

# NITRO Programming Manual

Version 1.56



# CONFIDENTIAL

## Confidential

---

---

These coded instructions, statements, and computer programs contain proprietary information of Nintendo of America Inc. and/or Nintendo Company Ltd., and are protected by Federal copyright law. They may not be disclosed to third parties or copied or duplicated in any form, in whole or in part, without the prior written consent of Nintendo.

---

---



## Table of Contents

1	System.....	1
1.1	System Outline .....	1
1.1.1	NITRO Processor.....	2
1.1.2	Main Memory .....	3
1.1.3	LCD.....	4
1.1.4	Digital Keys .....	4
1.1.5	Touch Screen.....	4
1.1.6	Microphone .....	4
1.1.7	RTC.....	4
1.1.8	Wireless Communications.....	5
1.1.9	Nintendo DS Game Card .....	5
1.1.10	DS Accessories.....	5
1.2	Memory Map.....	6
1.3	Accessing Devices Connected to the Subprocessor .....	8
1.4	Startup Mode .....	8
1.4.1	NITRO Mode.....	8
1.4.2	AGB Compatibility Mode.....	8
1.5	Destination.....	8
2	Memory.....	9
2.1	External Memory.....	11
2.1.1	Main Memory .....	13
2.2	NITRO Processor's Internal Memory .....	17
2.2.1	VRAM.....	17
2.2.2	Work RAM.....	29
2.2.3	I/O Registers .....	31
2.3	Memory Map for Game Card Boot.....	32
3	Main Processor Core (ARM946E-S) .....	35
3.1	Protection Unit .....	35
3.2	Tightly Coupled Memory (TCM).....	36
3.2.1	Instruction TCM.....	36
3.2.2	Data TCM.....	36
3.3	Cache Memory .....	36
3.3.1	Instruction Cache .....	37
3.3.2	Data Cache .....	38
3.3.3	Cache Operations .....	40
3.3.4	Optimizing the Cache.....	41
3.4	Write Buffer.....	42
3.4.1	Write Buffer Operations .....	43
3.5	Ensuring Coherency .....	43
3.5.1	Write-Back Mode .....	43
3.5.2	Write-Through Mode .....	44
4	Display.....	47
4.1	Display System .....	47
4.2	LCD.....	49
4.2.1	LCD Controller Specifications .....	49
4.3	Display Status.....	51
4.4	Display Control .....	54
4.4.1	Top LCD/Bottom LCD Output Switching.....	54
4.4.2	Display Control of 2D Graphics Engine A .....	55
4.4.3	2D Graphics Engine B Display Controls .....	57
4.4.4	Display Modes .....	58
4.5	Display Capture .....	67

4.6	Master Brightness .....	71
5	2D Graphics.....	73
5.1	Controlling the 2D Display .....	73
5.2	BG.....	77
5.2.1	BG Mode .....	77
5.2.2	BG Control .....	81
5.2.3	Character BG .....	88
5.2.4	Bitmap BG .....	103
5.2.5	BG Scroll.....	105
5.2.6	BG Rotation and Scaling (Affine Transformation) .....	106
5.3	OBJ.....	109
5.3.1	OBJ Display Control.....	111
5.3.2	OAM .....	112
5.3.3	Character OBJ .....	121
5.3.4	Bitmap OBJ .....	128
5.4	Backdrop.....	135
5.5	Color Palettes .....	136
5.5.1	Standard Palettes .....	136
5.5.2	Extended Palettes .....	137
5.6	Windows .....	142
5.6.1	Precedence of Windows .....	145
5.7	Color Special Effects .....	146
5.8	Mosaic .....	150
5.9	Display Priority .....	151
6	3D Graphics.....	153
6.1	3D Display Control.....	155
6.2	Geometry Engine .....	158
6.2.1	Overview .....	158
6.2.2	Coordinate System .....	158
6.2.3	Coordinate Transformations.....	159
6.2.4	Projection Transformations .....	161
6.2.5	Depth Buffering .....	163
6.2.6	Geometry Commands.....	167
6.2.7	Swapping the Rendering Engine's Reference Data.....	178
6.2.8	Viewport .....	180
6.2.9	Matrices .....	181
6.2.10	Light .....	189
6.2.11	Material .....	190
6.2.12	Polygon Attributes .....	196
6.2.13	Polygons .....	200
6.2.14	Texture Mapping .....	206
6.2.15	Tests .....	216
6.2.16	Status .....	220
6.2.17	Warnings Regarding Calculation Precision.....	225



6.3	Rendering Engine .....	226
6.3.1	Overview .....	226
6.3.2	Rendering Methods.....	228
6.3.3	Initializing the Rendering Buffers .....	231
6.3.4	Rasterizing .....	235
6.3.5	Textures .....	244
6.3.6	Alpha-Test.....	258
6.3.7	Alpha-Blending.....	258
6.3.8	Edge Marking.....	259
6.3.9	Fog Blending .....	260
6.3.10	Anti-aliasing .....	264
6.3.11	Status .....	267
6.4	2D Graphics Features you can Apply to the 3D Screen after Rendering .....	268
6.4.1	Raster Scroll .....	268
6.4.2	Order of Display Priority with a 2D Screen.....	268
6.4.3	Windows .....	269
6.4.4	Color Effects .....	270
7	DMA.....	273
8	Timer .....	279
9	Interrupts .....	281
9.1	Interrupt Master Enable Register .....	281
9.2	Interrupt Enable Register.....	282
9.3	Interrupt Request Register.....	283
9.4	Interrupt Cautions .....	284
9.4.1	Clearing IME and IE .....	284
9.4.2	Multiple Interrupts .....	284
9.4.3	Interrupt Delays During DMA Operation .....	284
9.4.4	Interrupts from ARM7.....	284
10	Power Management .....	285
10.1	Sleep Mode.....	285
10.2	Controlling Various Power Supplies .....	286
10.2.1	Sound.....	286
10.2.2	LCD Backlight .....	286
10.2.3	LCD.....	286
10.2.4	Microphone .....	286
10.2.5	System .....	286
10.2.6	Graphics.....	287
10.3	Power Status .....	290
10.3.1	Low Battery State.....	290
10.3.2	DS Open/Closed State.....	290
11	Accelerators.....	291
11.1	Divider.....	291
11.1.1	Number of Calculation Cycles.....	293
11.2	Square-Root Unit.....	294
11.2.1	Number of Calculation Cycles.....	295
12	Keys.....	297
12.1	Input Keys.....	297
12.2	Interrupt Handling for Key Input.....	298
13	Sound .....	299

13.1	Hardware Specifications .....	300
13.1.1	Data Format .....	300
13.1.2	Channels .....	302
13.1.3	Mixer .....	302
13.1.4	Master Volume .....	303
13.1.5	Sound Capture .....	303
13.1.6	Power Control .....	303
13.1.7	Cautions .....	303
13.2	Sound Block Diagrams .....	304
13.2.1	Overall Sound .....	304
13.2.2	Channels 0 – 3 and Sound Capture 0 - 1 .....	305
13.2.3	Channels 4 - 7 .....	307
13.2.4	Channels 8 - 15 .....	308
13.2.5	Examples of Using Sound .....	309
13.3	NITRO-Composer .....	312
13.3.1	NITRO-Composer Playback Method .....	312
14	Wireless Communications .....	313
14.1	Hardware Specifications .....	313
14.2	Wireless Manager .....	313
14.2.1	Internet Play .....	313
14.2.2	Multi-Card Play .....	314
14.2.3	Single-Card Play .....	314
15	Touch Panel .....	315
15.1	Touch Panel Structure .....	316
16	Microphone .....	319
17	Real-Time Clock (RTC) .....	321
18	Internal Flash Memory .....	323
18.1	Touch Panel Calibration Data .....	323
18.2	Owner Information Data .....	323
18.3	NITRO Initial Setting Data .....	324
18.4	RTC Operation Information Data .....	324
Appendix A	Register List .....	325
A.1	Addresses 0x04000000 and Higher .....	325
A.2	Addresses 0x04001000 and higher (2D Graphics Engine B-related) .....	362
A.3	Addresses 0x04100000 and higher .....	365
Appendix B	List of VRAM Data Capacities .....	369
Appendix C	Data Formats .....	371

## Figures

Figure 1-1 : Overall System Block Diagram .....	1
Figure 1-2 : ARM9 and ARM7 Overall Memory Maps .....	7
Figure 2-1 : Transfer Sequence from Main Memory to Work RAM (Basic Cycles).....	14
Figure 2-2 : Transfer Sequence from Main Memory to Work RAM (Worst Case) .....	14
Figure 2-3 : Transfer Sequence from Work RAM to Main Memory (Basic Cycles).....	15
Figure 2-4 : Transfer Sequence from Main Memory to VRAM (Basic Cycles).....	15
Figure 2-5 : Transfer Sequence from Main Memory to VRAM (Worst Case) .....	16
Figure 2-6 : Transfer Sequence from VRAM to Main Memory (Basic Cycles).....	16
Figure 2-7 : Texture Image Slot Memory Map .....	22
Figure 2-8 : Texture Palette Slot Memory Map .....	25
Figure 2-9 : BG Extended Palette Slot Memory Map .....	25
Figure 2-10 : OBJ Extended Palette Slot Memory Map .....	26
Figure 2-11 : Memory Maps for Various Settings of ARM9, ARM7 Shared Internal Work RAM .....	30
Figure 2-12 : Memory Map for Game Card Boot .....	32
Figure 3-1 : Block Diagram of the Main Processor Core .....	35
Figure 3-2 : Structure and Actions of the Instruction Cache .....	38
Figure 3-3 : Structure and Actions of the Data Cache .....	39
Figure 3-4 : Cache Line State Transitions (Write-Back Mode) .....	44
Figure 3-5 : Cache Line State Transitions (Write-Through Mode) .....	45
Figure 4-1 : Display System Block Diagram .....	48
Figure 4-2 : LCD Scan Timing .....	49
Figure 4-3 : Display Mode Selection (Display Output A Side Only) .....	59
Figure 4-4 : Display Mode Selection (Display Output A Side Only) .....	60
Figure 4-5 : Example of Displaying the Bitmap OBJ Results of 3D Rendering.....	62
Figure 4-6 : VRAM Address Map of the LCD Pixels .....	63
Figure 4-7 : Example of the Motion Blur Effect that Uses the Display Capture .....	64
Figure 4-8 : LCD Pixel EVEN/ODD Map of the Main Memory Display FIFO Register.....	66
Figure 4-9 : LCD Pixel Map of the Capture Data (When the Capture Size is 256 x 192 Dots).....	69
Figure 5-1 : Out-of-Area Processing Method Differences .....	84
Figure 5-2 : Text BG Screen Size .....	86
Figure 5-3 : Affine BG Screen Size .....	87
Figure 5-4 : VRAM Offset for BG Character Data.....	89
Figure 5-5 : VRAM Offset for BG Screen Data .....	90
Figure 5-6 : 256x256-Dot Address Mapping (Text BG).....	92
Figure 5-7 : 256x512-Dot Address Mapping (Text BG).....	92
Figure 5-8 : 512x256-Dot Address Mapping (Text BG).....	93
Figure 5-9 : 512x512-Dot Address Mapping (Text BG).....	93
Figure 5-10 : Character Data Address Mapping (Text BG 16-Color Mode).....	94
Figure 5-11 : Character Data Address Mapping (Text BG 256-Color Mode).....	95
Figure 5-12 : 128X128-Dot Address Mapping (Affine BG) .....	97
Figure 5-13 : 256x256-Dot Address Mapping (Affine BG) .....	97
Figure 5-14 : 512x512-Dot Address Mapping (Affine BG) .....	98
Figure 5-15 : 1024x1024-Dot Address Mapping (Affine BG) .....	99
Figure 5-16 : Character Data Address Mapping (Affine BG) .....	100
Figure 5-17 : Character Data Address Mapping (256-Color x 16-Palette Character BG).....	102
Figure 5-18 : Offset Schematic .....	105
Figure 5-19 : BG Rotation and Scaling .....	106
Figure 5-20 : OAM Memory Map (Add 0x400h to 2D Graphics Engine B Addresses) .....	112
Figure 5-21 : Affine Transformation of Double-Size OBJ Field .....	114
Figure 5-22 : Problem of OBJ Wrapping.....	115
Figure 5-23 : OBJ Rotation and Scaling .....	120
Figure 5-24 : Character Data Address Mapping (16-Color Mode Character OBJ) .....	123

