data items becomes NREG x NLOOP pieces (in units of 64 bits). When an odd number is obtained, however, the upper 64 bits of the last qword data are discarded.

Moreover, the PRIM field and PRE field of the GIFtag are disregarded. Therefore, an idle state of 1 cycle is caused at the start of the GS primitive.

### 7.4.2. Operation Example in REGLIST Mode

This is an example in which the vertex data string is output in the REGLIST mode.

Input data alignment		GIFtag	→	Output data
GIFtag				
RGBAQ	ST	NREG=3 REGS={ ST, RGBAQ, XYZF2 } PRE=0 NLOOP=3		ST, RGBAQ, XYZF2, ST, RGBAQ, XYZF2, ST, RGBAQ, XYZF2
ST	XYZF			
XYZF	RGBAQ			
RGBAQ	ST			
--	XYZF			

## 7.5. IMAGE Mode

In the IMAGE mode, the data following the GIFtag is considered to be data strings of 64 bits x 2 and output to the host-local transfer register HWREG (addr = 0 x 54) of the GS.

The IMAGE mode is used when transferring image data such as texture data to the Graphics Synthesizer.

### 7.5.1. IMAGE Mode Operation

The concrete operation in the IMAGE mode is as follows.

1. The lower 64 bits of the first qword data are output to the HWREG.
2. The upper 64 bits of the first qword data are output to the HWREG.
3. Steps 1 and 2 are repeated NLOOP times.

The specification for the REGS, NREG, PRIM, and PRE fields of the GIFtag is disregarded in the IMAGE mode. Therefore, an idle state of 1 cycle is caused.

### 7.5.2. Operation Example in IMAGE Mode

This is an example in which image data is transferred in the IMAGE mode.

GS packet		GIFtag	→	Output data
GIFtag				
IMAGE1	IMAGE0	NLOOP=5 (Other fields have no meaning.)		IMAGE0
IMAGE3	IMAGE2			IMAGE1
IMAGE5	IMAGE4			:
IMAGE7	IMAGE6			:
IMAGE9	IMAGE8			IMAGE9

## 7.6. GIF Control Register

The following registers are I/O-mapped to control the GIF.

Register name	r/w	Width	Function	Initial value
GIF_CTRL	w	32 bits	Control register	Indeterminate
GIF_MODE	w	32 bits	Mode setting register	Indeterminate
GIF_STAT	r	32 bits	Status register	All 0
GIF_TAG0	r*	32 bits	GIFtag register (bits 31:0)	Indeterminate
GIF_TAG1	r*	32 bits	GIFtag register (bits 63:32)	Indeterminate
GIF_TAG2	r*	32 bits	GIFtag register (bits 95:64)	Indeterminate
GIF_TAG3	r*	32 bits	GIFtag register (bits 127:96)	Indeterminate
GIF_CNT	r*	32 bits	Count register	Indeterminate
GIF_P3CNT	r*	32 bits	PATH3 count register	Indeterminate
GIF_P3TAG	r*	32 bits	PATH3 tag register	Indeterminate

r\* shows the register is accessible only when stopped temporarily by the PSE flag of the GIF\_CTRL register.

In addition, the privileged register of the GS is mapped to the memory space of the CPU.

## GIF\_CTRL Register : Control register

W

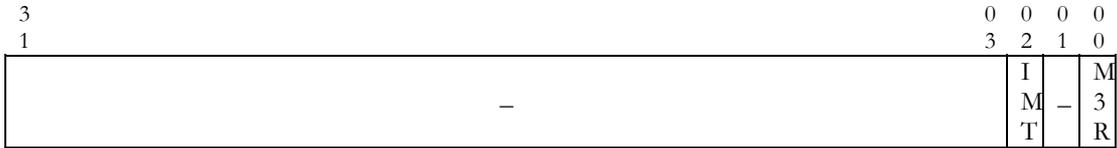
3		0	0	0	0	0
1		4	3	2	1	0
-		P	S	-	R	T
		E				S
					T	

Name	Pos.	Contents
RST	0	GIF reset 1 Reset
PSE	3	Temporary transfer stop 1 Temporary stop 0 Transfer restart

When writing with the RST field set to 1, the GIF is reset and the internal register returns to the initial value. The PSE field controls transfer processing stop and restart. During transfer stop, it is possible to read the register for debugging. There is no influence in the operation even if 0 is written when the register is not stopped temporarily.

## GIF\_MODE Register : Mode setting register

W



Name	Pos.	Contents
M3R	0	PATH3 Mask 0 Mask cancel (Initial value) 1 Mask
IMT	2	PATH3 transfer mode specification 0 Continuous transfer mode (Initial value) 1 Intermittent transfer mode in every 8 qwords

When writing with the M3R field set to 1, PATH3 is masked. When this field is set to 1 during transmission operation, the mask is enabled after the end of transmission.

For the intermittent transfer mode by the specification of the IMT field, refer to "7.1.3. PATH3 Transfer Mode".

### GIF\_STAT Register : Status register

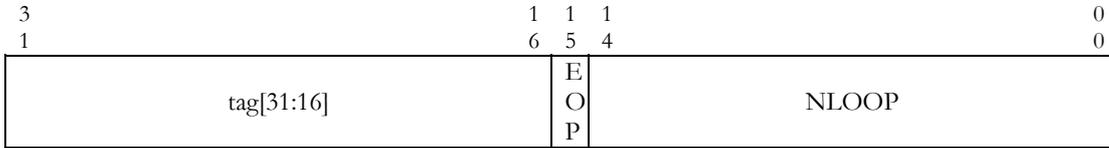
r

3 1	2 9	2 8	2 4	2 3	1 3	1 2	1 1	1 0	1 9	0 8	0 7	0 6	0 5	0 4	0 3	0 2	0 1	0 0
-	FQC	-	-	-	DIR	APATHH	OPH	P1Q	P2Q	P3Q	I3	-	PSE	IMT	M3P	M3R	M3R	M3R

Name	Pos.	Contents
M3R	0	PATH3 mask status 0 Enable (Initial value) 1 Disable (Masked by MR3 flag of GIF_MODE register.)
M3P	1	PATH3 VIF mask status 0 Enable (Initial value) 1 Disable (Masked by MASKP3 of VIF.)
IMT	2	PATH3 IMT status 0 Continuous transfer mode (Initial value) 1 Intermittent transfer mode
PSE	3	Temporary transfer stop 0 Normal (Initial value) 1 Temporary stop state by PSE flag of GIF_CTRL register
IP3	5	Interrupted PATH3 0 No interrupted transfer via PATH3 (Initial value) 1 Interrupted transfer via PATH3
P3Q	6	PATH3 in queue 0 No request to wait for processing in PATH3 (Initial value) 1 Request to wait for processing in PATH3
P2Q	7	PATH2 in queue 0 No request to wait for processing in PATH2 (Initial value) 1 Request to wait for processing in PATH2
P1Q	8	PATH1 in queue 0 No request to wait for processing in PATH1(Initial value) 1 Request to wait for processing in PATH1
OPH	9	Output Path 0 Idle state (Initial value) 1 Outputting data
APATH	11:10	Data path transferring data 00 Idle state (Initial value) 01 Transferring data via PATH1 10 Transferring data via PATH2 11 Transferring data via PATH3
DIR	12	Transfer direction 0 EE to GS (Initial value) 1 GS to EE
FQC	28:24	Effective data count in GIF-FIFO (0-16: in qword units)

### GIF\_TAG0 : GIFtag save register

r



Name	Pos.	Contents
NLOOP	14:0	Most recently read NLOOP field of GIFtag
EOP	15	Most recently read EOP flag of GIFtag
tag	31:16	Values from bit 31 to bit 16 among most recently read GIFtag values (Undefined)

This register is accessible only when stopped temporarily by the GIF\_CTRL register.

## GIF\_TAG1 : GIFtag save register

r

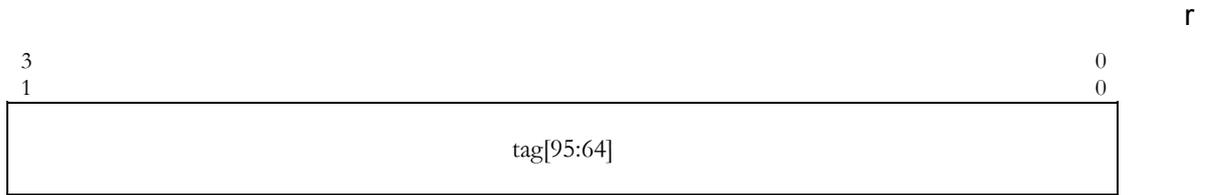
3 1	2 2 2 2 8 7 6 5	1 1 1 5 4 3	0 0
NREG	F L G	PRIM	P R E
		tag[45:32]	

Name	Pos.	Contents
tag	13:0	Values from bit 45 to bit 32 among most recently read GIFtag values (Undefined)
PRE	14	Most recently read PRE flag of GIFtag
PRIM	25:15	Most recently read PRIM field of GIFtag
FLG	27:26	Most recently read FLG field of GIFtag
NREG	31:28	Most recently read NREG field of GIFtag

This register is accessible only when stopped temporarily by the GIF\_CTRL register.

## GIF\_TAG2 : GIFtag save register

---



Name	Pos.	Contents
REGS	31:0	Values from bit 95 to bit 64 among most recently read GIFtag values (Lower part of REGS field)

This register is accessible only when stopped temporarily by the GIF\_CTRL register.

## GIF\_TAG3 : GIFtag save register

---

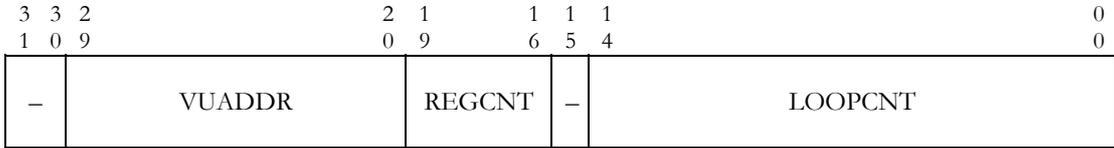


Name	Pos.	Contents
REGS	31:0	Values from bit 127 to bit 96 among most recently read GIFtag values (Upper part of REGS field)

This register is accessible only when stopped temporarily by the GIF\_CTRL register.

### GIF\_CNT : Count register

r

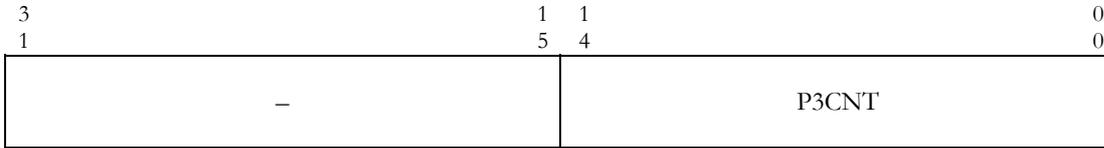


Name	Pos.	Contents
LOOPCNT	14:0	Value of current loop counter (backward counter from NLOOP) Decrementated when starting processing of the NREGSth register descriptor.
REGCNT	19:16	Register descriptor No. (0 - 15) currently in process 1    Lowest significant register descriptor of REGS field 2    2 <sup>nd</sup> lowest significant register descriptor of REGS field (Continued in the same way) 15   15 <sup>th</sup> lowest significant register descriptor of REGS field 0    Most significant register descriptor of REGS field
VUADDR	29:20	Address of VU memory under transfer

This register is accessible only in pause status by the GIF\_CTRL register.  
 The value of the REGCNT field becomes indeterminate in the IMAGE mode.

## GIF\_P3CNT : PATH3 count register

r

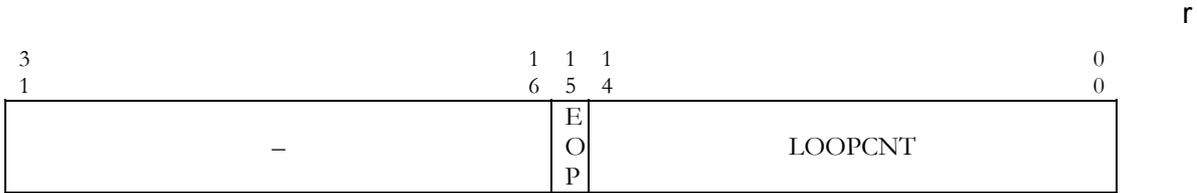


Name	Pos.	Contents
P3CNT	14:0	LOOPCNT field value of GIF_CNT register when transfer via PATH3 is interrupted

This register is accessible only in pause status by the GIF\_CTRL register.  
 The transfer restart via PATH3 is not reflected in the P3CNT. Refer to the following example.

	P3	P1 interrupt	P3	P1 interrupt	P3 →
LOOPCNT:	b a 9 8	x x x x x x x	7 6 5 4	x x x x x	3 2 1 0
P3CNT:	x x x x	8 8 8 8 8 8 8	8 8 8 8	4 4 4 4 4	4 4 4 4

## GIF\_P3TAG : PATH3 tag register



Name	Pos.	Contents
LOOPCNT	14:0	NLOOP field value of GIFtag read via PATH3 at the end
EOP	15	EOP flag vlaue of GIFtag read via PATH3 at the end

This register is accessible only in pause status by the GIF\_CTRL register.  
 The effective value is reflected only when PATH3 is transferring data in the IMAGE mode.

(This page is left blank intentionally)

## 8. IPU: Image Data Processor

---

The IPU (Image Processing Unit) is an image data processor with the ability to do MPEG2 macro block decoding, RGB conversion, vector quantization, and bit stream decompression. It does not have a special buffer, and shares the main memory with other processors by timesharing. That is, processing is performed in such a way that compressed data put in memory is decoded, decompressed, and written back again to main memory.

The decoded image is transferred to the GS and used as animation data and texture data.

## 8.1. IPU Overview

The IPU is an image data decompression processor, which decompresses compressed animations and texture images, and has the following features:

### Decoding of Standard Bit Stream

The IPU interprets and decodes the MPEG2 bit stream.

At this time, however, MC (Motion Compensation) is not performed by the IPU, but by EE Core multimedia instructions.

### Macro Block Decoding

The IPU decodes macro blocks converted into RGB from the bit stream, which composes the macro block layer of the MPEG2 I-PICTURE. These are used as parts of the image.

When the output is 16 bits, an ordered dither is applied. As a result, the Mach band for 16-bit output can be reduced.

### Vector Quantization

The IPU can convert an image decoded by direct color into an index color of 4 bits by carrying out vector quantization with a given CLUT (Color LookUp Table). This is used for the texture pattern the CLUT requires. It is also used when the texture area capacity of the frame buffer is limited.

### Transparency Control

The alpha plane (transparency plane) is generated from the decoded luminance value according to a fixed rule. This is useful in effectively cutting out the texture pattern when decoding the bit stream without the stencil pattern (transparent pixel mask pattern).

### 8.1.1. Basic Functions

The IPU has the following basic functions.

Function	Contents
MacroBlock decode	MPEG2 macro block layer decoding
MPEG2 decode	MPEG2 bit stream decoding
BitStream decode	Bit stream decompression

The basic decoding unit is a macro block (rectangular image of 16 x 16 pixels).

The IPU handles the data of the following formats.

Name	Contents	Width
BS128	MPEG2 bit stream subset	128 bits
RGB32	RGBA pixel (A8+R8+G8+B8)	32 bits
RGB16	RGBA pixel (A1+R5+G5+B5)	16 bits
RAW8	Unsigned 8-bit YCbCr pixel	8 bits
RAW16	Signed 16-bit YCbCr pixel (Only lower 9 bits are effective.)	16 bits
INDX4	Unsigned 4-bit index pixel	4 bits

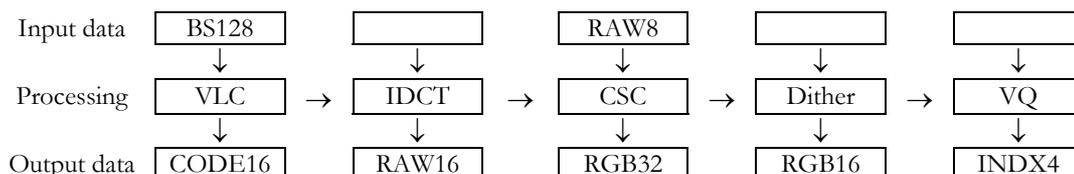
The following ten commands are available.

Code	Name	Contents	Input	Output
0000	BCLR	Input FIFO Initialization command	-	-
0001	IDEC	Intra decoding command	BS128	RGB32/RGB16
0010	BDEC	Block decoding command	BS128	RAW16
0011	VDEC	Variable-length data decoding command	BS128	Variable-length data + decoding code
0100	FDEC	Fixed-length data decoding command	BS128	Fixed-length data
0101	SETIQ	IQ table setting command	RAW8	-
0110	SETVQ	VQ table setting command	RGB16	-
0111	CSC	Color space conversion command	RAW8	RGB32/RGB16
1000	PACK	Format conversion command	RGB32	RGB16/INDX4
1001	SETTH	Threshold setting command	-	-

The IPU can perform the following post processing to the decoded macro block image.

Function	Contents
CSC (Color Space Conversion)	YCbCr -> RGB color conversion
Dither	4 x 4 ordered dither
VQ (Vector Quantization)	Vector quantization

The basic processing flow of the IPU is as illustrated below.



## 8.1.2. I/O and Execution of Command

### Data Input

Data is input to the IPU via DMA transfer from main memory or scratchpad memory to the IPU\_in\_FIFO. This DMA transfer uses the toIPU channel (ch-4).

### Command Execution

A command is issued to the IPU by writing the command code and the parameter to the IPU\_CMD register. While executing the command, the BUSY flag of the IPU\_CTRL register is set to 1.

### Data Output

Data from the IPU is output by means of DMA transfer from the IPU\_out\_FIFO to main memory or scratchpad memory. This DMA transfer uses the fromIPU channel (ch-3).

### Interrupt

When it becomes possible to accept the next command after the end of IPU processing, the BUSY flag of the IPU\_CTRL register is cleared to 0, and a request for the command end interrupt (INT\_IPU) is made to the INTC. If the INT\_IPU is not masked, the INT0 interrupt is asserted to the EE Core.

**Reset**

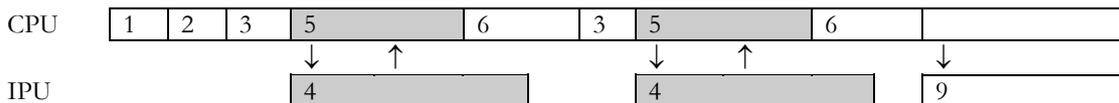
All commands can be ended forcibly by writing 1 to the RST bit of the IPU\_CTRL register. When the command is reset, each of the IPU\_CTRL.BUSY, IPU\_CTRL.IFC, and IPU\_CTRL.OFC bits are cleared to 0, and data being decoded inside the IPU is abandoned.

**8.1.3. Processing Example Using IPU**

A typical flow of the processing through the IPU is shown below.

- 1) The BCLR command is issued, and the input FIFO (IPU\_in\_FIFO) of the IPU is initialized. The toIPU channel (ch-4) of the DMAC is set, and the bit stream data in main memory is transferred to the IPU.
- 2) The FDEC command is issued, and the sequence layer, GOP layer, and picture layer are decoded. The IQM (Intra Quantizer Matrix) and the NIQM (Non Intra Quantizer Matrix) in the sequence layer are set in the quantization matrix table for the IPU by using the SETIQ command.
- 3) In the slice layer and the macro block layer, fixed-length symbols are decoded by the FDEC command, and variable-length symbols are decoded by the VDEC command.
- 4) After setting the fromIPU channel (ch-3) of the DMAC, the CBP in the macro block layer and the block layer are decoded by using the BDEC command. The output data of the BDEC command is transferred via DMA from the IPU output FIFO (IPU\_out\_FIFO) to the scratchpad memory (SPR).
- 5) The address of the reference image is calculated from the decoding result of 3), and the reference image data is transferred via DMA from main memory to the scratchpad memory over the toSPR channel (ch-9).
- 6) The decompression image data is obtained from the BDEC command output data of 4) and the reference image data of 5), and is transferred via DMA to main memory over the fromSPR channel (ch-8).
- 7) When checking IPU\_CTRL.SCD, if the next symbol is not a start code or is the SSC (Slice Start Code), the processing flow returns to 3).
- 8) If the next symbol is a start code other than SSC, after saving the status information of the IPU\_in\_FIFO so that the DMA transfer of the bit stream data can be restarted, the IPU\_in\_FIFO is initialized by the BCLR command. Next, by setting the toIPU channel, the frame data in main memory is DMA-transferred to the IPU\_in\_FIFO.
- 9) After the fromIPU channel is set, color space conversion is done by the CSC command. The output data from the CSC command is DMA-transferred from the IPU\_out\_FIFO to main memory.
- 10) The toIPU channel is set again by using the data saved in 8), transfer of the bit stream data in main memory is restarted, and the processing flow returns to 2).

The processing in 4), 5), and 6) above can be executed concurrently by double-buffering the SPR and pipelining EE Core processing and IPU processing.



## 8.2. Data Format

The IPU handles the following data formats:

Name	Contents	Width
BS128	MPEG2 bit stream subset	128 bits
RGB32	RGBA pixel (A8+R8+G8+B8) A8 Alpha (0/64/128) R8 Red color (0-255) G8 Green color (0-255) B8 Blue color (0-255)	32 bits
RGB16	RGBA pixel (A1+R5+G5+B5) A1 Alpha (A8=64? 1:0) R5 Red (R8>>3) G5 Green (G8>>3) B5 Blue (B8>>3)	16 bits
RAW8	YCbCr pixel (Unsigned 8-bit) Bias value 128 is added to Cb and Cr.	8 bits
RAW16	YCbCr pixel (Signed 16-bit) Only lower 9 bits are effective. Bias value 128 is added to Cb and Cr.	16 bits
INDX4	Unsigned 4-bit index color pixel	4 bits

The descriptions of the pixel format and the two-dimensional array format of 16 x 16 pixels are given to each of the data formats, as shown below.

### 8.2.1. BS128

MPEG2 bit stream.

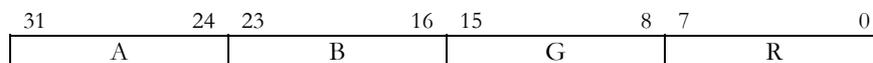
For decodable symbols in the bit stream, refer to "8.5. Bit Stream Symbols".



### 8.2.2. RGB32

Pixel data with RGBA of 8 bits each.

The data for 1 pixel is as shown in the following format.



The two-dimensional array of 16 x 16 pixels becomes 64 qwords in which pixels are arranged as follows. In this array, P<sub>yx</sub> indicates the pixel value at the (x, y) position.

127	96	95	64	63	32	31	0
P03	P02	P01	P00				
P07	P06	P05	P04				
P0B	P0A	P09	P08				
P0F	P0E	P0D	P0C				
P13	P12	P11	P10				
P17	P16	P15	P14				
.....							
PFF	PFE	PFD	PFC				

### 8.2.3. RGB16

Pixel data with 1-bit alpha value added to the RGB of 5 bits each.

Data format per pixel is as shown below.

15	14	10	9	5	4	0
A	B	G	R			

The two-dimensional array of 16 x 16 pixels becomes 32 qwords in which pixels are arranged as follows. In this array, the P<sub>yx</sub> shows the pixel value at (x, y) position.

127	112	96	80	64	48	32	16	0
P07	P06	P05	P04	P03	P02	P01	P00	
P0F	P0E	P0D	P0C	P0B	P0A	P09	P08	
P17	P16	P15	P14	P13	P12	P11	P10	
P1F	P1E	P1D	P1C	P1B	P1A	P19	P18	
.....								
PFF	PFE	PFD	PFC	PFB	PFA	PF9	PF8	

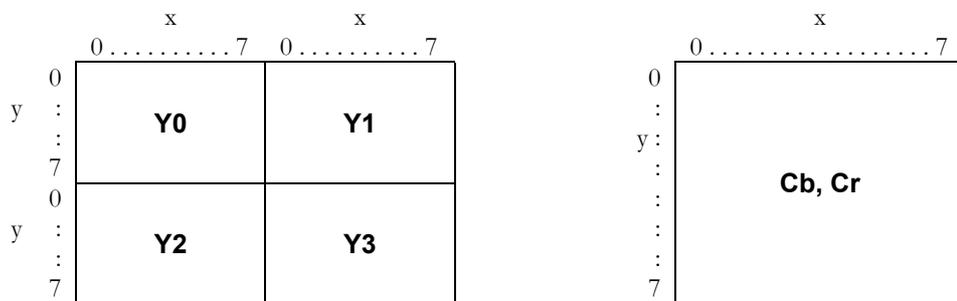
### 8.2.4. RAW8

YCbCr pixel data of unsigned 8 bits.