Figure 5-25 : Character Data Address Mapping (256-Color Mode Character OBJ) .....	124
Figure 5-26 : 2D Mapping .....	125
Figure 5-27 : 1D Mapping when Character Name Boundary is 32 Bytes .....	126
Figure 5-28 : 1D Mapping when Character Name Boundary is 128 Bytes .....	127
Figure 5-29 : 2D Map of Bitmap OBJ Data VRAM (128 Horizontal Dots) .....	131
Figure 5-30 : 2D Image Map of Character Name VRAM .....	131
Figure 5-31 : 2D Map of Bitmap OBJ Data VRAM (256 Horizontal Dots) .....	132
Figure 5-32 : 2D Image Map of Character Name VRAM .....	132
Figure 5-33 : 1D Map of VRAM with 8x8-Dot Characters .....	133
Figure 5-34 : 1D Map of VRAM with 16x16-Dot Characters .....	134
Figure 5-35 : Backdrop Schematic.....	135
Figure 5-36 : Standard Palette RAM Addresses (Add 0x400h for 2D Graphics Engine B) .....	136
Figure 5-37 : 16 Colors x 16 Palettes .....	136
Figure 5-38 : 256 Colors x 1 Palette .....	136
Figure 5-39 : BG Extended Palette Memory Map .....	138
Figure 5-40 : OBJ Extended Palette Slot Memory Map .....	141
Figure 5-41 : Altering a Window Shape .....	144
Figure 5-42 : Display Priority of Window 0, Window 1, and the OBJ Window .....	145
Figure 5-43 : Alpha-Blending Display Priority .....	148
Figure 5-44 : Display Changes According to Mosaic Size .....	150
Figure 5-45 : Display Priority.....	151
Figure 6-1 : 3D Graphics Hardware Block Diagram.....	153
Figure 6-2 : Right-Handed Coordinate System.....	158
Figure 6-3 : Coordinate Transformation Flow Chart .....	160
Figure 6-4 : Perspective Projections .....	161
Figure 6-5 : Orthogonal Projections .....	162
Figure 6-6 : Z-Buffering and W-Buffering (Perspective Projection).....	164
Figure 6-7 : Z-Buffering and W-Buffering (Orthogonal Projection).....	166
Figure 6-8 : Transferring Packed and Non-Packed Commands .....	168
Figure 6-9 : Continuous Writing to the Geometry FIFO using STM or STRD Instructions.....	169
Figure 6-10 : Case 1: Preventing a Command without Parameters from Being the First Valid Command .....	170
Figure 6-11 : Case 2: Preventing a Command without Parameters from Being the First Valid Command .....	171
Figure 6-12 : When the First Valid Command has no Parameters .....	172
Figure 6-13 : Schematic of the Main Geometry Command Processes.....	177
Figure 6-14 : Size and Position of the Viewport.....	180
Figure 6-15 : Material Color Schematic .....	190
Figure 6-16 : Directional Vector Relational Diagram (Diffuse Reflection Color) .....	190
Figure 6-17 : Directional Vector Relational Diagram (Specular Reflection Color) .....	191
Figure 6-18 : Specular Reflection Shininess.....	193
Figure 6-19 : Order in which the Vertex commands issues vertices.....	200
Figure 6-20 : Line segment using sides from a triangle .....	201
Figure 6-21 : Quadrilateral Polygon shapes that yield unintended shapes.....	201
Figure 6-22 : The Process for Adding the X Coordinate .....	204
Figure 6-23 : Polygon Clipping Color Distortion.....	205
Figure 6-24 : Texture Image Space (for an Image of 1,024x1,024 Texels) .....	206
Figure 6-25 : Texture Image Space (No Repeats).....	210
Figure 6-26 : Texture Image Space (with Repeats) .....	211
Figure 6-27 : Box to Be Tested .....	216
Figure 6-28 : Release of Shared Vertices Among Connected Polygons (Clip Coordinate System) .....	224
Figure 6-29 : Color Buffer's FIFO Operation .....	228
Figure 6-30 : Rendering Engine Blanking Periods.....	229
Figure 6-31 : VRAM Mapping of Clear Images (Texture Image Slots 2 and 3 Shared).....	233

Figure 6-32 : Clear Image Offset .....	234
Figure 6-33 : Shadow Volume .....	238
Figure 6-34 : When Drawing a Shadow Polygon for a Mask .....	239
Figure 6-35 : When Drawing the Shadow Polygon for Rendering .....	239
Figure 6-36 : Technique for Rendering a Shadow on a Translucent Polygon .....	240
Figure 6-37 : Transformations Using a Toon Table .....	241
Figure 6-38 : Texture Image Sampling .....	244
Figure 6-39 : When an 8x8 texel Texture is Applied to an 14-dot Wide Polygon.....	244
Figure 6-40 : When an 8x8 texel Texture Is Applied to an 8-dot Wide Polygon .....	245
Figure 6-41 : Displaying Front and Back Surfaces of an LCD .....	245
Figure 6-42 : Texture Image Slots .....	252
Figure 6-43 : Palette Base and Palette Address (4-color palette).....	255
Figure 6-44 : Palette Base and Palette Address (16-Color Palette and 256-Color Palette) .....	256
Figure 6-45 : Palette Base and Palette Address (4x4 Texel Compression).....	256
Figure 6-46 : Palette Base and Palette Address (A3I5, A5I3).....	257
Figure 6-47 : Depth Values and Fog Density .....	262
Figure 6-48 : Anti-aliasing .....	265
Figure 6-49 : Final LCD Image Output (Anti-Aliasing) .....	266
Figure 6-50 : H Offset for a 3D Surface .....	268
Figure 10-1 : POWCNT: Graphics Power Control Register .....	287
Figure 13-1 : Sound Circuit Outline Diagram .....	299
Figure 13-2 : Pulse Width Modulation (PWM).....	303
Figure 13-3 : Overall Sound Block Diagram .....	304
Figure 13-4 : Channel 0-3 and Sound Capture 0-1 Block Diagram .....	305
Figure 13-5 : Channel 4-7 Block Diagram.....	307
Figure 13-6 : Channel 8-15 Block Diagram.....	308
Figure 13-7 : Example of Sound Usage (Normal).....	309
Figure 13-8 : Example of Sound Usage (Reverb).....	310
Figure 13-9 : Example of Sound Usage (Effect) .....	311
Figure 15-1 : Comparison of LCD Dot Size and Touch Pen Size .....	316
Figure 15-2 : Touch Panel Structure .....	317
Figure 16-1 : Microphone Schematic .....	319

## Tables

Table 1-1 : Overview of LCD Screen Specifications .....	4
Table 1-2 : Differences in NITRO Consoles by Destination Region .....	8
Table 2-1 : Memory Configuration and Specifications .....	9
Table 2-2 : DMA Transfer Speeds between Internal Work RAM and VRAM .....	9
Table 2-3 : DMA Settings to Function as a Look-Ahead Buffer .....	10
Table 2-4 : DMA Transfer Speeds between Main Memory and Internal Work RAM .....	10
Table 2-5 : DMA Transfer Speeds between Main Memory and VRAM .....	10
Table 2-6 : Basic Access Cycles .....	13
Table 2-7 : Inserting Waits According to the Access Start Address .....	13
Table 2-8 : Options for VRAM Use .....	17
Table 2-9 : VRAM-A and VRAM-B Allocations .....	27
Table 2-10 : VRAM-C and VRAM-D Allocations .....	27
Table 2-11 : VRAM-E Allocations .....	27
Table 2-12 : VRAM-F and VRAM-G Allocations .....	28
Table 2-13 : VRAM-H Allocations .....	28
Table 2-14 : VRAM-I Allocations .....	28
Table 2-15 : Result of Accessing an Undefined Register .....	31
Table 3-1 : Cache Specifications .....	37
Table 3-2 : Cache Operations .....	40
Table 3-3 : Access Modes When the Data is Being Written .....	42
Table 3-4 : Cache Line States (Write-Back Mode) .....	43
Table 3-5 : Cache Line States (Write-Through Mode) .....	44
Table 4-1 : Selector and Register Selection Flag Map .....	47
Table 4-2 : LCD Clock Specifications .....	49
Table 4-3 : LCD Scan Timing Specifications .....	50
Table 4-4 : Period when Graphics Engines Access Memory .....	52
Table 4-5 : Overview of the Display Modes (2D Graphics Engine A) .....	58
Table 4-6 : Overview of the Display Modes (2D Graphics Engine B) .....	58
Table 4-7 : DMA Configuration when Using the Main Memory Display Mode .....	65
Table 5-1 : List of BG Modes (2D Graphics Engine A) .....	77
Table 5-2 : List of BG Modes (2D Graphics Engine B) .....	78
Table 5-3 : Basic Features of BG Types .....	79
Table 5-4 : Specifications for BG Types .....	80
Table 5-5 : Screen Sizes (2D Graphics Engine A) .....	83
Table 5-6 : Screen Sizes (2D Graphics Engine B) .....	84
Table 5-7 : OBJ Overview .....	109
Table 5-8 : Rendering Cycle Count and Number of OBJ Displayable on One Line .....	110
Table 5-9 : OBJ Shape and OBJ Size Settings .....	116
Table 5-10 : Character OBJ .....	118
Table 5-11 : Bitmap OBJ .....	118
Table 5-12 : Starting Character Name Boundaries for OBJ Attribute 2 .....	121
Table 5-13 : Starting Character Name Boundaries for OBJ Attribute 2 .....	122
Table 5-14 : Character Name Boundaries .....	133
Table 5-15 : Palettes and BG Types .....	139
Table 5-16 : Color Special Effects .....	146
Table 5-17 : Color Special Effects and Processing .....	147
Table 6-1 : Capacity of Polygon List RAM and Vertex RAM .....	154
Table 6-2 : Geometry Engine Specifications .....	158
Table 6-3 : Geometry Commands (in Command Code Order) .....	173
Table 6-4 : Number of Geometry Command Run Cycles & Timing Related to Command Issue (in Command Code Order) .....	174
Table 6-5 : PLTT_BASE Values and Shift Volumes .....	212

Table 6-6 : Vertex RAM Consumed and the Maximum Number of Polygons Stored per Primitive Type .....	223
Table 6-7 : Rendering Engine Specification List .....	226
Table 6-8 : Overview of Rendering Engine Features .....	227
Table 6-9 : Buffers in the Rendering Engine .....	228
Table 6-10 : Rendering Engine Timing Specifications .....	229
Table 6-11 : Maximum Polygons Rendered per Line and Fill Rate (Calculated Values) .....	230
Table 6-12 : Texture Blending Equations (toon table) .....	242
Table 6-13 : Texture Blending Equation (Highlight Shading) .....	243
Table 6-14 : Texture Blending Equations (Decal Mode) .....	246
Table 6-15 : Texture Blending Expressions (Modulation Mode) .....	247
Table 6-16 : List of Texture Formats .....	248
Table 6-17 : Texel Color Values .....	251
Table 6-18 : Equation when $\alpha$ -Blending .....	258
Table 6-19 : Fog-Blending Equations .....	263
Table 6-20 : Anti-aliasing Equations .....	264
Table 6-21 : Anti-Aliasing and Alpha-Blending with a 2D Surface .....	266
Table 7-1 : Processing Details for the Address Update Method .....	275
Table 7-2 : Register Configuration (Step 1) .....	276
Table 7-3 : Register Configuration (Step 3) .....	277
Table 7-4 : ARM9-DMA Parallel Start Category Chart .....	277
Table 10-1 : Conditions for Waking from Sleep Mode .....	285
Table 10-2 : Access to Memory and Registers when Clock Signal is Stopped .....	289
Table 10-3 : Battery State Data .....	290
Table 10-4 : DS Opened/Closed State Data .....	290
Table 11-1 : Calculation Bit Count and Calculation Cycle Count by Divider Mode .....	293
Table 11-2 : Input Bit and Calculation Cycle Count by Computation Mode .....	295
Table 13-1 : Duty Ratio and PSG Rectangular Wave Waveforms .....	301
Table 13-2 : Overview of Data Formats and Playable Channels .....	302
Table 13-3 : Switch Input Priority from Channels 1 and 3 to the Mixer .....	306
Table 14-1 : Wireless Communications Hardware Specifications .....	313
Table 15-1 : Touch Panel Input Data .....	315
Table 16-1 : Ranges of Possible Settings for Gain and Amplitude Resolution .....	319
Table 16-2 : Microphone Input Values when there is no Sound .....	320
Table 16-3 : Guaranteed Microphone Input Ranges .....	320
Table 17-1 : Real-Time Data .....	321
Table 17-2 : Settings for Alarm 1 and Alarm 2 .....	321
Table 18-1 : Owner Information Data .....	323
Table 18-2 : NITRO Initial Setting Data .....	324
Table 18-3 : RTC Operation Information Data .....	324