The two-dimensional array is shown as the following 24-qword data in which 8×8 Cb and Cr blocks follow Y block of 16×16 pixels. Y<sub>nxy</sub>, Cb<sub>nxy</sub>, and Cr<sub>nxy</sub> show the pixel value at (x, y) position in the Y<sub>n</sub>, Cb, and Cr blocks respectively.

127														0	
Y107	Y106	Y105	Y104	Y103	Y102	Y101	Y100	Y007	Y006	Y005	Y004	Y003	Y002	Y001	Y000
Y117	Y116	Y115	Y114	Y113	Y112	Y111	Y110	Y017	Y016	Y015	Y014	Y013	Y012	Y011	Y010
Y127	Y126	Y125	Y124	Y123	Y122	Y121	Y120	Y027	Y026	Y025	Y024	Y023	Y022	Y021	Y020
Y137	Y136	Y135	Y134	Y133	Y132	Y131	Y130	Y037	Y036	Y035	Y034	Y033	Y032	Y031	Y030
Y177	Y176	Y175	Y174	Y173	Y172	Y171	Y170	Y077	Y076	Y075	Y074	Y073	Y072	Y071	Y070
Y307	Y306	Y305	Y304	Y303	Y302	Y301	Y300	Y207	Y206	Y205	Y204	Y203	Y202	Y201	Y200
Y317	Y316	Y315	Y314	Y313	Y312	Y311	Y310	Y217	Y216	Y215	Y214	Y213	Y212	Y211	Y210
Y327	Y326	Y325	Y324	Y323	Y322	Y321	Y320	Y227	Y226	Y225	Y224	Y223	Y222	Y221	Y220
Y337	Y336	Y335	Y334	Y333	Y332	Y331	Y330	Y237	Y236	Y235	Y234	Y233	Y232	Y231	Y230
Y377	Y376	Y375	Y374	Y373	Y372	Y371	Y370	Y277	Y276	Y275	Y274	Y273	Y272	Y271	Y270
Cb17	Cb16	Cb15	Cb14	Cb13	Cb12	Cb11	Cb10	Cb07	Cb06	Cb05	Cb04	Cb03	Cb02	Cb01	Cb00
Cb37	Cb36	Cb35	Cb34	Cb33	Cb32	Cb31	Cb30	Cb27	Cb26	Cb25	Cb24	Cb23	Cb22	Cb21	Cb20
Cb57	Cb56	Cb55	Cb54	Cb53	Cb52	Cb51	Cb50	Cb47	Cb46	Cb45	Cb44	Cb43	Cb42	Cb41	Cb40
Cb77	Cb76	Cb75	Cb74	Cb73	Cb72	Cb71	Cb70	Cb67	Cb66	Cb65	Cb64	Cb63	Cb62	Cb61	Cb60
Cr17	Cr16	Cr15	Cr14	Cr13	Cr12	Cr11	Cr10	Cr07	Cr06	Cr05	Cr04	Cr03	Cr02	Cr01	Cr00
Cr37	Cr36	Cr35	Cr34	Cr33	Cr32	Cr31	Cr30	Cr27	Cr26	Cr25	Cr24	Cr23	Cr22	Cr21	Cr20
Cr57	Cr56	Cr55	Cr54	Cr53	Cr52	Cr51	Cr50	Cr47	Cr46	Cr45	Cr44	Cr43	Cr42	Cr41	Cr40
Cr77	Cr76	Cr75	Cr74	Cr73	Cr72	Cr71	Cr70	Cr67	Cr66	Cr65	Cr64	Cr63	Cr62	Cr61	Cr60

A block diagram of the Y<sub>n</sub> is as follows.



An example of defining the structure in C is shown as follows.

```
struct MACROBLOCK {
    unsigned char Y[16][16];      /* Luminance block */
    unsigned char Cb[8][8];      /* Chrominance block */
    unsigned char Cr[8][8];      /* Chrominance block */
};
```

### 8.2.5. RAW16

YCbCr pixel data of signed 16 bits.

The two-dimensional array is shown as the following 48-qword data in which 8×8 Cb and Cr blocks follow Y block of 16 x 16 pixels.

127	112	96	64	48	32	16	0
Y007	Y006	Y005	Y004	Y003	Y002	Y001	Y000
Y107	Y106	Y105	Y104	Y103	Y102	Y101	Y100
Y017	Y016	Y015	Y014	Y013	Y012	Y011	Y010
Y117	Y116	Y115	Y114	Y113	Y112	Y111	Y110
Y077	Y076	Y075	Y074	Y073	Y072	Y071	Y070
Y177	Y176	Y175	Y174	Y173	Y172	Y171	Y170
Y207	Y206	Y205	Y204	Y203	Y202	Y201	Y200
Y307	Y306	Y305	Y304	Y303	Y302	Y301	Y300
Y277	Y276	Y275	Y274	Y273	Y272	Y271	Y270
Y377	Y376	Y375	Y374	Y373	Y372	Y371	Y370
Cb07	Cb06	Cb05	Cb04	Cb03	Cb02	Cb01	Cb00
Cb17	Cb16	Cb15	Cb14	Cb13	Cb12	Cb11	Cb10
Cb77	Cb76	Cb75	Cb74	Cb73	Cb72	Cb71	Cb70
Cr07	Cr06	Cr05	Cr04	Cr03	Cr02	Cr01	Cr00
Cr17	Cr16	Cr15	Cr14	Cr13	Cr12	Cr11	Cr10
Cr77	Cr76	Cr75	Cr74	Cr73	Cr72	Cr71	Cr70

An example of defining the structure in C is shown as follows.

```
struct MACROBLOCK {
    short Y[16][16];      /* Luminance block */
    short Cb[8][8];      /* Chrominance block */
    short Cr[8][8];      /* Chrominance block */
};
```

### 8.2.6. INDX4

Pixel data of 4-bit index color. This is the index value in the CLUT (Color LookUp Table) set by the SETVQ command.

The two-dimensional array of 16 x 16 pixels becomes 8 qwords in which pixels are arranged as follows.

	124	120		68	64	60	56		4	0
I1F	I1E		I11	I10	I0F	I0E		I01	I00	
I3F	I3E		I31	I30	I2F	I2E		I21	I20	
I5F	I3E		I51	I50	I4F	I4E		I41	I40	
I7F	I7E		I71	I70	I6F	I6E		I61	I60	
IF7	IFE		IF1	IF0	IEF	IEE		IE1	IE0	

### 8.3. IPU Register Set

The IPU has the following registers.

Name	Width	r/w	Contents
IPU_CMD	64 bits	read	Decoded-code read register
		write	IPU command register
IPU_TOP	64 bits	read	Bit stream read register
IPU_CTRL	32 bits	read	IPU status register
		write	Control register
IPU_BP	32 bits	read	Bit position register

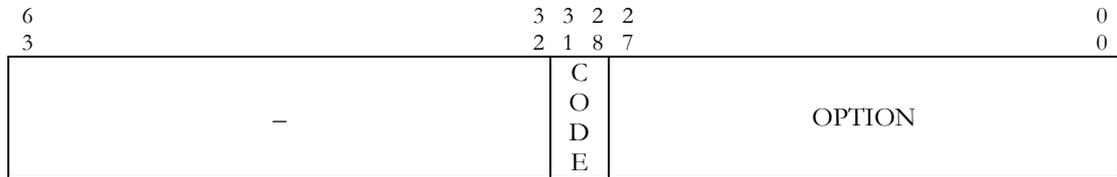
The IPU has the following two FIFOs for data I/O. Both of them are 8-qword size, mapped to main memory, and DMA-transferable.

Name	Width	r/w	Contents
IPU_in_FIFO	128 bits	write	Input FIFO to IPU
IPU_out_FIFO	128 bits	read	Output FIFO from IPU

## IPU\_CMD : Decoded-code read register / IPU command register

r / w

**write**

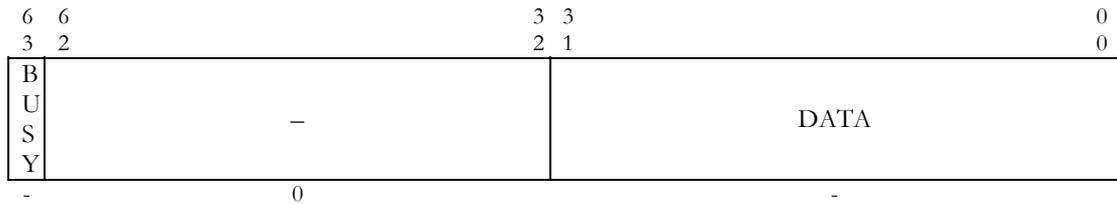


Name	Pos.	Contents
OPTION	27:0	Command option
CODE	31:28	Command code

When writing, only the lower 32 bits become effective as the command register.

The contents of the OPTION field differ depending on the command. For details of each command, refer to "8.4. IPU Commands".

**read**



Name	Pos.	Contents
DATA	31:0	VDEC/FDEC decoded value
BUSY	63	VDEC/FDEC command busy 0 DATA field enable 1 DATA field disable (VDEC/FDEC in execution)

When reading, all 64 bits become effective as the register to read the result of the VDEC/FDEC command.

When the BUSY bit is read to be 1, the decoding process is still being executed, therefore, the value that is read in the DATA field is meaningless.



**IPU\_CTRL : IPU status register / control register**

r / w

3	3	2	2	2	2	2	2	2	1	1	1	1	1	1	1	1	0	0	0	0	0	0	
1	0	9	7	6	4	3	2	1	0	9	8	7	6	5	4	3	8	7	4	3	2	1	0
B	R	-	PCT	M	Q	I	A	-	IDP	S	E	CBP	OFC	IFC									
U	S			P	S	V	S			C	C												
Y	T			1	T	F				D	D												
0	0		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	

Name	Pos.	r/w	Contents
IFC	3:0	r	Input FIFO counter Same as IPU_BP.IFC
OFC	7:4	r	Output FIFO counter
CBP	13:8	r	Coded block pattern
ECD	14	r	Error code detected 0 Not detected 1 Detected
SCD	15	r	Start code detected 0 Not detected 1 Detected
IDP	17:16	r/w	Intra DC precision 00 8 bits 01 9 bits 10 10 bits 11 Reserved
AS	20	r/w	Alternate scan 0 Zigzag scanning 1 Alternate scanning
IVF	21	r/w	Intra VLC format 0 MPEG1-compatible 2-dimensional VLC table 1 2-dimensional VLC table specially for intra macro block
QST	22	r/w	Q scale step 0 Linear step 1 Non-linear step
MP1	23	r/w	MPEG1 bit stream 0 MPEG2 bit stream 1 MPEG1 bit stream
PCT	26:24	r/w	Picture Type 000 Reserved 001 I-PICTURE 010 P-PICTURE 011 B-PICTURE 100 D-PICTURE
RST	30	w	Reset Reset when 1 is written
BUSY	31	r	Busy 0 Ready 1 Busy (command in execution)

The ECD/SCD bit is cleared to 0 when issuing a command and set to 1 when encountering the Error Code/Start Code in the bit stream.

When writing 1 to the RST bit, the command under execution is ended forcibly and the IPU is reset. Also, the remaining data being decoded in the IPU is abandoned.

## IPU\_BP : Bit position register

	1	1	1	1	1	0	0	0	0
	7	6	5	2	1	8	7	6	0
-	FP	-	IFC	-	BP				

Name	Pos.	r/w	Contents
BP	6:0	r	Bit stream pointer Bit position to start decoding within the first 128-bit data
IFC	11:8	r	Input FIFO counter No. of qwords in IPU_in_FIFO
FP	17:16	r	FIFO pointer No. of qwords remaining in IPU except IPU_in_FIFO

IPU\_BP is the register that maintains positional information on the bit stream data located in the IPU.

The IFC field shows the data size located in the IPU\_in\_FIFO in qword units. This value becomes the same as IPU\_CTRL.IFC.

The FP field shows in qword units size of data stored in the internal buffer except the IPU\_in\_FIFO among the data in the IPU.

The BP field shows the position of the first unprocessed bits in the qword data. Each of the commands that contain an FB field, such as IDEC, BDEC, VDEC, FDEC, and SETIQ, starts processing at the position where the first (IPU\_BP.BP + FB) bits are skipped in the first qword.

When each command ends, the bit position following the most recently processed data is left in IPU\_BP.BP.

Moreover, of the data left inside the IPU, the data size in the IPU\_in\_FIFO is left in the IPU\_BP.IFC and the remaining size is left in IPU\_BP.FP.

When IPU processing is suspended, the number of bits of data remaining in the IPU among the data transferred to the IPU can be calculated by the following formula:

$$(IPU\_BP.IFC \times 128 + IPU\_BP.FP \times 128 - IPU\_BP.BP)$$

When IPU processing is suspended and initialized with the BCLR command and processing is to be restarted from the same position, the requirements are:

Set D4\_MADR to the value of D4\_MADR (when processing was suspended) minus the (IPU\_BP.IFC+IPU\_BP.FP) qword, and issue the BCLR command once with an argument of the value of IPU\_BP.BP when processing was suspended.

## 8.4. IPU Commands

The IPU has the following ten commands:

Code	Name	Contents	Input	Output
0000	BCLR	Input FIFO initialization command	-	-
0001	IDEC	Intra decoding command	BS128	RGB32/RGB16
0010	BDEC	Block decoding command	BS128	RAW16
0011	VDEC	Variable-length data decoding command	BS128	Variable-length code + decoding code
0100	FDEC	Fixed-length data decoding command	BS128	Fixed-length data
0101	SETIQ	IQ table setting command	RAW8	-
0110	SETVQ	VQ table setting command	RGB16	-
0111	CSC	Color space conversion command	RAW8	RGB32/RGB16
1000	PACK	Format conversion command	RGB32	RGB16/INDX4
1001	SETTH	Threshold setting command	-	-

The following are descriptions of each command.

## BCLR : Input FIFO initialization command

3 1	2 2 8 7	0 0 7 6	0 0
0000	-	BP	

Name	Pos.	Contents
BP	6:0	Bit stream Point Bit position to start decoding within the first qword data

The BCLR command clears the IPU input FIFO (IPU\_in\_FIFO).

When clearing the bit stream data in the IPU\_in\_FIFO, D4\_CHCR.STR of the DMAC is cleared to 0 before issuing the BCLR command. By setting D4\_CHCR.STR to 1 after issuing the BCLR command, DMA transfer to the IPU\_in\_FIFO is started.

Of the newly transferred bit stream data, the first 1 qword becomes effective from the bit position specified by the BP field of the BCLR command.

## IDEC : Intra decoding command

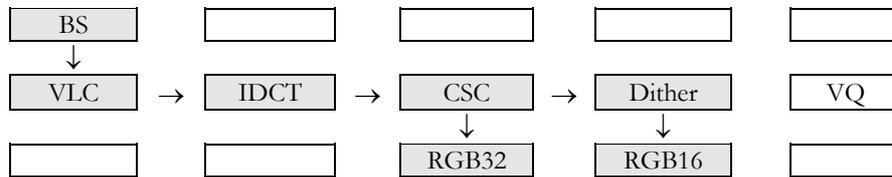
3 1	2 2 2 2 2 2 8 7 6 5 4 3	2 2 1 0	1 1 6 5	0 0 6 5	0 0
0001	O F M	D T E	S G N	D T D	- QSC - FB

Name	Pos.	Contents
FB	5:0	Forward Bit No. of bits to proceed with the bit stream before decoding (0-32)
QSC	20:16	Quantizer Step Code Quantization step code
DTD	24	DT Decode 0 Does not decode DT (DCT Type). 1 Decodes DT (DCT Type).
SGN	25	Pseudo Sign Offset 0 Outputs the post-decoding RGB value as it is. 1 Outputs the post-decoding RGB value by decreasing by 128. Underflow results in wraparound.
DTE	26	Dither Enable 0 No dithering 1 Dithering (effective only in RGB16)
OFM	27	Output Format 0 RGB32 1 RGB16

The IDEC command decodes the bit stream data for 1 slice after proceeding with the bit stream in the IPU\_in\_FIFO only for FB bits. The bit stream in the FIFO consumed by decoding proceeds automatically. For the symbols decoded by the IDEC command, refer to "8.5.1. IDEC Symbols". Bit strings that do not correspond to the symbols described in this section are processed as error codes.

The decoding result is output as macro block pixels in the RGB32 or RGB16 format via CSC (Color Space Conversion). In the RGB16 format, dithering can be added when the lower 3 bits of the luminance value are rounded down.

The processing flow is as follows.



The output data is read from the IPU\_out\_FIFO by the EE Core or the DMAC.

Decoding ends when a start code or error code is detected. The SCD or ECD flag of the IPU\_CTRL register is set according to the detected code, the BUSY flag is cleared to 0, and an interrupt request is made to the INTC. In addition, the bit stream position of the start code or error code is retained in the IPU\_BP register.

**Notes**

In the IDEC command, decoding of the MBT (Macroblock Type) symbol begins with the macro block at the beginning of the slice. Up to the MBE (Macroblock Escape)/MBAI (Macroblock Address Increment) symbol of the macro block at the beginning, decoding should be performed on the CPU side. The MBAI is decoded from the second macro block, but if the result is anything but 1, decoding is ended as an error code. When starting decoding, the DC prediction value is reset.

When the DTD (DT Decode) bit is set to 1, the DT (DCT type) symbol following the MBT symbol is decoded. When the decoded DT is 1, the Field DCT is enabled. When the decoded DT is not 1, the Frame DCT is enabled.

Decoding ends with the detection of a start code or error code. Therefore, to end decoding normally, the start code (32 bits) must be aligned on a byte boundary following the EOB of the last block. When supplying the bit stream to the IPU via DMAC, it is necessary to specify the transfer size including the start code. The internal pointer (after decoding) points to the beginning of the detected start code. Therefore, it is necessary to issue an FDEC or specify an appropriate value to the FB field by the following command to skip the pointed start code mentioned above.

## BDEC : Block decoding command

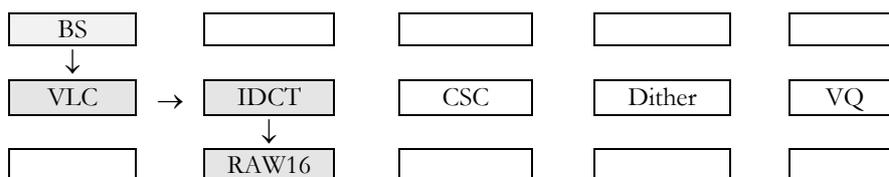
3 1	2 8	2 7	2 6	2 5	2 4	2 1	2 0	1 6	1 5	0 6	0 5	0 0
0010	M B I	D C R	D T	-	QSC	-	FB					

Name	Pos.	Contents
FB	5:0	Forward Bit No. of bits to proceed with the bit stream before decoding (0-32)
QSC	20:16	Quantizer Step Code Quantization step code
DT	25	DCT Type 0 Frame DCT 1 Field DCT
DCR	26	DC Reset 0 Does not reset DC prediction value. 1 Resets DC prediction value.
MBI	27	MacroBlock Intra 0 Non-intra macro block 1 Intra macro block

The BDEC command decodes 1 macro block after proceeding with the bit stream in the IPU\_in\_FIFO for FB bits only. Unlike the IDEC command, the BDEC command does not perform CSC (Color Space Conversion). For macro blocks that require MC (motion compensation), CSC should be performed anew after decoding by the BDEC command and MC-processing on the EE Core side.

The output data is arranged in RAW16 format, and the field DCT/frame DCT is processed appropriately by DT (DCT Type) bit setting.

The processing flow is illustrated below.



Output data is read from the IPU\_out\_FIFO by the EE Core or DMAC.

For symbols decoded by the BDEC command, refer to "8.5.2. BDEC Symbols". Bit strings not described in this section are processed as error codes.

When writing 0 to the MBI bit, only the blocks specified by the CBP (Coded Block Pattern) after the CBP symbol of the macro block are decoded. When writing 1 to the MBI bit, CBP is considered not to be existent, and all the 6 blocks composing the macro block are decoded.

Decoding ends under the following conditions:

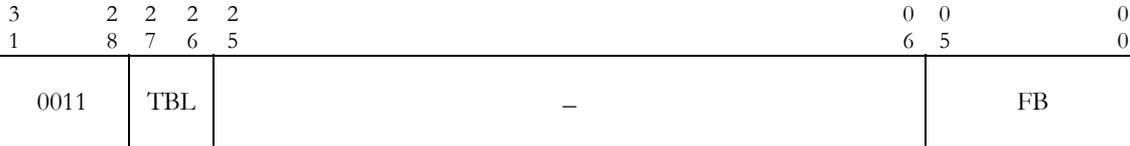
- a) When decoding of a macro block ended and more than 7 zero bits are detected after the EOB of the last block (when it is forecasted that the start code of the sequence layer, GOP layer, picture layer, or the slice layer follows).
- b) When decoding of a macro block ended and more than 7 zero bits are not detected after the EOB of the last block (when it is forecasted that the macro block follows).
- c) When an error code is detected while decoding 1 macro block.

When detecting the start code in a), the SCD flag of the IPU\_CTRL register is set. When detecting an error code in a) and in the case of c), the ECD flag of the IPU\_CTRL register is set. In both cases, an interrupt request is made to the INTC by clearing IPU\_CTRL.BUSY to 0 when decoding ends.

The bit stream positions of the beginning of the start code or the error code, of the bit next to the EOB (End Of Block) symbol, and of the beginning of the error code are left in the IPU\_BP register for a), b), and c) respectively.

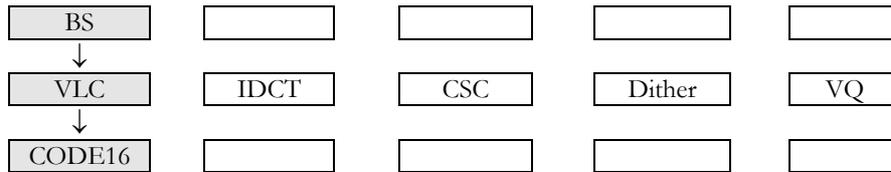
When setting 1 to the DCR bit, the BDEC command resets the DC prediction value at the start of decoding.

## VDEC : Variable-length data decoding command



Name	Pos.	Contents
FB	5:0	Forward Bit No. of bits to proceed with the bit stream before decoding (0-32)
TBL	27:26	VLC table 00 Macroblock Address Increment 01 Macroblock Type 10 Motion Code 11 DMVector

The VDEC command decodes the symbol specified in the TBL field after proceeding with the bit stream in the IPU\_in\_FIFO for FB bits. The processing flow is shown in the figure below.



The result of decoding is read from the IPU\_CMD register.

When the VDEC command is issued, IPU\_CMD.BUSY is set to 1. When decoding ends, the result is reflected in IPU\_CMD.DATA, and IPU\_CMD.BUSY is cleared to 0. The input bit stream automatically proceeds only for the length of the decoded sign, and the following bit stream position is left in the IPU\_BP register.

### Notes

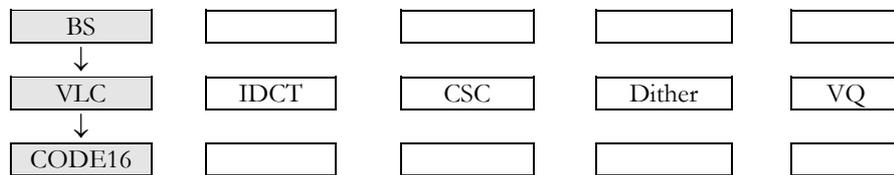
The VLC table is retrieved in 32-bit units in the VDEC command. Therefore, a code of 32 bits or more must be supplied to the IPU after the decoding start position specified with IPU\_BP.BP and IPU\_CMD.FB.