## Revision History

Version	Date	Description
1.56	2007/05/17	<ul style="list-style-type: none"> <li>Added a detailed figure about boxes that are tested with the BoxTest command of the geometry engine. (The figure numbers after 6-27 also changed because Figure 6-27 was added.)</li> </ul>
1.55	2007/04/18	<ul style="list-style-type: none"> <li>Changed the term for the headset to the official name "Nintendo DS Series Headset" (including Figure 16-1).</li> <li>Changed the startup time from power-on of the microphone to 3 seconds.</li> </ul>
1.54	2006/01/09	<ul style="list-style-type: none"> <li>Added Korea to the destination regions.</li> <li>Corrected error in A315 translucent texture INDEX value.</li> <li>Added Korean to the language setting for NITRO initial setting data.</li> </ul>
1.53	2006/10/20	<ul style="list-style-type: none"> <li>Added information regarding color distortion resulting from polygon clipping (the figure numbers for chapter 6 have been changed due to the addition of figure 6.23).</li> <li>Added supplementary information polygon process cycle count.</li> </ul>
1.52	2006/08/08	<ul style="list-style-type: none"> <li>Clarified the range of microphone input values when there is no sound, and the range of guaranteed microphone input values.</li> </ul>
1.51	2006/06/15	<ul style="list-style-type: none"> <li>Added to Table 6-4 the fact that the number of run cycles corresponding to each source increases when performing texture coordinate conversion.</li> </ul>
1.50	2006/03/13	<ul style="list-style-type: none"> <li>Made clarifications that this manual supports the Nintendo DS and Nintendo DS Lite systems.</li> <li>Described the differences in the System Overview LCD on the Nintendo DS and the Nintendo DS Lite.</li> <li>Described the difference in the directions of the Upper Screen LCD and the Lower Screen LCD on the Nintendo DS.</li> <li>Changed the area of the 0x01008000 address in Figure 1-2 to a Command TCM image.</li> <li>Described the reason for a cartridge return from sleep mode; pak uses a cartridge interrupt.</li> <li>Described conditions where the headphones will not generate any sound when the LCD is OFF.</li> <li>Described conditions where a DS may not reliably shut down when its LCD is OFF and the API is used to shut down.</li> </ul>
1.44	2005/12/19	<ul style="list-style-type: none"> <li>Corrected a description regarding data preload (page 41) because the data preload feature does not work in ARM946E-S.</li> <li>In <a href="#">Chapter 16</a>, added a caution regarding variation of microphone input values when there is no sound.</li> <li>In <a href="#">Chapter 16</a>, added that there is a possibility of noise, due to feedback, when continuously recording the input from the microphone and playing sounds at the same time.</li> </ul>
1.43	2005/11/25	<ul style="list-style-type: none"> <li>Added that an overflow is generated when performing additions by setting a value with the BoxTest command in the geometry engine.</li> </ul>
1.42	2005/9/15	<ul style="list-style-type: none"> <li>Added paragraph in 6.2.16.1 "Reasons for Released Vertices in the Polygon Attribute Settings for Rendering 1-Dot Polygons."</li> </ul>

Version	Date	Description
1.41	2005/07/08	<ul style="list-style-type: none"> <li>Replaced the contents of <a href="#">Figure 13-7</a>, since they were identical to those of <a href="#">Figure 13-9</a>.</li> </ul>
1.40	2005/07/01	<ul style="list-style-type: none"> <li>Added a caution regarding the address of the data TCM in the memory map.</li> <li>Added a description regarding the Chinese specification to <a href="#">Chapter 1</a>.</li> <li>Changed the address for the DTCM inside the memory map.</li> <li>Made corrections to an error in the cycling number of <a href="#">Figure 2-1</a>.</li> <li>Clarified that the protection units mentioned in this manual are examples of configuration.</li> <li>Deleted the protection unit configurations for the release version and the debug version (listed in the NITRO-SDK Function Reference Manual).</li> <li>Added cautions for the data cache output.</li> <li>Added a supplemental description for the NCNB mode.</li> <li>Added that the virtual screen in <a href="#">Figure 5-22</a> has a size of 512x256.</li> <li>Corrected an error in the attributes of the color special effects / change shininess factor register.</li> <li>Corrected an error in the formula for the color value of the texel in <a href="#">Table 6-17</a>.</li> <li>Corrected an error in the sound circuit of <a href="#">Figure 13-4</a>, and added a related caution.</li> <li>Added a description about individual margins of error for microphone sensitivity.</li> <li>Deleted the ROM internal register data in Chapter 19, since the same contents are now listed in the Nintendo DS Game Card Manual, and added references to that manual where necessary.</li> <li>Added Chinese as one of the configurable languages that can be used in the initial NITRO configuration data.</li> </ul>
1.31	2005/04/15	<ul style="list-style-type: none"> <li>Corrected error in 2-1 (page 12) regarding access cycle (changed "wait" to "access").</li> <li>Added notes on page 145 about windows and corrected errors regarding the window position setting register.</li> <li>Added note on page 236 about wireframes.</li> </ul>

Version	Date	Description
1.30	2005/02/15	<ul style="list-style-type: none"> <li>Revised expressions that may be misleading in the overview description of <a href="#">RTC</a>.</li> <li>Added a description for <a href="#">Table 2-1</a>, because it was unclear what was indicated by the numerical values in the table.</li> <li>Added 8 bits to <a href="#">Table 2-1</a> items in the "Read" column under "Bit width which Main processor can access."</li> <li>Added a note regarding the access cycle of a Game Pak on page <a href="#">12</a>.</li> <li>Corrected errors in the address and size of ARM7 dedicated internal Work RAM in <a href="#">Figure 2-11</a>.</li> <li>Added a note regarding the blank detection flag. (Added <a href="#">Table 4-4</a> and renumbered those that follow.)</li> <li>Corrected the expressions, beginning on page <a href="#">183</a>, regarding the multiplication of matrices with Geometry Engine so that they match the names of the determinant of matrices.</li> <li>Added an example of application to "Fog Enable Flag" in "<a href="#">3. Draw the shadow polygon for rendering</a>" of "Shadow Polygon."</li> <li>Added that Sleep Mode is recommended for the power control of <a href="#">LCD Backlight</a> and LCD.</li> <li>Added PWM block in sound block diagrams, <a href="#">Figure 13-3</a> through 13-9.</li> <li>Added the indications in the sound block diagrams (<a href="#">Figure 13-7</a> through <a href="#">Figure 13-9</a>) to show the data precision of the sound circuitry.</li> <li>Added to a note on <a href="#">page 315</a> regarding the touch determination flag and data validity flag of the touch panel.</li> <li>Corrected the misalignment of the ruled lines in Figure 19-1.</li> </ul>
1.22	2004/11/16	<ul style="list-style-type: none"> <li>Corrected errors in <a href="#">Figure 5-19</a>.</li> </ul>
1.21	2004/11/11	<ul style="list-style-type: none"> <li>Documented the difference between directions of upper and lower LCDs in "<a href="#">1.1.3 LCD</a>" and "<a href="#">4.2 LCD</a>".</li> <li>Revised description of character base block in 2D graphics engine B of "<a href="#">5.2.2 BG Control</a>".</li> </ul>
1.20	2004/11/8	<ul style="list-style-type: none"> <li>Removed 512x256 settings.</li> <li>Deleted Figure 5-14 and renumbered subsequent figures.</li> <li>"<a href="#">6.2.7 Swapping the Rendering Engine's Reference Data</a>": Revised the description of the rendering engine's depth buffering selection flag.</li> <li>"<a href="#">6.3.5.2.1.4 4x4 Texel Compression Textures</a>": Added a note.</li> <li>Corrected numerical error in Figure 6-24.</li> <li>DMAx control register (x=0-3): Corrected word count.</li> <li>"<a href="#">16 Microphone</a>": Explained that noise synched to the V-Blank is superimposed onto microphone input signals.</li> <li>Added a note concerning the method of updating DMA addresses.</li> <li>"<a href="#">10.2.3 LCD</a>": Added a note.</li> <li>In general: removed T.B.D. and revised text.</li> </ul>

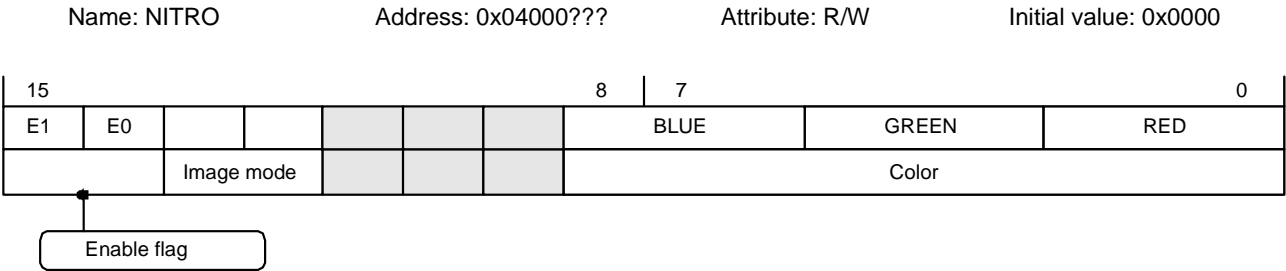
Version	Date	Description
1.10	2004/9/24	<ul style="list-style-type: none"> <li>Deleted "AGB" in the descriptions of cartridges throughout the manual.</li> <li>Revised Figures 1-2, 3-2, 3-3, and 3-4 as well as Tables 3-1 and 3-2 in conjunction with moving the location of the DTCM.</li> <li>Changed the Memory Map during Card Boot.</li> <li>Corrected Figures 5-29, 5-31, and 5-34.</li> <li>Added a note regarding the Start Character Name Boundary in "<a href="#">5.3.3 Character OBJ</a>".</li> <li>Corrected errors in Figure 5-28.</li> <li>Added Figure 5-29 and revised the subsequent numbering of figures.</li> <li>Added a note regarding Geometry FIFO in "<a href="#">6.2.6 Geometry Commands</a>".</li> <li>Added a note on quadrilateral polygons during drawing specification for the back in "<a href="#">6.2.12 Polygon Attributes</a>".</li> <li>Added supplemental description for specification of the polygon render plane. Also replaced Figures 6-34 and 6-35.</li> <li>Added a note on the location of TexImageParam in "<a href="#">6.2.14 Texture Mapping</a>".</li> <li>Changed the shadow polygon attribute to "Render Both Sides."</li> <li>Corrected errors in 6.3.5.2.1.5 regarding A3I5 translucent texture texel data format.</li> <li>Added explanation about the DMA bug that occurs when multiple DMA channels are started in parallel on the ARM9 system bus.</li> <li>Deleted "Standby Mode" in "<a href="#">10 Power Management</a>".</li> <li>Added "LCD" to "Power Controllers" in "<a href="#">10 Power Management</a>".</li> <li>Added the "LCDE Bit" to "Graphics Power Save Register" in "<a href="#">10 Power Management</a>".</li> <li>Included that battery capacity is 10 – 20% for low battery status in "<a href="#">10 Power Management</a>".</li> <li>Changed the mode names in "<a href="#">14 Wireless Communications</a>".</li> <li>Deleted the communication times in Table 14-1.</li> <li>Added a note regarding "<a href="#">15 Touch Panel</a>".</li> <li>Included 80 grams as the force to press in "<a href="#">15 Touch Panel</a>".</li> <li>Reflected in "<a href="#">17 Real-Time Clock (RTC)</a>" the elimination of time notation settings and the elimination of the PM flag.</li> <li>Changed the data contents in "<a href="#">18.2 Owner Information Data</a>".</li> <li>Reflected in "19 ROM Registration Data" the changes in ROM internal registration data.</li> </ul>
1.00	2004/8/2	Initial release.