## FDEC : Fixed-length data decoding command

3 1	2 2 8 7	0 0 6 5	0 0
0100	-	FB	

Name	Pos.	Contents
FB	5:0	Forward Bit No. of bits to proceed with the bit stream before decoding (0-32)

The FDEC command returns the first 32 bits as is, as a decoding result after proceeding with the bit stream sent to the IPU\_in\_FIFO only for FB bits. The processing flow is as shown in the figure below.



The output data is read from the IPU\_CMD register.

When the FDEC command is issued, IPU\_CMD.BUSY is set to 1. When processing ends, the result is reflected in IPU\_CMD.DATA, and IPU\_CMD.BUSY is cleared to 0. The bit stream position at that time is left in the IPU\_BP register.

### Notes

A code of 32 bits or more must be supplied to the IPU after the decoding start position specified with IPU\_BP.BP and IPU\_CMD.FB.

## SETIQ : IQ table set command

3 1	2 8	2 7	2 6	0 6	0 5	0 0
0101	I Q M	-				FB

Name	Pos.	Contents
FB	5:0	Forward Bit No. of bits to proceed with the bit stream before decoding (0-32)
IQM	27	Intra IQ Matrix 0 Intra quantization matrix 1 Non-intra quantization matrix

After proceeding with the bit stream in the IPU\_in\_FIFO only for FB bits, the SETIQ command takes out 64 bytes and sets them in the specified IQ table. Therefore, the bit stream consumed in the FIFO proceeds automatically, and the following bit stream position is left in the IPU\_BP register.

---

## SETVQ : VQ table set command

---

3	2	2	0
1	8	7	0
0110	-		

The SETVQ command sets the CLUT data of 32 bytes (RGB16 format) written in the IPU\_in\_FIFO in the VQCLUT table.

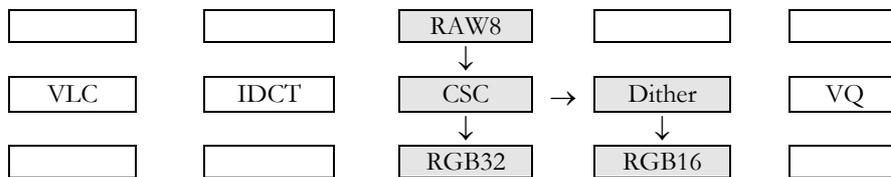
## CSC : Color space conversion command

3 1	2 8	2 7	2 6	2 5	1 1 0	1 0	0 0	
0111	O F M	D T E	-			MBC		

Name	Pos.	Contents
MBC	10:0	MacroBlock Count No. of macro blocks to be converted
DTE	26	Dither Enable 0 No dithering 1 Dithering (effective only in RGB16)
OFM	27	Output Format 0 RGB32 1 RGB16

The CSC command converts the macro block data (YCbCr) in the RAW8 format transferred to the IPU\_in\_FIFO into the RGB32/RGB16 format. When the output format is RGB16, the lower 3 bits of each RGB value are rounded down. At this time, the ordered dither can be added.

The processing flow is as shown in the figure below.



The number of macro blocks to be converted is specified in the MBC field.

The conversion result is read from the IPU\_out\_FIFO by the EE Core or the DMAC.

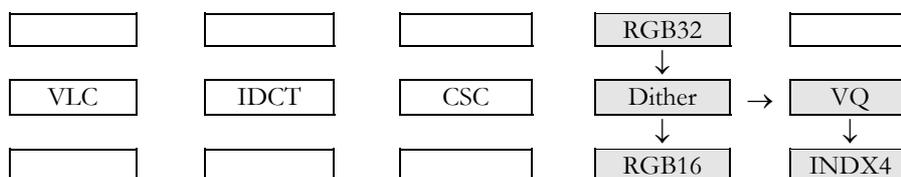
## PACK : Format conversion command

3 1	2 8	2 7	2 6	2 5	1 1	1 0	0 0
1000	O F M	D T E	-			MBC	

Name	Pos.	Contents
MBC	10:0	No. of macro blocks to be converted
DTE	26	Dither Enable 0 No dithering 1 Dithering
OFM	27	Output Format 0 INDX4 1 RGB16

The PACK command converts a macro block in RGB32 format transferred to the IPU\_in\_FIFO into the RGB16/INDX4 format.

In the RGB16 output format, the ordered dither can be added when the lower 3 bits are rounded down. In the INDX4 output format, vector quantization is performed with the CLUT set by the SETVQ command, and the corresponding index value is output. The processing flow is shown in the figure below.



The output result is read from the IPU\_out\_FIFO by the EE Core or the DMAC.

## SETTH : Threshold value setting

3 1	2 2 8 7	2 2 5 4	1 1 6 5	0 0 9 8	0 0
1001	-	TH1	-	TH0	

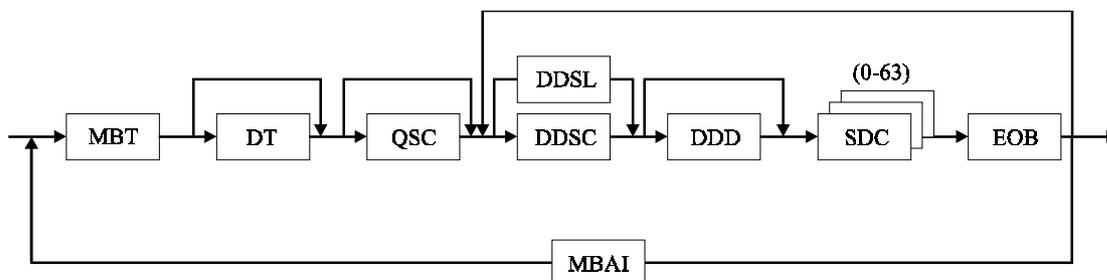
Name	Pos.	Contents
TH0	8:0	Transparent threshold value
TH1	24:16	Translucent threshold value

The SETTH command sets the threshold value to decide the transparent color by obtaining the alpha value when converting from YCbCr to RGBA. For details, refer to "8.6.1. CSC: Conversion from RAW8 to RGB32".

## 8.5. Bit Stream Symbols

### 8.5.1. IDEC Symbols

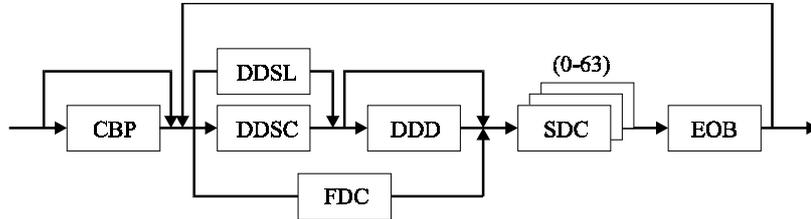
The following are the symbols of the bit streams to be decoded by the IDEC command.



Layer	Symbol	Name	Length	Note
MB	MBAI	Macroblock Address Increment	1	Exists in each block except the first macro block.
MB	MBT	Macroblock type	1-2	
MB	DT	DCT Type	1	Decodes when the DTD bit of the IDEC command is 1.
MB	QSC	Quantizer Step Code	5	Exists in the block whose MBT value is 01 (Intr + Q).
B	DDSL	DCT DC Size Luminance	2-9	Exists in Y block.
B	DDSC	DCT DC Size Chrominance	2-10	Exists in Cb/Cr block.
B	DDD	DCT DC Differential	1-11	Exists in the block whose DDSL/DDSC-specified size is other than 0.
B	SDC	Subsequent DCT Coefficients	3-24	Exists two or more SDCs (0-63) until EOB appears.
B	EOB	End Of Block	2 or 4	

### 8.5.2. BDEC Symbols

The following are the bit stream symbols to be decoded by the BDEC command.



Layer	Symbol	Name	Length	Note
MB	CBP	Coded Block Pattern	3-9	Exists in non-intra block.
B	DDSL	DCT DC Size Luminance	2-9	Exists in Y block in intra block.
B	DDSC	DCT DC Size Chrominance	2-10	Exists in Cb/Cr block in intra block
B	DDD	DCT DC Differential	1-11	Exists in intra block whose DDSL/DDSC-specified size is other than 0.
B	FDC	First DCT Coefficients	2-24	Exists in non-intra block.
B	SDC	Subsequent DCT Coefficients	3-24	Exists two or more SDCs (0-63) until EOB appears.
B	EOB	End Of Block	2or4	

### 8.5.3. VDEC VLC Table

The VLC tables used for decoding by the VDEC command are as follows.

#### Macroblock Address Increment (TBL=00)

MP1	DATA[15:0]	Decoded data(HEX)	Note
-	1-----	00010001	
-	011-----	00030002	
-	010-----	00030003	
-	0011-----	00040004	
-	0010-----	00040005	
-	00011-----	00050006	
-	00010-----	00050007	
-	0000111-----	00070008	
-	0000110-----	00070009	
-	00001011-----	0008000a	
-	00001010-----	0008000b	
-	00001001-----	0008000c	
-	00001000-----	0008000d	
-	00000111-----	0008000e	
-	00000110-----	0008000f	
-	0000010111-----	000a0010	
-	0000010110-----	000a0011	
-	0000010101-----	000a0012	
-	0000010100-----	000a0013	
-	0000010011-----	000a0014	
-	0000010010-----	000a0015	
-	00000100011-----	000b0016	
-	00000100010-----	000b0017	
-	00000100001-----	000b0018	
-	00000100000-----	000b0019	
-	00000011111-----	000b001a	
-	00000011110-----	000b001b	
-	00000011101-----	000b001c	
-	00000011100-----	000b001d	
-	00000011011-----	000b001e	
-	00000011010-----	000b001f	
-	00000011001-----	000b0020	
-	00000011000-----	000b0021	
1	00000001111-----	000b0022	macroblock_stuffing
-	00000001000-----	000b0023	macroblock_escape
-	00000010-----	00000000	ERROR
0	00000001111-----	00000000	ERROR
-	00000001110-----	00000000	ERROR

MP1	DATA[15:0]	Decoded data(HEX)	Note
-	0000000110-----	00000000	ERROR
-	0000000101-----	00000000	ERROR
-	00000001001-----	00000000	ERROR
-	00000000-----	00000000	ERROR

**Macroblock Type I-Picture (TBL=01, PCT=001)**

MP1	DATA[15:0]	Decoded data(HEX)	Note
-	1-----	00010001	Intra
-	01-----	00020011	Intra + Q
-	00-----	00000000	Error

**Macroblock Type P-Picture (TBL=01, PCT=010)**

MP1	DATA[15:0]	Decoded data(HEX)	Note
-	1-----	0001000a	MC Coded (f)
-	01-----	00020002	Non MC Coded
-	001-----	00030008	MC Not Coded (f)
-	00011-----	00050001	Intra
-	00010-----	0005001a	MC Coded + Q (f)
-	00001-----	00050012	Non MC Coded + Q (f)
-	000001-----	00060011	Intra + Q
-	000000-----	00000000	ERROR

**Macroblock Type B-Picture (TBL=01, PCT=011)**

MP1	DATA[15:0]	Decoded data(HEX)	Note
-	10-----	0002000c	MC Not Coded
-	11-----	0002000e	MC Coded
-	010-----	00030004	MC Not Coded (b)
-	011-----	00030006	MC Coded (b)
-	0010-----	00040008	MC Not Coded (f)
-	0011-----	0004000a	MC Coded (f)
-	00011-----	00050001	Intra
-	00010-----	0005001e	MC Coded+Q
-	000011-----	0006001a	MC Coded+Q (f)
-	000010-----	00060016	MC Coded+Q (b)
-	000001-----	00060011	Intra+Q
-	000000-----	00000000	ERROR

**Macroblock Type D-Picture (TBL=01, PCT=100)**

MP1	DATA[15:0]	Decoded data(HEX)	Notes
-	1-----	00010001	Intra
-	0-----	00000000	Error

**Motion Code (TBL=10)**

MP1	DATA[15:0]	Decoded data(HEX)	Notes
-	00000011001----	000bfff0	
-	00000011011----	000bfff1	
-	00000011101----	000bfff2	
-	00000011111----	000bfff3	
-	00000100001----	000bfff4	
-	00000100011----	000bfff5	
-	0000010011-----	000afff6	
-	0000010101-----	000afff7	
-	0000010111-----	000afff8	
-	00000111-----	0008fff9	
-	00001001-----	0008fffa	
-	00001011-----	0008fffb	
-	0000111-----	0007fffc	
-	00011-----	0005fffd	
-	0011-----	0004ffff	
-	011-----	0003ffff	
-	1-----	00010000	
-	010-----	00030001	
-	0010-----	00040002	
-	00010-----	00050003	
-	0000110-----	00070004	
-	00001010-----	00080005	
-	00001000-----	00080006	
-	00000110-----	00080007	
-	0000010110-----	000a0008	
-	0000010100-----	000a0009	
-	0000010010-----	000a000a	
-	00000011001----	000bfff0	
-	00000100010----	000b000b	
-	00000100000----	000b000c	
-	00000011110----	000b000d	
-	00000011100----	000b000e	
-	00000011010----	000b000f	
-	00000011000----	000b0010	
-	00000010-----	00000000	ERROR
-	00000000-----	00000000	ERROR

**DMVector (TBL=11)**

MP1	DATA[15:0]	Decoded data(HEX)	Notes
-	11-----	0002ffff	
-	0-----	00010000	
-	10-----	00020001	

## 8.6. Post Processing

The following are performed after macro block decoding. The selection is made as necessary depending on the command or by the command option setting.

### CSC (Color Space Conversion)

CSC converts YCbCr pixel data colors obtained from decoding to RGBA pixel data.

### Ordered Dither

The ordered dither rounds down the lower 3 bits of the 8-bit luminance value when converting from the RGB32 format to the RGB16 format. At this time, the Mach band is reduced by adding/subtracting the dither value corresponding to the pixel position.

### VQ (Vector Quantization)

VQ compares the RGB luminance value with the CLUT (Color LookUp Table) when converting to the IND4, and outputs the closest value.

The CLUT is a list of 16 luminance values, and can be set by the SETVQ command.

### Transparency Control

Transparency Control generates the alpha value based on the RGB value in obtaining the output of the RGBA format, because the alpha value of the pixel is not included in usual bit streams.

The threshold value used at this time can be set by the SETTH command.

The following are details of each post-processing function. Bit accuracy is expressed as shown below.

(sn.m) → Signed n bit. fraction part m bit included

(un.m) → Unsigned n bit. fraction part m bit included

Clipping processing/rounding processing is expressed as follows.

clip (x,min,max) Clipping data x to max or min

round (x,n) Rounding off bit n of data x

### 8.6.1. CSC: Conversion from RAW8 to RGB32

The details of processing to convert the data in RAW8 format obtained from macro block decoding to RGB32 format are as follows.

RAW8(Y(u8.0),Cb(u8.0),Cr(u8.0))

→ RGB32(A8(u8.0),R8(u8.0),G8(u8.0),B8(u8.0))

- 1) The bias (+128) applied to Cr and Cb is removed.

$$\text{Cr1}(s8.0) = \text{Cr}(u8.0) - 128$$

$$\text{Cb1}(s8.0) = \text{Cb}(u8.0) - 128$$

- 2) Cr1 and Cb1 are multiplied by the coefficient.

$$\text{Cr2}(s17.7) = 0x0cc(u9.7) \times \text{Cr1}(s8.0)$$

$$\text{Cr3}(s17.7) = 0x068(u9.7) \times \text{Cr1}(s8.0)$$

$$\text{Cb2}(s17.7) = 0x032(u9.7) \times \text{Cb1}(s8.0)$$

$$\text{Cb3}(s17.7) = 0x102(u9.7) \times \text{Cb1}(s8.0)$$

- 3) The lower 6 bits are rounded down.
 
$$\begin{aligned} \text{Cr4}(s11.1) &= \text{Cr2}(s17.7) \& \sim 0x003f \\ \text{Cr5}(s11.1) &= \text{Cr3}(s17.7) \& \sim 0x003f \\ \text{Cb4}(s11.1) &= \text{Cb2}(s17.7) \& \sim 0x003f \\ \text{Cb5}(s11.1) &= \text{Cb3}(s17.7) \& \sim 0x003f \end{aligned}$$
- 4) The bias (+16) applied to Y is removed.
 
$$\text{Y1}(u8.0) = \text{Y}(u8.0) - 16$$
- 5) Y1 is multiplied by the coefficient.
 
$$\text{Y2}(u17.7) = 0x095(u9.7) \times \text{Y1}(u8.0)$$
- 6) The lower 6 bits are rounded down.
 
$$\text{Y3}(u11.1) = \text{Y2}(u17.7) \& \sim 0x003f$$
- 7) Addition (subtraction) is done with Y3.
 
$$\begin{aligned} \text{r0}(s12.1) &= \text{Y3}(u11.1) + \text{Cr4}(s11.1) \\ \text{g0}(s12.1) &= \text{Y3}(u11.1) - \text{Cb4}(s11.1) - \text{Cr5}(s11.1) \\ \text{b0}(s12.1) &= \text{Y3}(u11.1) + \text{Cb5}(s11.1) \end{aligned}$$
- 8) The lower 1 bit is rounded off.
 
$$\begin{aligned} \text{r1}(s11.0) &= \text{round}(\text{r0}(s12.1), 0) \\ \text{g1}(s11.0) &= \text{round}(\text{g0}(s12.1), 0) \\ \text{b1}(s11.0) &= \text{round}(\text{b0}(s12.1), 0) \end{aligned}$$
- 9) Clipping is applied to the unsigned 8 bits.
 
$$\begin{aligned} \text{R8}(u8.0) &= \text{clip}(\text{r1}(s11.0), 0, 255) \\ \text{G8}(u8.0) &= \text{clip}(\text{g1}(s11.0), 0, 255) \\ \text{B8}(u8.0) &= \text{clip}(\text{b1}(s11.0), 0, 255) \end{aligned}$$
- 10) The alpha value is generated.
 
$$\begin{aligned} &\text{if } (\text{R8} < \text{th0} \&\& \text{G8} < \text{th0} \&\& \text{B8} < \text{th0}) \{ \\ &\quad \text{R8}(u8.0) = 0x00; \\ &\quad \text{G8}(u8.0) = 0x00; \\ &\quad \text{B8}(u8.0) = 0x00; \\ &\quad \text{A8}(u8.0) = 0x00; \\ &\} \text{ else if } (\text{R8} < \text{th1} \&\& \text{G8} < \text{th1} \&\& \text{B8} < \text{th1}) \{ \\ &\quad \text{A8}(u8.0) = 0x40; \\ &\} \text{ else } \{ \\ &\quad \text{A8}(u8.0) = 0x80; \\ &\} \end{aligned}$$

11) When the SGN bit is 1 in the IDEC command, MSB of each luminance value is reversed.

```

if (SGN) {
    R8(s8.0) = R8(u8.0) ^ 0x80;
    G8(s8.0) = G8(u8.0) ^ 0x80;
    B8(s8.0) = B8(u8.0) ^ 0x80;
}

```

### 8.6.2. Dithering: Conversion from RGB32 to RGB16

In conversion from the RGB32 format to the RGB16 format, the luminance value is rounded down following the dithering. When no dithering is applied, only the luminance value is rounded down.

When DTE=1 (Dither ON) and SGN=1 are set as options on the IDEC command, clipping is not performed correctly.

```

RGB32(A8(u8.0),R8(u8.0),G8(u8.0),B8(u8.0))
-> RGB16(A1(u1.0),R5(u5.0),G5(u5.0),B5(u5.0))

```

1) The coefficient of the dither matrix is added.

```

R10(s10.1) = R8(u8.0) + D(s4.1);
G10(s10.1) = G8(u8.0) + D(s4.1);
B10(s10.1) = B8(u8.0) + D(s4.1);

```

2) Clipping is applied to the unsigned 8 bits.

```

R8d(u8.0) = clip(R10(s10.1), 0, 255);
G8d(u8.0) = clip(G10(s10.1), 0, 255);
B8d(u8.0) = clip(B10(s10.1), 0, 255);

```

3) The lower 3 bits are rounded down.

```

R5(u5.0) = R8d(u8.0) >> 3;
G5(u5.0) = G8d(u8.0) >> 3;
B5(u5.0) = B8d(u8.0) >> 3;

```

4) A8 is converted to A1.

```

A1(u1.0) = (A8(u8.0) == 0x40) ? 1 : 0;

```

The coefficient D of the dither matrix are as follows. The selection is made according to the pixel position.

	x+0	x+1	x+2	x+3
y+0	-4.0	0.0	-3.0	1.0
y+1	2.0	-2.0	3.0	-1.0
y+2	-2.5	1.5	-3.5	0.5
y+3	3.5	-0.5	2.5	-1.5

### 8.6.3. VQ: Conversion from RGB16 to INDX4

The processing to convert the pixel data in the RGB16 format to the index color in the INDX4 format is performed as follows.

RGB16(A1(u1.0),R5(u5.0),G5(u5.0),B5(u5.0))  
→ INDX4(u4.0)

- 1) The absolute value of the difference between the RGB value and each CLUT element is calculated.

$$\begin{aligned}dr(u5.0) &= | R5(u5.0) - CLUTr[i](u5.0) | \\dg(u5.0) &= | G5(u5.0) - CLUTg[i](u5.0) | \\db(u5.0) &= | B5(u5.0) - CLUTb[i](u5.0) |\end{aligned}$$

- 2) The square sum of dr, dg, and db is obtained.

$$\begin{aligned}d1(u12.0) &= dr(u5.0) * dr(u5.0) \\ &+ dg(u5.0) * dg(u5.0) \\ &+ db(u5.0) * db(u5.0)\end{aligned}$$

- 3) 1) and 2) are repeated from  $i=0$  to 15. When  $d1(u12.0)$  becomes minimum,  $i$  becomes the output value (u4.0).

## **9. Appendix**

---

## 9.1. Data Flow Model

This section includes examples of typical data flow inside the EE, which provide a better understanding of the EE process.

### 9.1.1. Graphics Data Flow

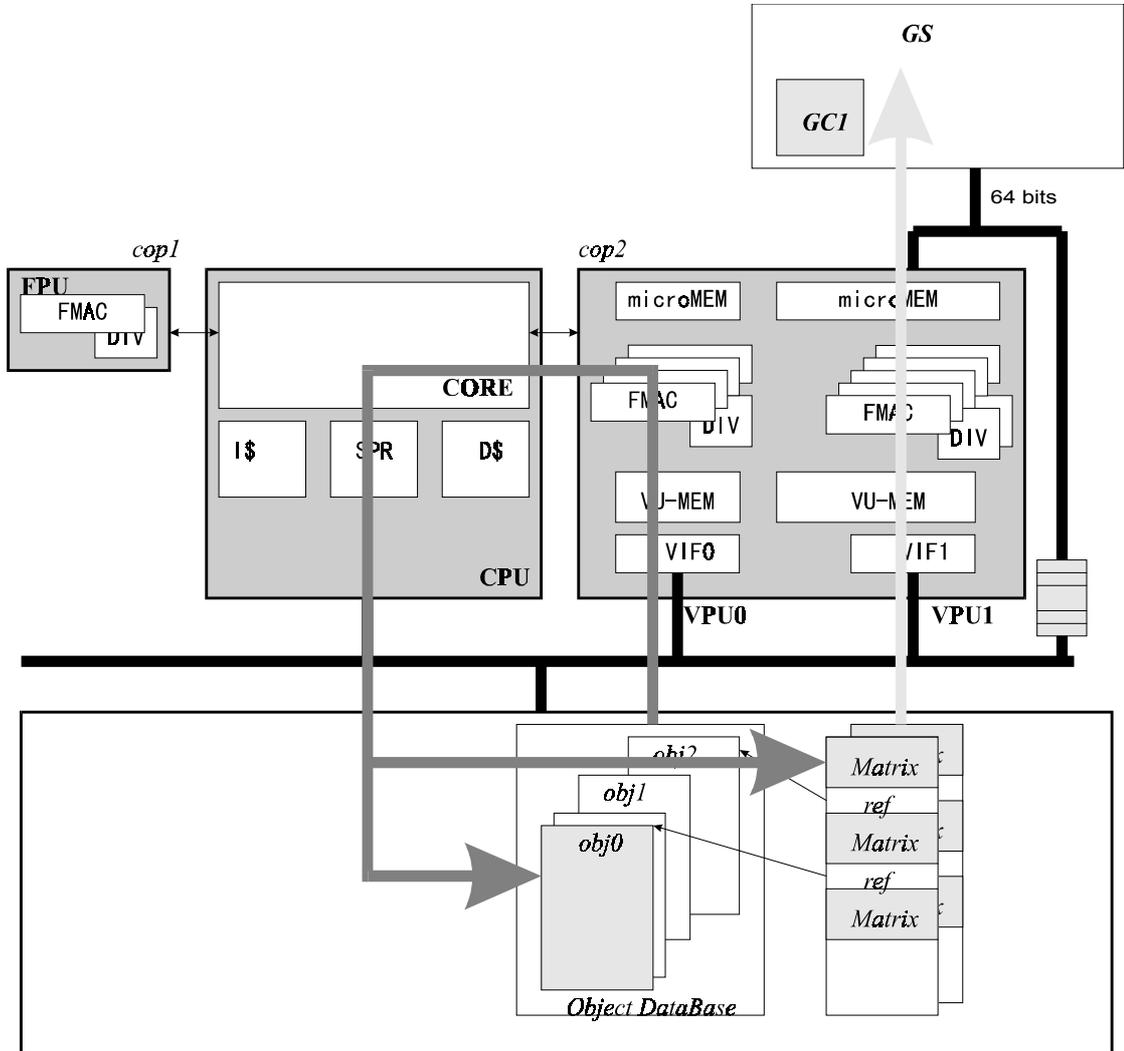
The basic idea in the Graphics data flow is to separate geometry processing into patterned processing (Simple Stream) and non-patterned processing (Complex Stream). Each of them generates display lists of its own context to be transferred to the Graphics Synthesizer for drawing.