# About the Notation Used in this Programming Manual

## Registers

Detailed classifications are shown at the top of the register, while broader classifications are shown on the bottom. If text does not fit, then the description is shown below the register, as shown with the “enable flag” in the example below. Notation such as “d15” is used to refer to a specific bit (in this case the highest-order bit in a 16-bit register).

### Example: NITRO Register



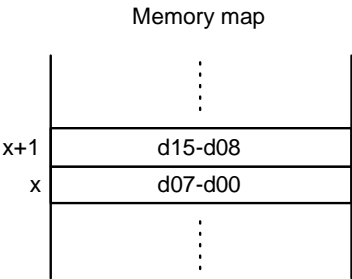
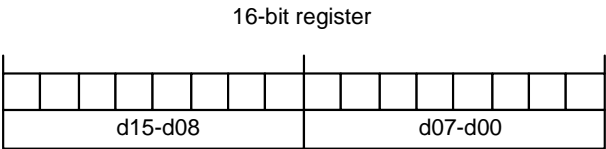
## Bit lengths

Bit lengths for bytes, half-words, and words are defined as follows:

- 8-bit: Byte
- 16-bit: Half-word
- 32-bit: Word

## Endian

NITRO adopts the *little-endian* method. Therefore, in a 16-bit register, the address for d15–d08 is one more than the address for d07–d00.





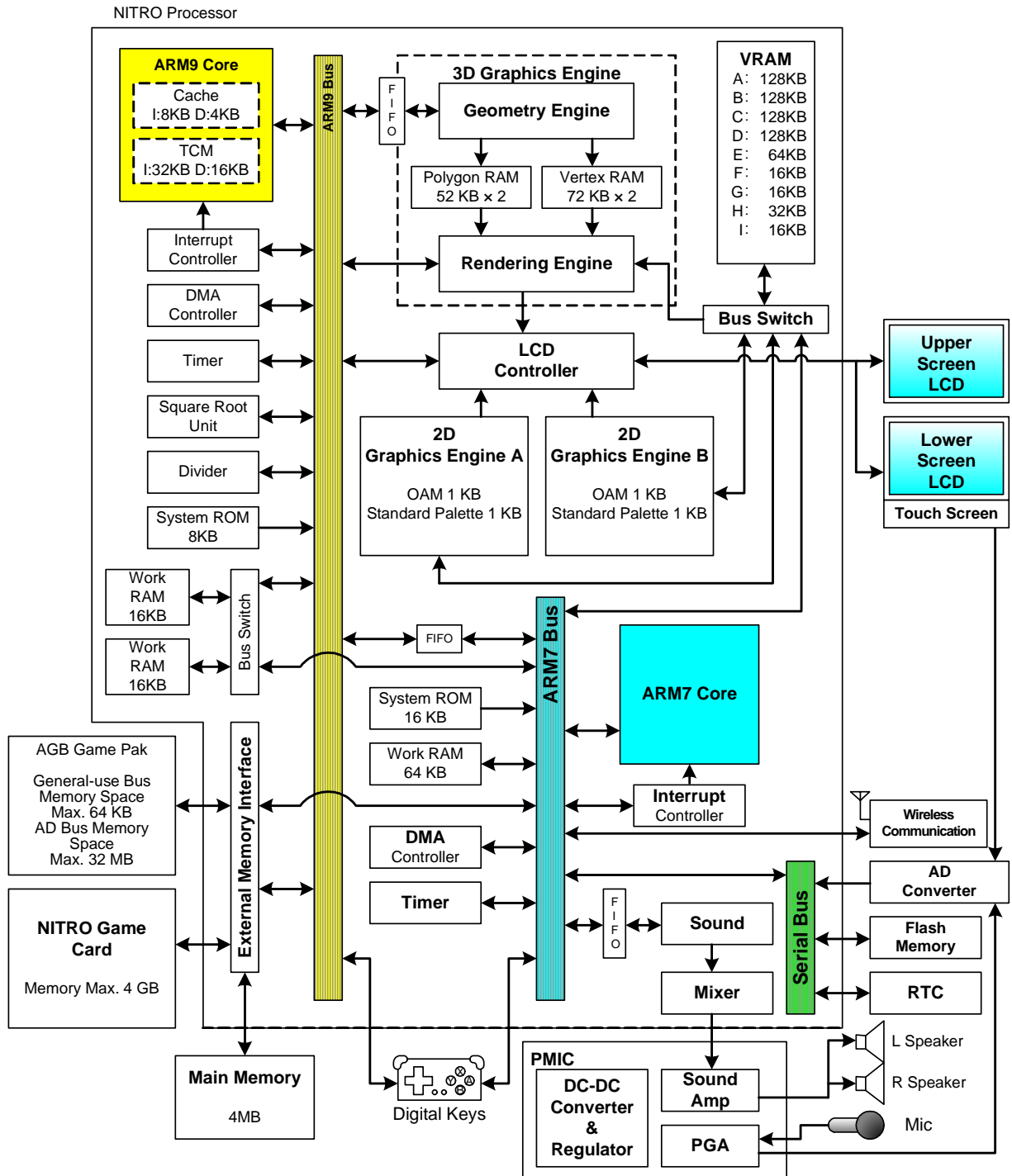
# 1 System

## 1.1 System Outline

The overall NITRO system block diagram is shown in Figure 1-1.

The supported systems are Nintendo DS™ and Nintendo DS Lite.

**Figure 1-1 : Overall System Block Diagram**



### 1.1.1 NITRO Processor

The NITRO processor is a combined chip that consolidates ARM9 and ARM7 CPU cores with NITRO features and memory for the 2D and 3D graphics engines.

The specifications of the NITRO processor are as follows:

- The CMOS Multi CPU

Main processor core	ARM946E-S (67.028 MHz)
Subprocessor core	ARM7TDMI (33.514 MHz)

- Compatibility

Switches between NITRO mode and AGB compatibility mode.

- Graphics Engines

2D Graphics Engines A and B		33.514MHz
3D Graphics Engine	Geometry Engine	33.514MHz Maximum 4 million vertices per second 4 x 4 matrix computation 6-plane clipping Lighting (4 parallel light sources) Matrix stack Texture coordinate conversion Box culling test
	Rendering Engine	33.514MHz Maximum 120 thousand polygons per second Maximum 30 million pixels per second Triangular and quadrilateral rendering Texture format 4-, 16-, and 256-color palette formats Bitmap format 4 x 4 texel compression format Translucent (A3I5, A5I3) format Texture size 8 x 8 to 1024 x1024 Alpha blending Alpha test Fog Toon shading Edge marking Anti-aliasing



- Memory

System ROM	ARM9 : 8 KB (2K x 32 bit) ARM7 : 16 KB (4K x 32 bit)
NITRO Processor Internal Work RAM	ARM9, ARM7 shared: 32 KB ( 8 K x 32 bit) ARM7 dedicated : 64 KB (16 K x 32 bit)
VRAM	Total of 656 KB (128 KB + 128 KB + 128 KB + 128 KB + 64 KB + 16 KB + 16 KB + 32 KB + 16 KB)
System Clock	33.514 MHz

- LCD Controller (built-in for two LCDs: the upper and lower screens)

Display Size	256 x 192 x RGB dots
Display Colors	262,144 colors (R:G:B = 6:6:6)
Dot Clock	5.586 MHz

- Sound

ADPCM/PCM 16 channels (up to 6 channels for the PSG sound source and up to 2 channels for noise)  
Includes sound capture capabilities (using reverb, etc.).

- Timers

ARM9 : 16-bit timer x 4

ARM7 : 16-bit timer x 4

- DMA

ARM9 : 4 channel

ARM7 : 4 channel + Sound DMA features

- Accelerator

Divider

Square root unit

- External Memory Interface

DS Game Card interface, DS accessories interface (AGB compatible)

### 1.1.2 Main Memory

The main memory is 4 MB (expanded to 8 MB for NITRO debugging) and is connected to the NITRO processor as an independent chip.

Because the NITRO card bus is not mapped to the CPU address space, applications and data must be executed after loading them into main memory.

The load speed from the NITRO card bus into main memory is approximately 5.96 MB per second.

The application for ARM9 is transmitted from the NITRO card to main memory by system ROM at startup.

The application for ARM7 is transmitted to the ARM7 exclusive work RAM at startup.

### 1.1.3 LCD

There are two LCD screens, an upper screen and a lower screen.

An overview of both LCD screen specifications is shown in Table 1-1.

**Table 1-1 : Overview of LCD Screen Specifications**

Features	Details	
	Nintendo DS	Nintendo DS Lite
Display Resolution	256 x 192 dots (Ratio 4:3)	Same
Number of Displayable Colors	262,144 colors (RGB=6:6:6)	Same
Screen Size	3 inches	Same
Type	Semi-Transparent Reflective	Transparent
Backlight	Can be switched ON and OFF	Can be switched ON and OFF. Four steps of brightness can be adjusted via the IPL console configurations.

The top and bottom LCDs have the same specifications, but the directions differ on the Nintendo DS, so the order of RGB pixel arrays differs.

### 1.1.4 Digital Keys

The digital keys are START, SELECT, the + Keypad, A, B, X, Y, L, and R.

### 1.1.5 Touch Screen

The entire lower screen LCD is a resistive membrane touch panel that can obtain dot-unit coordinates.

The Nintendo DS system comes equipped with a standard stylus.

### 1.1.6 Microphone

Either the Nintendo DS Series Headset or a built-in omnidirectional condenser microphone can be used.

Sound input from the microphone can be sampled.

### 1.1.7 RTC

The RTC handles timekeeping operations.

By means of an alarm feature, the RTC can wake up the DS from Sleep Mode at a specified time.

### 1.1.8 Wireless Communications

The DS includes an on-board wireless communications unit capable of using the 2.4-GHz bandwidth.

The following modes are available:

- Internet Play that allows connections to wireless LAN (IEEE 802.11b/g) access points
- Multi-Card Play that enables communications with up to 16 DS devices
- Single-Card Play that downloads games from a parent device to child devices that are not equipped with DS Game Cards

### 1.1.9 Nintendo DS Game Card

The DS Game Card is a game card with NITRO-exclusive security features.

A backup device can be installed in addition to the ROM.

The DS Game Card connects to the NITRO Processor with an external memory interface. Data transfer speeds within the DS Game Card can be as fast as 5.96 MB per second.

For more information, refer to the Nintendo DS Game Card Manual.

### 1.1.10 DS Accessories

Existing AGB Game Paks can be used in the AGB compatibility mode.

For applications in NITRO mode, the DS can access the data inside an AGB Game Pak plugged into the Game Pak slot on the DS. However, the EEPROM cannot be accessed on an AGB Game Pak that uses EEPROM in a backup device.

NITRO option paks, which can be used to save data or for sensors, can be used as accessories for NITRO games.

**Note:** DMG and CGB Game Paks cannot be used with NITRO.

## 1.2 Memory Map

The overall memory maps for the ARM9 and ARM7 on the DS are shown in Figure 1-2.

The access attributes for the ARM9 memory space are determined by the configuration of the protection units.

For more details, see "[3.1 Protection Unit](#)" on page 35.

- About the image

A decoder converts the address output by the CPU to a memory address.

Because the decoder does not normally decode all the address bits, when an address that is not mounted in memory is accessed, it is converted to the address of the memory located closest to the smallest part of the address. (See the note below.)

Images are regions of the memory map that appear where nothing normally exists (regions differ from the physical memory).

Even if the address is different on the CPU, the address is the same in memory. Therefore, accessing an image is the same as accessing the physical memory. Furthermore, when the image region is larger than the memory size, an image the same size as the physical memory is repeated.

For example, the BG-VRAM physical memory for 2D Graphics Engine A occupies the 512 kilobytes in the 0x06000000 - 0x0607FFFF section in the Memory Map, but the image region occupies the 1536 KB of the 0x06080000 - 0x061FFFFFF section, so there are three images.

**Note:** For the shared ARM9 and 7 internal Work RAM, the position of the physical memory on the memory map is shifted.

Normally, physical memory begins at 0x03000000. However, there is an advantage when accessing with the memory map address because the addresses for ARM7 dedicated Work RAM continue from those values.

In addition, there are regions of indeterminate data that have no image (see Figure 1-2).

**Note:** Do not access the indeterminate data.

**Figure 1-2 : ARM9 and ARM7 Overall Memory Maps**

ARM9 Overall Memory Map		ARM7 Overall Memory Map	
0x0A010000		0x0A010000	
0x0A000000	DS Accessory RAM (64 KB)	0x0A000000	DS Accessory RAM (0–64 KB)
0x08000000	DS Accessory ROM (32 KB)	0x08000000	DS Accessory ROM (32 MB)
0x07000800	OAM Image	0x07000000	Indeterminate Data
0x07000400	2D Graphics Engine B OAM (1 KB)		
0x07000000	2D Graphics Engine A OAM (1 KB)	0x06040000	Internal Expanded Work RAM Image
0x068A4000	VRAM Image for LCDC		
0x06800000	VRAM for LCDC (656 KB)		
0x06620000	OBJ-VRAM Image		
0x06600000	2D Graphics Engine B OBJ-VRAM (max. 128 KB)		
0x06440000	OBJ-VRAM Image		
0x06400000	2D Graphics Engine A OBJ-VRAM (max. 256 KB)		
0x06220000	BG-VRAM Image		
0x06200000	2D Graphics Engine B BG-VRAM (max. 128 KB)		
0x06080000	BG-VRAM Image		
0x06000000	2D Graphics Engine A BG-VRAM (max. 512 KB)	0x06000000	Internal Expanded Work RAM (max. 256 KB)
0x05000800	Palette RAM Image	0x04810000	Indeterminate Data
0x05000600	2D Graphics Engine B Palette RAM for OBJ (512 B)		
0x05000400	2D Graphics Engine B Palette RAM for BG (512 B)		
0x05000200	2D Graphics Engine A Palette RAM for OBJ (512 B)		
0x05000000	2D Graphics Engine A Palette RAM for BG (512 B)		
0x04000000	I/O Registers (see note)	0x04808000	Wireless Communications Wait State 1
0x03800000	ARM9, 7 Shared Internal Work RAM Image	0x04800000	Wireless Communications Wait State 0
		0x04000000	I/O Registers (see note)
0x037F8000	ARM9, 7 Shared Internal Work RAM (max. 32 KB)	0x03810000	ARM7 Exclusive Internal Work RAM Image
0x03000000	ARM9, 7 Shared Internal Work RAM Image	0x03800000	ARM7 Exclusive Internal Work RAM (64 KB)
0x02800000	Main Memory Image	0x037F8000	ARM7, 9 Shared Internal Work RAM (max. 32 KB)
0x027E0000	Data TCM (16 KB): Moveable	0x03000000	ARM7, 9 Shared Internal Work RAM Image
0x02400000	Main Memory (When Extended) (4 MB)	0x02800000	Main Memory Image
0x02000000	Main Memory (4 MB)	0x02400000	Main Memory (When Expanded) (4 MB)
0x01008000	Instruction TCM Image	0x02000000	Main Memory (4 MB)
0x01000000	Instruction TCM (32 KB)	0x00010400	Indeterminate Data
0x00000000	Indeterminate Data	0x00000000	System ROM (64 KB)

**Note:** The I/O registers are different on ARM9 and ARM7.

**Note:** The data TCM can be moved around, so the address in the figure above is just an example.

### 1.3 Accessing Devices Connected to the Subprocessor

On NITRO, you must use the API to access devices connected to the subprocessor.

By using the API, you can access the device, regardless of what state the subprocessor is in.

The following devices connect to the subprocessor: wireless communications, a portion of the digital keys, the sound, Touch Screen, microphone, RTC, and built-in flash memory.

- What is an API?

An *Application Program Interface* (API) is a group of functions that increase efficiency when developing applications.

In general, the API is used in low-level system calls and to control hardware.

**Note:** It is possible to access the registers related to the interface with the ARM7 subprocessor in ARM9, but these registers should not be accessed if using the API.

### 1.4 Startup Mode

The following modes can be selected from the menu that appears after NITRO starts up.

Startup mode is available only from the menu that appears after NITRO starts up.

Startup mode cannot be switched in the application.

#### 1.4.1 NITRO Mode

In this mode, all NITRO features are usable.

#### 1.4.2 AGB Compatibility Mode

Of the NITRO processors, the subprocessor starts up at 16.777 MHz as an AGB CPU.

In this mode, the LCD1 screen, the 2D graphics engine, the LCD controller (LCDC), and a part of VRAM are usable, but the ARM9 and ARM9-related peripheral circuitry, the 3D graphics engine, and the serial bus are not usable.

In other words, the features not implemented on the AGB are not usable.

**Note:** When operating in AGB compatibility mode, the serial bus becomes unusable. Therefore, the wireless communications, Touch Screen, RTC, microphone, and built-in flash memory connected to that bus are also not usable. In addition, the X Button and Y Button become unusable.

### 1.5 Destination

As shown in Table 1-2, there are three Nintendo DS system configurations, depending on the destination market.

**Table 1-2 : Differences in NITRO Consoles by Destination Region**

Destination	Everywhere Except China and Korea	Korea	China (Excluding Hong Kong and Taiwan)
<b>Usable Banner Font</b>	Hiragana, Katakana, Alphabetical, Signed Alphabetical, Numeric, etc.	Adds 2350 Hangul characters to the character sets found in the non-Chinese/non-Korean systems	Adds approximately 6700 simplified Chinese characters to the character sets found in the non-Chinese/non-Korean systems
<b>Configurable Languages</b>	Japanese, English, French, German, Italian, Spanish	Japanese, English, French, German, Spanish, Korean	English, French, German, Italian, Spanish, Chinese

**Caution:** Applications made specifically for China will not function in Nintendo DS systems sold in regions outside of China.

## 2 Memory

Table 2-1 shows the configuration and specifications of the memory built into NITRO.

**Table 2-1 : Memory Configuration and Specifications**

Memory Type	Bus Width	Access Cycle	Bit Width that Allows DMA Access		Bit Width that Allows Main Processor Access	
			Read	Write	Read	Write
DS Accessory RAM (SRAM, flash memory, etc.)	8	6-18	-	-	8	8
DS Accessory ROM (ROM, flash memory, etc.)	16	1st 6-18 2nd 4-6	16/32	16/32	8/16/32	16/32
OAM	32	1	16/32	16/32	8/16/32	16/32
VRAM	16	1	16/32	16/32	8/16/32	16/32
Palette RAM	16	1	16/32	16/32	8/16/32	16/32
I/O Registers	32	1	16/32	16/32	8/16/32	8/16/32
Internal Work RAM	32	1	16/32	16/32	8/16/32	8/16/32
Main Memory	16	1st R:5 / W:4 2nd 1	16/32	16/32	8/16/32	8/16/32
System ROM	32	1	-	-	8/16/32	-
TCM/Cache	32	½	-	-	8/16/32	8/16/32

The values given for the number of access cycles correspond to a bus frequency of 33.514 MHz.

Furthermore, these values are for when memory is accessed in a bit width that is equal to or less than the bus width. When memory is accessed in a bit width that is larger than the bus width, the number of access cycles is limited to the bit width divided by the bus width.

**Note:** Because the access cycle in NITRO Mode when accessing AGB Game PAK EEPROM is shorter than the AGB access cycle, data cannot be correctly read and written.

- Transfer speeds between memories**

DMA transfer speeds between memories can be calculated from the bus width and the access cycle values shown in Table 2-1. Table 2-2 shows an example of DMA transfers between Internal Work RAM and VRAM.

**Table 2-2 : DMA Transfer Speeds between Internal Work RAM and VRAM**

Transfer Memory	DMA Transfer Bit Count	Cycles Used for Reading	Cycles Used for Writing	Total Number of Cycles	Transfer Speed (MB/sec)
From Internal Work RAM to VRAM	16	1	1	2	31.96
	32	1	2	3	42.62
From VRAM to Internal Work RAM	16	1	1	2	31.96
	32	2	1	3	42.62

- Main memory transfer speeds

Main memory can function as a look-ahead buffer by setting DMA for reading as shown in Table 2-3.

**Table 2-3 : DMA Settings to Function as a Look-Ahead Buffer**

Property to Set	Set Value
Transfer Bit Count	32 bits
How to Update Source Address	Increment
Destination Address	Not main memory

Capable of holding 16 bits, the look-ahead buffer reads from main memory while data is being written to the destination memory.

Table 2-4 and Table 2-5 show the DMA transfer speeds between main memory and internal Work RAM, and between main memory and VRAM. The asterisk (\*) denotes a shortening of the total number of cycles due to the look-ahead buffer. For more details about the look-ahead buffer, see ["2.1.1 Main Memory"](#) on page 13.

**Table 2-4 : DMA Transfer Speeds between Main Memory and Internal Work RAM**

Transfer Memory	DMA Transfer Bit Count	Cycles Used for Reading	Cycles Used for Writing	Total Number of Cycles	Transfer Speed (MB/sec)
From Main Memory to Internal Work RAM	16	1st 5 2nd- 1	1	1st 6 2nd- 2	2nd- 31.96
	32	1st 6 2nd- 2	1	1st 7 * 2nd- 2	2nd- 63.92
From Internal Work RAM to Main Memory	16	1	1st 4 2nd- 1	1st 5 2nd- 2	2nd- 31.96
	32	1	1st 5 2nd- 2	1st 6 2nd- 3	2nd- 42.62

**Table 2-5 : DMA Transfer Speeds between Main Memory and VRAM**

Transfer Memory	DMA Transfer Bit Count	Cycles Used for Reading	Cycles Used for Writing	Total Number of Cycles	Transfer Speed (MB/sec)
From Main Memory to VRAM	16	1st 5 2nd- 1	1	1st 6 2nd- 2	2nd- 31.96
	32	1st 6 2nd- 2	2	1st 8 * 2nd- 3	2nd- 42.62
From VRAM to Main Memory	16	1	1st 4 2nd- 1	1st 5 2nd- 2	2nd- 21.31
	32	2	1st 5 2nd- 2	1st 7 2nd- 4	2nd- 21.31

Transfer speeds by the CPU tend to be slower than calculated because the actual transfer speed is related to the time it takes to execute commands and get to the bus.



## 2.1 External Memory

*DS accessory* refers to the hardware for the AGB-compatible 32-pin Game Pak slot. *DS Game Card* refers to the hardware for the DS Game Card slot.

### EXMEMCNT: External Memory Control Register

Name: EXMEMCNT

Address: 0x04000204

Attribute: R/W

Initial Value: 0x0000

15	14			11			8	7	6	5	4	3	2	1	0
EP	IFM			MP				CP		PHI	ROM2		ROM 1st		RAM
Main Memory				Game Card				DS Accessory							

- [d15, d14]: Main Memory: Settings related to main memory

- EP[d15]: Select the CPU priority

This defines which CPU has priority when ARM9 and ARM7 access main memory at the same time.

0	ARM9 priority
1	ARM7 priority

- IFM[d14]: Interface mode switch flag

0	Asynchronous mode (this setting prohibited)
1	Synchronous mode

**Note:** You must set this to Synchronous mode.

- [d11]: Setting for the DS Game Card

- MP[d11]: Select the CPU with access rights

0	ARM9
1	ARM7

- [d07–d00]: Setting for DS accessories

- CP[d07]: Select the CPU with access rights

0	ARM9
1	ARM7

- PHI[d06–d05]: PHI terminal-output control

This is for supplying the clock from NITRO when the DS accessory has a special chip.