Type	Contents
Patterned processing	Images that can be generated by control point and matrix operations VPU programmable processing Perspective conversion, parallel light source calculation, creation of secondary curved surface, etc.
Non-patterned processing	Images involving complex polygon operation Complex processing in which CPU intervention is essential Simulation of deductive reasoning or physical phenomena.

Generally, patterned processing is performed by VPU1, which is rather inflexible but efficient and high-performance, while non-patterned processing is performed by the EE Core, which is more flexible, with VPU0 used as a coprocessor.

### 9.1.2. Graphics Data Flow Example (1)

This is an example of macro serial connection of VPU0 and VPU1. In this example, the EE Core and VPU0 generate control points and matrices of the object. VPU1 then generates a display list based on them.



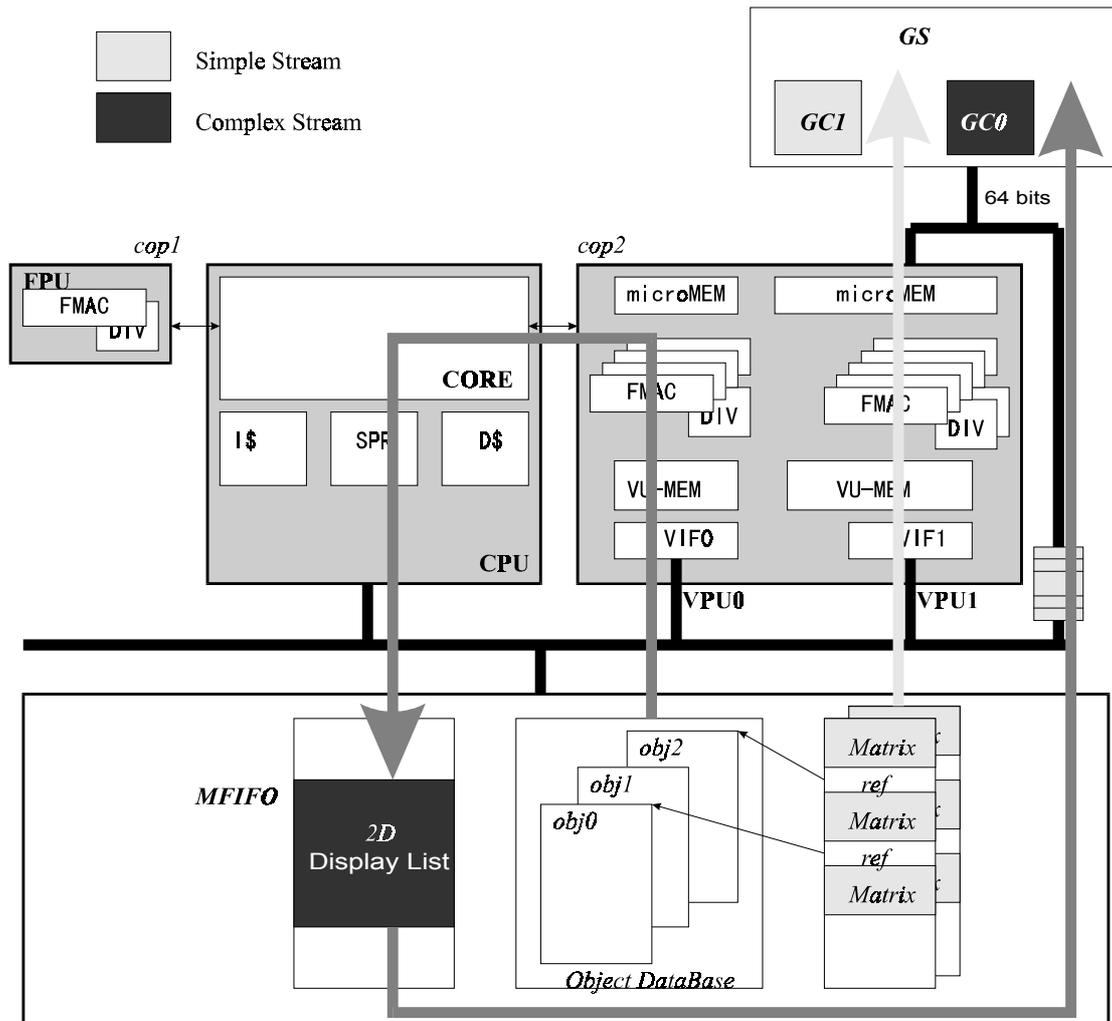
DMA is used as described in the table below.

Channel	Mode
fromSPR	Normal
VIF1	Source Chain

DMA Stall
fromSPR -(Stall Control) → VIF1

### 9.1.3. Graphics Data Flow Example (2)

This is an example of macro parallel connection of VPU0 and VPU1. In this example, while the EE Core and VPU0 generate a display list involving complex processing, VPU1 generates a patterned display list. Meanwhile, the EE Core and VPU0 generate the matrix for VPU1. The stream on the CPU side is buffered to memory (MFIFO), because of its low priority.



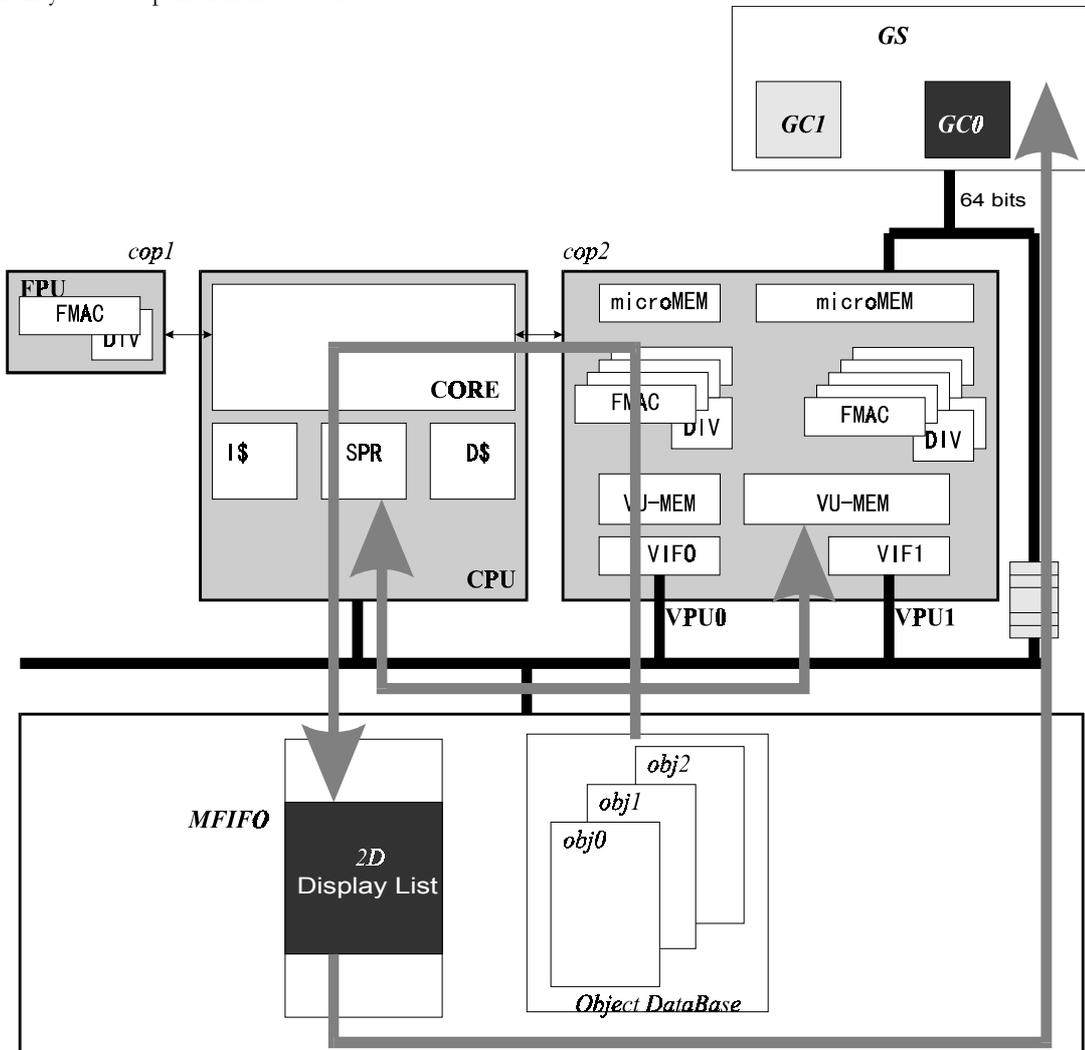
DMA is used as described in the table below.

Channel	Mode
fromSPR	Normal
VIF0	Source Chain
VIF1	Source Chain
GIF	Source Chain

DMA Stall
fromSPR -(MFIFO) → GIF

### 9.1.4. Graphics Data Flow Example (3)

The following illustrates the flow of the VPU0 and VPU1 operation, which is similar to the operation of a coprocessor. In this simple flow, all the display lists run through the EE Core and the main bus with high flexibility but low performance level.