You should normally set this to 00 (Low-level output).

00	Low-level output
01	4.19-MHz clock output
10	8.38-MHz clock output
11	16.76-MHz clock output

- ROM[d04]: ROM 2nd access cycle control \*

0	6 cycles
1	4 cycles

- ROM[d03–d02]: ROM 1st access cycle control \*

00	10 cycles
01	8 cycles
10	6 cycles
11	18 cycles

- RAM[d00–d01]: RAM-region access cycle control \*

00	10 cycles
01	8 cycles
10	6 cycles
11	18 cycles

**\* Access Cycles:**

The system clock frequency of NITRO is twice that of AGB, and the access cycles can be calculated as follows:

(Wait cycle of AGB Game Pak access + 1) \* 2

For example, if the ROM access with AGB is 3-1 access, it will be 8-4 access with NITRO. However, unlike AGB, NITRO has only one address space that can be set to access cycles. There is a possibility that a device that cannot access with the same cycles as that of AGB mask ROM may be mounted. Check the specifications for memory before setting it.

Operations for cycles set shorter than the memory specifications are not guaranteed.

## 2.1.1 Main Memory

### 2.1.1.1 Look-Ahead Buffer

A look-ahead buffer is implemented as the interface to main memory. This look-ahead buffer is shared by ARM9 DMA and ARM7 DMA.

When a 32-bit DMA transfer is conducted from a source address in main memory to a destination address other than main memory, the write time to the destination address is used to read the next data in 16-bit units. This method enables 32-bit reads to be conducted in a single read cycle.

Note that the look-ahead buffer cannot be used for other types of access, such as reading by the CPU or a 16-bit DMA transfer.

### 2.1.1.2 Burst Mode

In Burst mode, the sequential access of a half-word (16-bit width) is conducted in one cycle.

Table 2-6 shows the basic access cycles for random access and sequential access in Burst mode.

**Table 2-6 : Basic Access Cycles**

	Read	Write
Random	5 cycles	4 cycles
Sequential	1 cycle	1 cycle

#### 2.1.1.2.1 Burst Access Conditions

In Burst mode, sequential access is called *burst access*.

Burst access is used during DMA transfer when either the source or the destination address is the main memory and the address update method for main memory is set to increment.

Because DMA transfers from main memory to main memory are all first accesses, routing through internal RAM provides fast transfer speeds.

In addition, burst access is also used when continuous transfers using LDM or STM instructions are executed.

#### 2.1.1.2.2 Conditions for Inserting Waits

When a span of 236 half-words occurs, a wait of three cycles, called a *termination*, is inserted.

When the address is defined from 0 in units of 16 half-words, waits are also inserted if access starts from address 13, 14, or 15, as shown in Table 2-7 (a maximum of 3 waits are inserted). Processes are thus more efficient if data are located from a 4 half-word boundary (8-byte boundary).

**Table 2-7 : Inserting Waits According to the Access Start Address**

First Cycle	2	3	4	5	6	7	8	9	...
13	14	15	wait	16	17	18	19	20	...
14	15	wait	wait	16	17	18	19	20	...
15	wait	wait	wait	16	17	18	19	20	...

### 2.1.1.2.3 Main Memory DMA Transfer Cycles

#### 2.1.1.2.3.1 32-Bit DMA Transfer Cycle from Main Memory to Work RAM

Because main memory has a 16-bit bus and Work RAM has a 32-bit bus, data read in half-word units is written in word units. While writing in word units, the look-ahead buffer reads more half-word data. Transfers to the geometry command FIFO are the same as transfers to Work RAM.

- Basic Cycles

Figure 2-1 shows the basic cycles of the transfer sequence from main memory to Work RAM.

**Figure 2-1 : Transfer Sequence from Main Memory to Work RAM (Basic Cycles)**

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
Work RAM							1W		2W		3W		4W
Main Memory	wait	wait	wait	wait	1LR	1HR	2LR	2HR	3LR	3HR	4LR	4HR	5LR

...	475	476	477	478	479	480	481	482	483	484	485	486	487
...			236W					237W		238W		239W	
...	236LR	236HR	T	T	T	237LR	237HR	238LR	238HR	239LR	239HR	240LR	240HR

- Worst Case

Depending on the main memory address where access starts, a wait known as a termination may occur. In these cases, the access immediately after the termination becomes the first access.

Figure 2-2 shows the worst-case transfer sequence from main memory to Work RAM.

**Figure 2-2 : Transfer Sequence from Main Memory to Work RAM (Worst Case)**

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
Work RAM							1W						
Main Memory	wait	wait	wait	wait	1LR	1HR	T	T	T	wait	wait	wait	wait

14	15	16	17	18	19	20	21	22	23	24	25	26
		2W		3W		4W		5W		6W		7W
2LR	2HR	3LR	3HR	4LR	4HR	5LR	5HR	6LR	6HR	7LR	7HR	8LR

wait	Wait
T	Termination
nLR	Reads the lower 16 bits of the <i>n</i> th 32-bit data
nHR	Reads the upper 16 bits of the <i>n</i> th 32-bit data
nW	Writes the <i>n</i> th 32-bit data

### 2.1.1.2.3.2 Cycles for 32-bit DMA Transfers from Work RAM to Main Memory

Figure 2-3 shows the basic cycles of the transfer sequence from Work RAM to main memory.

**Figure 2-3 : Transfer Sequence from Work RAM to Main Memory (Basic Cycles)**

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
Work RAM	1R						2R			3R			4R
Main Memory		wait	wait	wait	1LW	1HW		2LW	2HW		3LW	3HW	

wait	Wait
nR	Reads the <i>n</i> th 32-bit data
nLW	Writes the lower 16 bits of the <i>n</i> th 32-bit data
nHW	Writes the upper 16 bits of the <i>n</i> th 32-bit data

### 2.1.1.2.3.3 Cycles for 32-bit DMA Transfers from Main Memory to VRAM

Because main memory and VRAM both have a 16-bit-width bus, data read in half-word units is written in half-word units. While writing in half-word units, the look-ahead buffer reads more data from main memory.

- Basic Cycle

Figure 2-4 shows the basic cycles of the transfer sequence from main memory to VRAM.

**Figure 2-4 : Transfer Sequence from Main Memory to VRAM (Basic Cycles)**

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
VRAM							1LW	1HW		2LW	2HW		3LW
Main memory	wait	wait	wait	wait	1LR	1HR	2LR		2HR	3LR		3HR	4LR

- Worst Case

According to the main memory address where access starts, a wait known as a termination may occur. In these cases, the access immediately after the termination becomes the first access.

Figure 2-5 shows the worst-case transfer sequence from main memory to VRAM.

**Figure 2-5 : Transfer Sequence from Main Memory to VRAM (Worst Case)**

Cycles	1	2	3	4	5	6	7	8	9	10	11	12	13
VRAM							1LW	1HW					
Main Memory	wait	wait	wait	wait	1LR	1HR	T	T	T	wait	wait	wait	wait

14	15	16	17	18	19	20	21	22	23	24	25	26
		2LW	2HW		3LW	3HW		4LW	4HW		5LW	5HW
2LR	2HR	3LR		3HR	4LR		4HR	5LR		5HR	6LR	

wait	Wait
T	Termination
nLR	Reads the lower 16 bits of the <i>n</i> th 32-bit data
nHR	Reads the upper 16 bits of the <i>n</i> th 32-bit data
nLW	Writes the lower 16 bits of the <i>n</i> th 32-bit data
nHW	Writes the upper 16 bits of the <i>n</i> th 32-bit data

#### 2.1.1.2.3.4 Cycles for 32-bit DMA Transfers from VRAM to Main Memory

Figure 2-6 shows the basic cycles of the transfer sequence from VRAM to main memory.

**Figure 2-6 : Transfer Sequence from VRAM to Main Memory (Basic Cycles)**

Cycle	1	2	3	4	5	6	7	8	9	10	11	12	13
VRAM	1LR	1HR						2LR	2HR			3LR	3HR
Main Memory			wait	wait	wait	1LW	1HW			2LW	2HW		

wait	Wait
nLR	Reads the lower 16 bits of the <i>n</i> th 32-bit data
nHR	Reads the upper 16 bits of the <i>n</i> th 32-bit data
nLW	Writes the lower 16 bits of the <i>n</i> th 32-bit data
nHW	Writes the upper 16 bits of the <i>n</i> th 32-bit data

## 2.2 NITRO Processor's Internal Memory

### 2.2.1 VRAM

VRAM (A to I) does not have a fixed use, so it can be assigned for each application in the ways that make the most efficient use of memory resources. This ability is called *VRAM bank control*. Do not switch banks during access to VRAM.

Table 2-8 shows the options for VRAM use.

**Table 2-8 : Options for VRAM Use**

VRAM		A	B	C	D	E	F	G	H	I
Use	KB	128	128	128	128	64	16	16	32	16
LCDC		X	X	X	X	X	X	X	X	X
ARM7				X	X					
2D Graphics Engine A	BG-VRAM	X	X	X	X	X	X	X		
	OBJ-VRAM	X	X			X	X	X		
	BG Extended Palette Slot					X	X	X		
	OBJ Extended Palette Slot						X	X		
2D Graphics Engine B	BG-VRAM			X					X	X
	OBJ-VRAM				X					X
	BG Extended Palette Slot								X	
	OBJ Extended Palette Slot									X
3D Graphics (Rendering Engine)	Texture Image Slot	X	X	X	X					
	Texture Palette Slot					X	X	X		

(BG is screen data or character data. OBJ is character data.)

Memory assigned to LCDC, ARM7, BG-VRAM, and OBJ-VRAM is also mapped to the ARM9 bus, enabling memory to be read and written by ARM9. Memory assigned to the extended palette and texture slots is not mapped to the ARM9 bus.

**Note 1:** LCDC: The LCD controller (LCDC) handles this region. VRAM A to D can be used as memory for holding bitmap data during VRAM display mode and it can also be set as memory for writing bitmap data during captures. (For details, see ["4.4.4.2 VRAM Display Mode"](#) on page 63 and ["4.5 Display Capture"](#) on page 67.)

VRAM assigned to LCD is uniquely mapped to the ARM9 bus. (When all A-to-I of VRAM is assigned to LCD, it is mapped in a contiguous region of the ARM9 bus.) Because there is no access from the CPU when VRAM is allocated to the Extended Palette slot or the Texture Image/Palette slot, you need to temporarily set data in LCD to write data.

**Note 2:** BG-VRAM: This region stores BG screen data, character data, and bitmap data. Up to 512 KB for 2D Graphics Engine A or up to 128 KB for 2D Graphics Engine B can be assigned for this purpose.

**Note 3:** OBJ-VRAM: This region stores OBJ character data and bitmap data. Up to 256 KB for 2D Graphics Engine A or up to 128 KB for 2D Graphics Engine B can be assigned for this purpose.

**Note 4:** Extended palette slots: This memory space is the property of the 2D graphics engine, which references color data when BG and OBJ are displayed. The slots are not mapped in the CPU's memory space.

**Note 5:** Texture image slot and texture palette slot: This memory space is the property of the rendering engine inside the 3D graphics engine. The rendering engine references texel colors when textures are blended. The slots are not mapped in CPU memory space.



**VRAMCNT: RAM Bank Control Register 0**

Name: VRAMCNT

Address: 0x04000240

Attribute: W

Initial Value: 0x00000000

31	28	27	26	24	23	20	19	18	16	15	12	11	9	8	7	4	3	1	0
E			OFS	MST	E			OFS	MST	E			OFS	MST	E			OFS	MST
VRAM-D					VRAM-C					VRAM-B					VRAM-A				

- VRAM-D, VRAM-C

- E[d31][d23] Enable flag

0	Disable
1	Enable

- OFS[d28–d27][d20–d19]: Allocated addresses (Allocated to the addresses shown below according to the MST: Allocation options)

- When MST = 000

Allocated to LCDC, and also mapped to ARM9 memory space

VRAM-C	0x06840000-0x0685FFFF
VRAM-D	0x06860000-0x0687FFFF

- When MST = 001

Allocated to 2D Graphics Engine A's BG, and also mapped to ARM9 memory space

00	0x06000000-0x0601FFFF
01	0x06020000-0x0603FFFF
10	0x06040000-0x0605FFFF
11	0x06060000-0x0607FFFF

- When MST = 010

Mapped to ARM7 memory space, so not mapped to ARM9 memory space

00	0x06000000-0x0601FFFF
01	0x06020000-0x0603FFFF
10	Setting prohibited
11	Setting prohibited

## 4. When MST = 011

Allocated to texture image slot, but not mapped to ARM9 memory space.

(See the texture image slot memory map in Figure 2-7.)

00	Texture image slot 0
01	Texture image slot 1
10	Texture image slot 2 (clear color image)
11	Texture image slot 3 (clear depth image)

## 5. When MST = 100

Mapped to the following ARM9 memory spaces, no matter what the setting.

VRAM-C	0x06200000-0x0621FFFF
VRAM-D	0x06600000-0x0661FFFF

- MST[d26–d24][d18–d16]: Allocation options

000	Allocate to LCDC
001	Allocate to 2D Graphic Engine A's BG
010	Allocate to ARM7
011	Allocate to 3D rendering engine's texture image
100	For VRAM-C: Allocate to 2D Graphic Engine B's BG For VRAM-D: Allocate to 2D Graphic Engine B's OBJ
101-111	Setting prohibited

**Note:** Although VRAM-C and VRAM-D can be allocated to the ARM7 subprocessor, to ensure that the subprocessor API operates correctly, do not change the register settings.

- VRAM-B, VRAM-A

- E[d15][d07]: Enable flag

0	Disable
1	Enable

- OFS[d12–d11][d04–d03]: Allocated addresses (Allocated to the addresses shown below according to the MST: Allocation options)

## 1. When MST = 00

Allocated to LCDC and also mapped to ARM9 memory space

VRAM-A	0x06800000-0x0681FFFF
VRAM-B	0x06820000-0x0683FFFF

## 2. When MST = 01

Allocated to the 2D Graphic Engine A's BG and also mapped to the ARM9 memory space

00	0x06000000-0x0601FFFF
01	0x06020000-0x0603FFFF
10	0x06040000-0x0605FFFF
11	0x06060000-0x0607FFFF

## 3. When MST = 10

Allocated to the 2D Graphic Engine A's OBJ and also mapped to the ARM9 memory space

00	0x06400000-0x0641FFFF
01	0x06420000-0x0643FFFF
10	Setting prohibited
11	Setting prohibited

## 4. When MST = 11

Allocated to the texture image slot but not mapped to ARM9 memory space

(See the texture image slot memory map in "[Figure 2-7 : Texture Image Slot Memory Map](#)" on page 22.)

00	Texture image slot 0
01	Texture image slot 1
10	Texture image slot 2 (clear color image)
11	Texture image slot 3 (clear depth image)

- MST[d09–d08][d01–d00]: Allocation options

00	Allocated to the LCDC
01	Allocated to the 2D Graphic Engine A's BG
10	Allocated to the 2D Graphic Engine A's OBJ
11	Allocated to the 3D rendering engine's texture image

Operation is not guaranteed when multiple VRAM blocks are mapped to the same address or assigned to the same slot.

Texture image slots are memory-mapped in the rendering engine as in Figure 2-7.

**Figure 2-7 : Texture Image Slot Memory Map**

0x00080000	
0x00060000	Slot 3 (clear depth image)
0x00040000	Slot 2 (clear color image)
0x00020000	Slot 1
0x00000000	Slot 0

Texture image Slot 2 and Slot 3 also works as a clear image buffer to initialize the rendering buffer. (Read about initializing with a clear image in "[6.3.3 Initializing the Rendering Buffers](#)" on page 231).

**WVRAMCNT: RAM Bank Control Register 1**

Name: WVRAMCNT

Address: 0x04000244

Attribute: W

Initial value: 0x00000000

31						24	23				20	19	18			16	15				12	11	10			8	7					2			0
						BANK	E				OFS			MST		E				OFS			MST		E						MST				
WRAM							VRAM-G							VRAM-F							VRAM-E														

- VRAM-G, VRAM-F
  - E[d23][d15]: Enable flag
- OFS[d20–d19][d12–d11]: Allocated addresses (Allocated to the addresses shown below according to the MST: Allocation options)

- When MST = 000

Allocated to LCDC and also mapped to ARM9 memory space

VRAM-F	0x06890000-0x06893FFF
VRAM-G	0x06894000-0x06897FFF

- When MST = 001

Allocated to the 2D Graphic Engine A's BG and also mapped to ARM9 memory space

00	0x06000000-0x06003FFF
01	0x06004000-0x06007FFF
10	0x06010000-0x06013FFF
11	0x06014000-0x06017FFF

- When MST = 010

Allocated to the 2D Graphic Engine A's OBJ and also mapped to ARM9 memory space

00	0x06400000-0x06403FFF
01	0x06404000-0x06407FFF
10	0x06410000-0x06413FFF
11	0x06414000-0x06417FFF

## 4. When MST = 011

Allocated to the texture palette slot, but not mapped to ARM9 memory space.

(See the texture palette slot memory map in "[Figure 2-8 : Texture Palette Slot Memory Map](#)" on page 25)

00	Texture palette slot 0
01	Texture palette slot 1
10	Texture palette slot 4
11	Texture palette slot 5

## 5. When MST = 100

Allocated to the 2D Graphic Engine A's BG extended palette slot, but not mapped to ARM9 memory space.

(See the BG extended palette slot memory map in "[Figure 2-9 : BG Extended Palette Slot Memory Map](#)" on page 25)

00	2D Graphic Engine A BG extended palette slots 0-1
01	2D Graphic Engine A BG extended palette slots 2-3
10	Setting prohibited
11	Setting prohibited

## 6. When MST = 101

Allocated to the 2D Graphic Engine A's OBJ extended palette slot, but not mapped to ARM9 memory space.

The lower 8 KB are allocated to the slot, but the upper 8 KB are invalidated.

See the OBJ extended palette slot memory map in "[Figure 2-10 : OBJ Extended Palette Slot Memory Map](#)" on page 26.

- MST[d18–d16][d10–d08]: Allocation options

000	Allocated to the LCDC
001	Allocated to the 2D Graphic Engine A's BG
010	Allocated to the 2D Graphic Engine A's OBJ
011	Allocated to the 3D Rendering Engine's texture palette
100	Allocated to the 2D Graphic Engine A's BG extended palette
101	Allocated to the 2D Graphic Engine A's OBJ extended palette
110, 111	Setting prohibited

- VRAM-E
  - E[d07]: Enable flag

0	Disable
1	Enable

- MST[d02–d00]: Allocation options

000	Allocated to the LCDC
001	Allocated to the 2D Graphic Engine A's BG
010	Allocated to the 2D Graphic Engine A's OBJ
011	Allocated to the 3D Rendering Engine's texture palette
100	Allocated to the 2D Graphic Engine A's BG extended palette
101-111	Setting prohibited

**Note:** VRAM-E mapping is fixed according to the MST settings shown below. (The offset cannot be changed.)

000	ARM9 addresses 0x06880000-0x0688FFFF
001	ARM9 addresses 0x06000000-0x0600FFFF
010	ARM9 addresses 0x06400000-0x0640FFFF
011	Texture palette slots 0-3
100	BG extended palette slots 0-3 (only the lower 32 KB)

The texture palette slots are mapped in the Rendering Engine (see Figure 2-8) when VRAM blocks E, F, and G are assigned. The BG palette slots are mapped as shown in Figure 2-9.

**Figure 2-8 : Texture Palette Slot Memory Map**

0x00018000	
0x00014000	Slot 5
0x00010000	Slot 4
0x0000C000	Slot 3
0x00008000	Slot 2
0x00004000	Slot 1
0x00000000	Slot 0

**Figure 2-9 : BG Extended Palette Slot Memory Map**

0x00008000	
0x00006000	Slot 3
0x00004000	Slot 2
0x00002000	Slot 1
0x00000000	Slot 0

Figure 2-10 shows the memory map when VRAM-F and -G are allocated to the OBJ extended palette.

**Figure 2-10 : OBJ Extended Palette Slot Memory Map**

0x00004000	
0x00002000	Slot 1
0x00000000	Slot 0

Proper operation is not guaranteed when multiple VRAM blocks are mapped to the same address or assigned to the same slot.

### VRAM\_HI\_CNT: RAM Bank Control Register 2

Name: VRAM\_HI\_CNT

Address: 0x04000248

Attribute: W

Initial value: 0x0000

15						10	9	8		7						2	1	0
E								MST		E								MST
VRAM-I										VRAM-H								

- VRAM-I, VRAM-H
  - E[d15][d07]: Enable flag

0	Disable
1	Enable

- MST[d09–d08][d01–d00]: Allocation options
  - For VRAM-H

00	Allocated to LCD C Mapped to main processor addresses 0x06898000-0x0689FFFF
01	Allocated to 2D Graphics Engine B's BG Mapped to main processor addresses 0x06200000-0x06207FFF
10	Allocated to 2D Graphics Engine B's BG extended palette Mapped to slots 0-3
11	Setting prohibited

- For VRAM-I

00	Allocated to LCD C Mapped to main processor addresses 0x068A0000-0x068A3FFFF
01	Allocated to 2D Graphics Engine B's BG Mapped to main processor addresses 0x06208000-0x0620BFFF
10	Allocated to 2D Graphics Engine B's OBJ Mapped to main processor addresses 0x06600000-0x06603FFF
11	Allocated to 2D Graphics Engine B's OBJ extended palette Mapped to slots 0-1

Table 2-9 through Table 2-14 show the addresses allocated to the various VRAM blocks for the given MST and OFS bits.



**Table 2-9 : VRAM-A and VRAM-B Allocations**

MST	Allocation	OFS	00	01	10	11
00	ARM9		VRAM-A: 0x06800000-0x0681FFFF VRAM-B: 0x06820000-0x0683FFFF			
01	2D Graphics Engine A		BG-VRAM (0x06000000-0x0601FFFF)	BG-VRAM (0x06020000-0x0603FFFF)	BG-VRAM (0x06040000-0x0605FFFF)	BG-VRAM (0x06060000-0x0607FFFF)
10	2D Graphics Engine A		OBJ-VRAM (0x06400000-0x0641FFFF)	OBJ-VRAM (0x06420000-0x0643FFFF)	Setting prohibited	
11	Texture Image		Slot 0	Slot 1	Slot 2	Slot 3
					(clear image)	

**Table 2-10 : VRAM-C and VRAM-D Allocations**

MST	Allocation	OFS	00	01	10	11
000	ARM9		VRAM-C: 0x06840000-0x0685FFFF VRAM-D: 0x06860000-0x0687FFFF			
001	2D Graphics Engine A		BG-VRAM (0x06000000-0x0601FFFF)	BG-VRAM (0x06020000-0x0603FFFF)	BG-VRAM (0x06040000-0x0605FFFF)	BG-VRAM (0x06060000-0x0607FFFF)
010	ARM7		ARM7 0x06000000-0x0601FFFF	ARM7 0x06020000-0x0603FFFF	Setting prohibited	
011	Texture Image		Slot 0	Slot 1	Slot 2	Slot 3
					(clear image)	
100	2D Graphics Engine B		VRAM-C: BG-VRAM (0x06200000-0x0621FFFF) VRAM-D: OBJ-VRAM (0x06600000-0x0661FFFF)			
101-111	Setting Prohibited		Setting prohibited			

**Table 2-11 : VRAM-E Allocations**

MST	Allocation	Address
000	ARM9	0x06880000-0x0688FFFF
001	2D Graphics Engine A	BG-VRAM (0x06000000-0x0600FFFF)
010	2D Graphics Engine A	OBJ-VRAM (0x06400000-0x0640FFFF)
011	Texture Palette	Texture palette slot 0-3
100	2D Graphics Engine A	BG extended palette slots 0-3 (only the lower 32 KB are valid)
101-111	Setting Prohibited	Setting prohibited