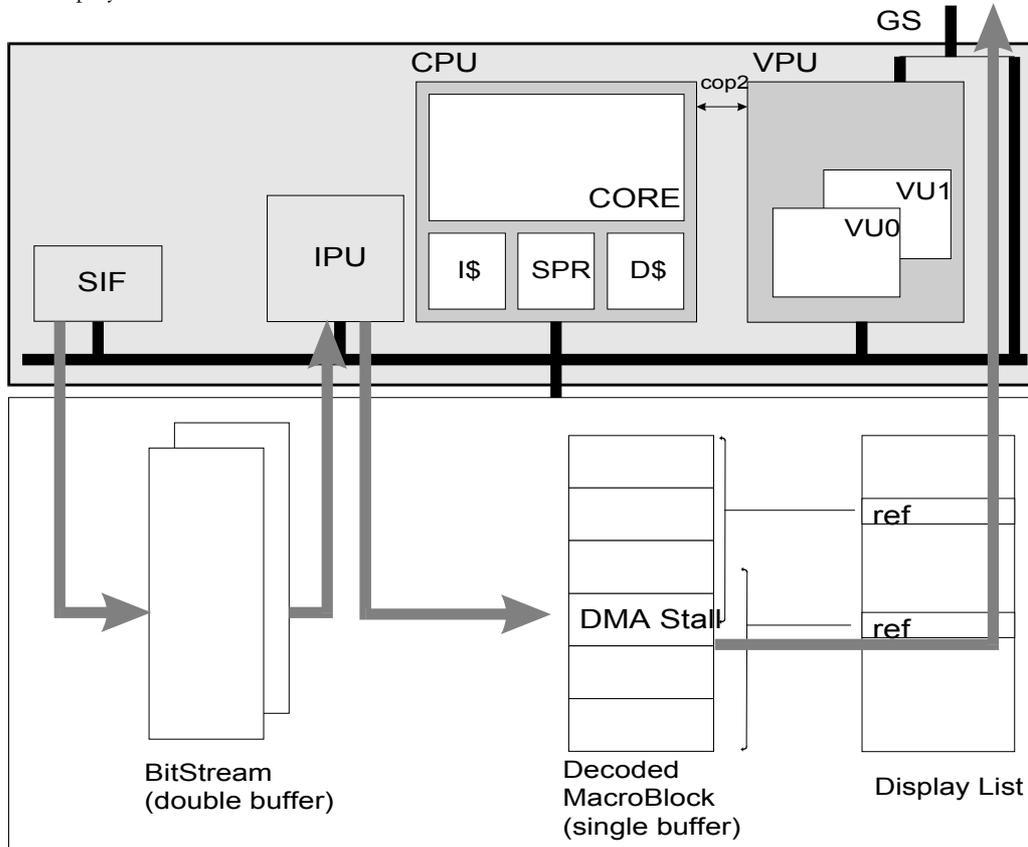
DMA is used as described in the table below.

Channel	Mode
fromSPR	Normal
VIF0	Source Chain
VIF1	Source Chain
GIF	Source Chain

DMA Stall
fromSPR -(MFIFO) → GIF

### 9.1.5. Image Data Flow Example (1)

In this example, the texture data obtained by decompressing the MPEG2 bit stream is transferred as a part of the display list.



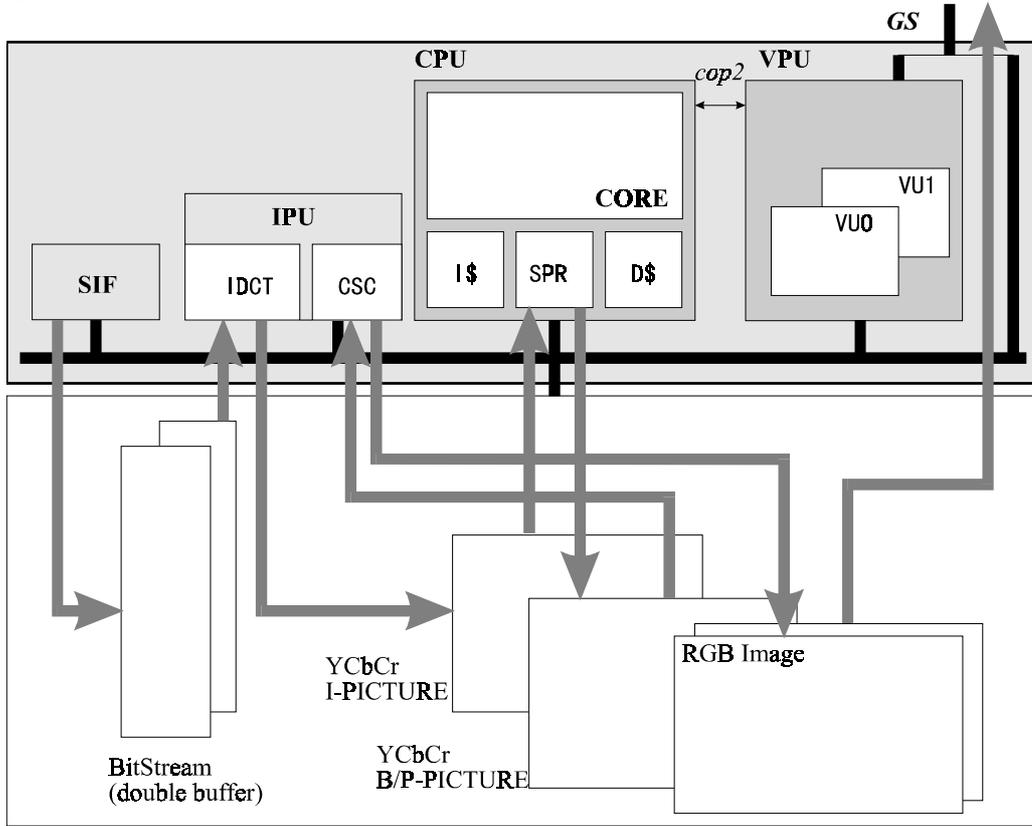
DMA is used as described in the table below.

Channel	Mode
SIF0	Normal/Destination Chain
toIPU	Normal
fromIPU	Normal
GIF	Source Chain

DMA Stall
fromIPU -(Stall Control)→ GIF

### 9.1.6. Image Data Flow Example (2)

This is an example of decoding an image when motion compensation (MC) is done. Since MC capability is not provided in the IPU, MC processing is done in the EE Core after a macro block is decoded, then CSC (color space conversion) is done by IPU.



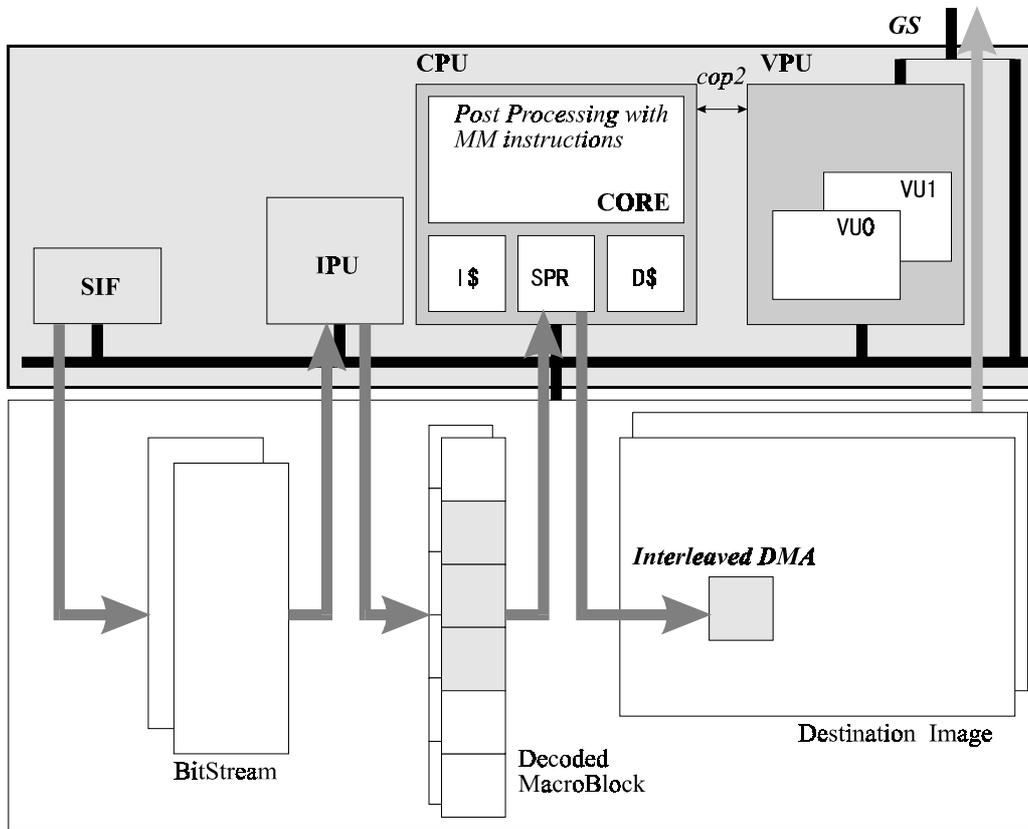
DMA is used as described in the table below.

Channel	Mode
SIF0	Normal/Destination Chain
toIPU	Normal
fromIPU	Normal
toSPR	Normal/Interleave
fromSPR	Normal
GIF	Normal/Source Chain

DMA Stall
(none)

### 9.1.7. Image Data Flow Example (3)

In this example, an image is created using the EE Core multimedia instructions, based on the image data decompressed via the IPU, in similar manner as creating an image with a PC application.



DMA is used as described in the table below.

Channel	Mode
SIF0	Normal/Destination Chain
toIPU	Normal
fromIPU	Normal
toSPR	Normal/Interleave
fromSPR	Normal/Interleave
GIF	Normal/Source Chain

DMA Stall
(none)

### 9.1.8. Data Transfer to VPU1

The structure of the display list transferred to VPU1 is shown in this example.

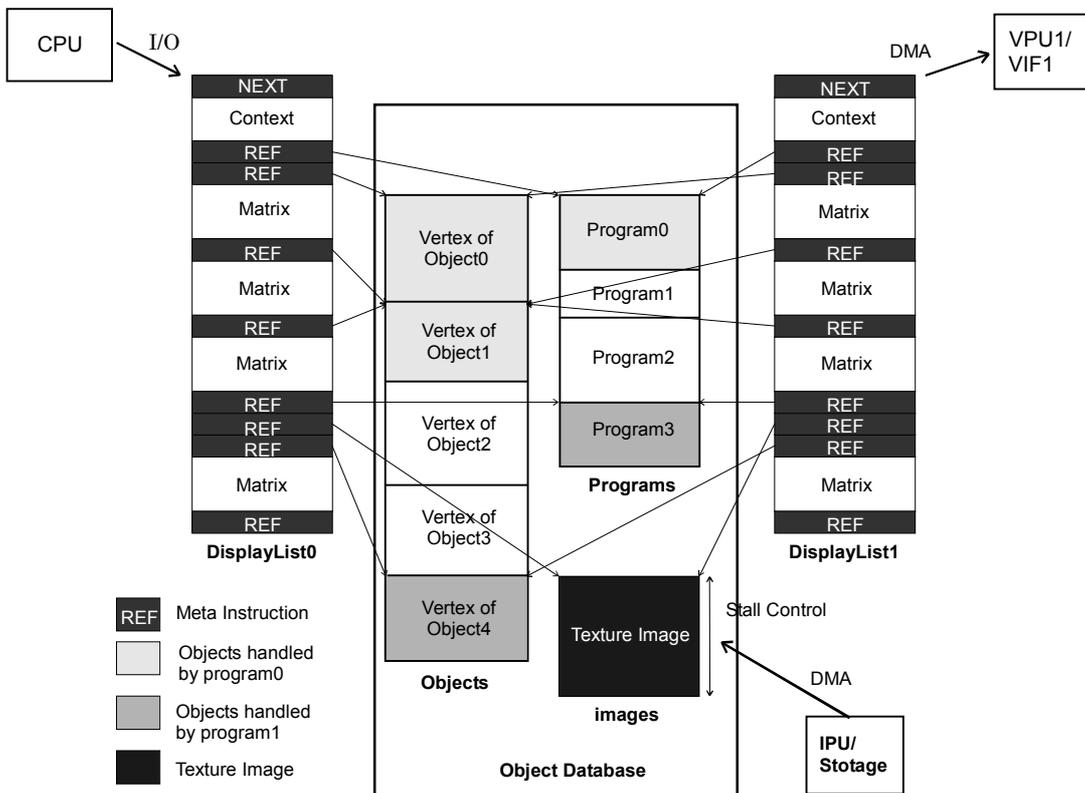
The display list is generated using a double buffer. While transferring the previously generated Display List 0 to VPU1, the EE Core generates Display List 1 for the next frame.

In this case, the data common to each frame, such as texture data, are referenced from the display list using tag instructions, and shared among the display lists. Generating the display list in this example only requires updating the data (e.g. matrix) that changes from frame to frame.

When the program that interprets the object data is not resident in VPU1, the program is transferred before transferring the object data. The transfer program specification is also performed by the tag instructions.

Moreover, when the texture image data is not resident in the frame buffer, it is transferred before transferring the object data. If the image data are decompressed data from the IPU or transferred data from the SBUS and change from frame to frame, they are synchronized by the stall control function of DMA.

Since VPU1 processing is temporarily suspended during image data transfer, other DMA channels are stopped to minimize this period. This is also specified with the tag instruction control bit.



(This page is left blank intentionally)