**Table 2-12 : VRAM-F and VRAM-G Allocations**

MST	Allocation	OFS	00	01	10	11
000	ARM9		VRAM-F: 0x06890000-0x06893FFF VRAM-G: 0x06894000-0x06897FFF			
001	2D Graphics Engine A		BG-VRAM (0x06000000-0x06003FFF)	BG-VRAM (0x06004000-0x06007FFF)	BG-VRAM (0x06010000-0x06013FFF)	BG-VRAM (0x06014000-0x06017FFF)
010	2D Graphics Engine A		OBJ-VRAM (0x06400000-0x06403FFF)	OBJ-VRAM (0x06404000-0x06407FFF)	OBJ-VRAM (0x06410000-0x06413FFF)	OBJ-VRAM (0x06414000-0x06417FFF)
011	Texture Palette		Slot 0	Slot 1	Slot 4	Slot 5
100	2D Graphics Engine A		BG extended palette slots 0-1	BG extended palette slots 2-3	Setting prohibited	
101	2D Graphics Engine A		OBJ extended palette slot 0 (only lower 8 KB are valid)			
110	Setting Prohibited		Setting prohibited			
111	Setting Prohibited					

**Table 2-13 : VRAM-H Allocations**

MST	Allocation	Address
00	ARM9	0x06898000-0x0689FFFF
01	2D Graphics Engine B	BG-VRAM (0x06200000-0x06207FFF)
10	2D Graphics Engine B	BG extended palette slots 0-3
11	Setting Prohibited	Setting prohibited

**Table 2-14 : VRAM-I Allocations**

MST	Allocation	Address
00	ARM9	0x068A0000-0x068A3FFFF
01	2D Graphics Engine B	BG-VRAM (0x06208000-0x0620BFFF)
10	2D Graphics Engine B	OBJ-VRAM (0x06600000-0x06603FFF)
11	2D Graphics Engine B	OBJ extended palette slots 0-1

## 2.2.2 Work RAM

Work RAM does not have a fixed use, so it can be assigned for each application in the ways that make the most efficient use of memory resources. This ability is called *WRAM bank control*.

Be sure not to switch banks while Work RAM is being accessed.

### WVRAMCNT: RAM Bank Control Register 1

Name: WVRAMCNT    Address: 0x04000244    Attribute: W    Initial value: 0x00000000

31						25	24	23				20	19	18	16	15				12	11	10	8	7					2	0	
							BANK	E				OFS		MST		E				OFS		MST		E						MST	
WRAM								VRAM-G								VRAM-F								VRAM-E							

- WRAM
  - BANK[d25–d24]: Bank Specification  
Selects whether to allocate 16 KB x 2 blocks to ARM9 or ARM7.

Figure 2-11 shows the memory maps for each value setting.

	ARM9	ARM7
00	32KB	None
01	16KB (Block1)	16KB (Block0)
10	16KB (Block0)	16KB (Block1)
11	None	32KB

**Note:** Though you can change the Work RAM settings, to ensure that the subprocessor API operates correctly, do not change the register settings.

**Figure 2-11 : Memory Maps for Various Settings of ARM9, ARM7 Shared Internal Work RAM****ARM9 Memory Map****Allocation for Entire Region (32KB)**

0x04000000	I/O registers
0x03800000	Image of ARM9, ARM7 shared internal Work RAM
0x037F8000	ARM9, ARM7 shared internal Work RAM (32KB)
0x03000000	Image of ARM9, ARM7 shared internal Work RAM
0x02400000	Main memory image

**Block 1 Allocation (16KB)**

0x04000000	I/O registers
0x03800000	Image of ARM9, ARM7 shared internal Work RAM (block 1)
0x037FC000	ARM9, ARM7 shared internal Work RAM (block 1: 16KB)
0x03000000	Image of ARM9, ARM7 shared internal Work RAM (block 1)
0x02400000	Main memory image

**Block 0 Allocation (16KB)**

0x04000000	I/O registers
0x03800000	Image of ARM9, ARM7 shared internal Work RAM (block 0)
0x037FC000	ARM9, ARM7 shared internal Work RAM (block 0: 16KB)
0x03000000	Image of ARM9, ARM7 shared internal Work RAM (block 0)
0x02400000	Main memory image

**No Allocation**

0x04000000	I/O registers
0x03800000	Indeterminate data
0x03000000	
0x02400000	Main memory image

**ARM7 Memory Map****No Allocation**

0x04000000	I/O registers
0x03810000	Image of ARM7 dedicated internal Work RAM
0x03800000	ARM7 dedicated internal Work RAM (64KB)
0x03000000	Image of ARM7 dedicated internal Work RAM (32KB)
0x02400000	Main memory image

**Block 0 Allocation (16KB)**

0x04000000	I/O registers
0x03810000	Image of ARM7 dedicated internal Work RAM
0x03800000	ARM7 dedicated internal Work RAM (64KB)
0x037FC000	Image of ARM9, ARM7 shared internal Work RAM (block 0: 16KB)
0x03000000	Image of ARM9, ARM7 shared internal Work RAM (block 0)
0x02400000	Main memory image

**Block 1 Allocation (16KB)**

0x04000000	I/O registers
0x03810000	Image of ARM7 dedicated internal Work RAM
0x03800000	ARM7 dedicated internal Work RAM (64KB)
0x037FC000	Image of ARM9, ARM7 shared internal Work RAM (block 1: 16KB)
0x03000000	Image of ARM 9, ARM7 shared internal Work RAM (block 1)
0x02400000	Main memory image

**Allocation for Entire Region (32KB)**

0x04000000	I/O registers
0x03810000	Image of ARM7 dedicated internal Work RAM
0x03800000	ARM7 dedicated internal Work RAM (64KB)
0x037F8000	ARM9, ARM7 shared internal Work RAM (32KB)
0x03000000	Image of ARM9, ARM7 shared internal Work RAM
0x02400000	Main memory image

### 2.2.3 I/O Registers

See the appendices to learn about the mapping for each I/O register.

- Accessing undefined registers

Table 2-15 shows the behaviors that occur when an undefined address is accessed in the I/O register regions.

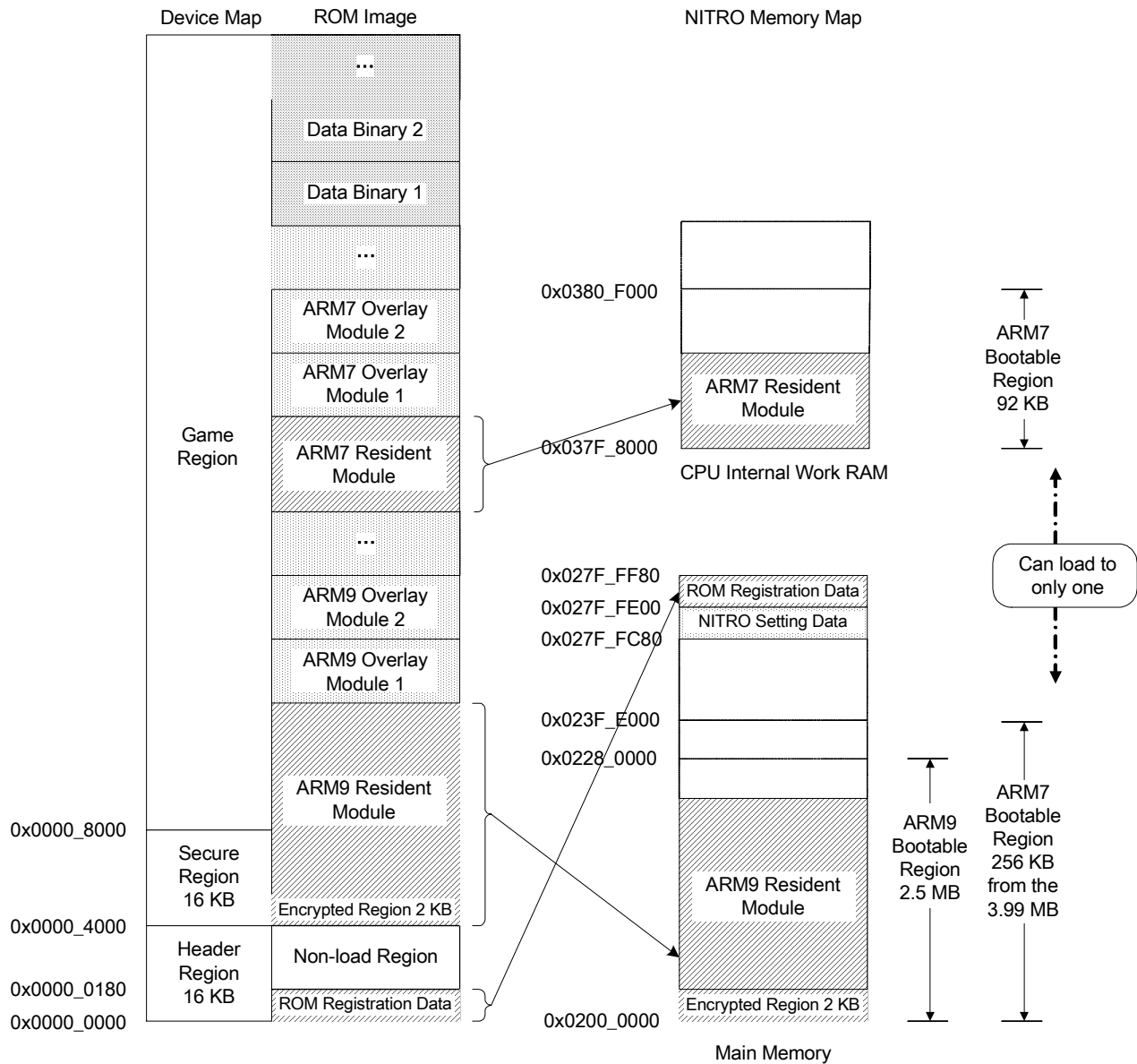
**Table 2-15 : Result of Accessing an Undefined Register**

Access Destination	Write	Read
Undefined Register	Invalid	ALL zero

## 2.3 Memory Map for Game Card Boot

Figure 2-12 shows the memory map when the system boots from a Game Card.

**Figure 2-12 : Memory Map for Game Card Boot**



The following are details about the regions shown in the memory map:

- **Resident Module:**  
Loaded at application startup time.
- **Overlay Module:**  
Loaded by the application itself.
- **Data Binary**  
Loaded by the application itself.  
Not linked; loaded by referencing FAT.

- **The Header Region:**  
Stores the ROM registration data. This data is loaded into the System region of main memory at boot time.  
Cannot be reloaded after application startup.  
(After the Game Code is determined) Provides a binary file called a ROM header template for each application.
  - **The Secure Region:**  
Stores the starting part of the ARM9 Resident Module. This is loaded at boot time to the address in main memory specified by the registration data in ROM.  
Must be located within the first 64 KB from the start of main memory, for security reasons.  
Decodes the first 2 KB, an encrypted region, when loaded.  
    To be specific, the encrypted system call library is linked to the start when the ARM9 resident module is created.  
    Because this cannot be reloaded after application startup, only the `.text` and `.rodata` sections should be located in the secure region. Otherwise, you cannot perform a software reset.
  - **The Game Region:**  
Loads (at boot time) the part of the ARM9 Resident Module that follows the security region and the ARM7 Resident Module to the address specified by the registration data in ROM.  
Set the bootable size for ARM9 so that it does not exceed 2.5 MB from the beginning of main memory. For ARM7, the bootable region should not exceed 256 KB within the 3.99 MB region from the beginning of main memory or should not exceed 92 KB from the beginning of the CPU internal work RAM.  
This region is always read-enabled. Load the Overlay Module and Data Binary as needed with the application.
  - **The NITRO Settings Data:**  
Contains owner information and other data, stored in internal flash memory.  
    If this data is available, it is loaded by the IPL to address 0x027F\_FC80 at startup.  
    For details about the data structure, see Chapter "[18 Internal Flash Memory](#)" on page 323.
- Note:** Refer to the Nintendo DS Game Card Manual for the ROM Internal Registration Data.

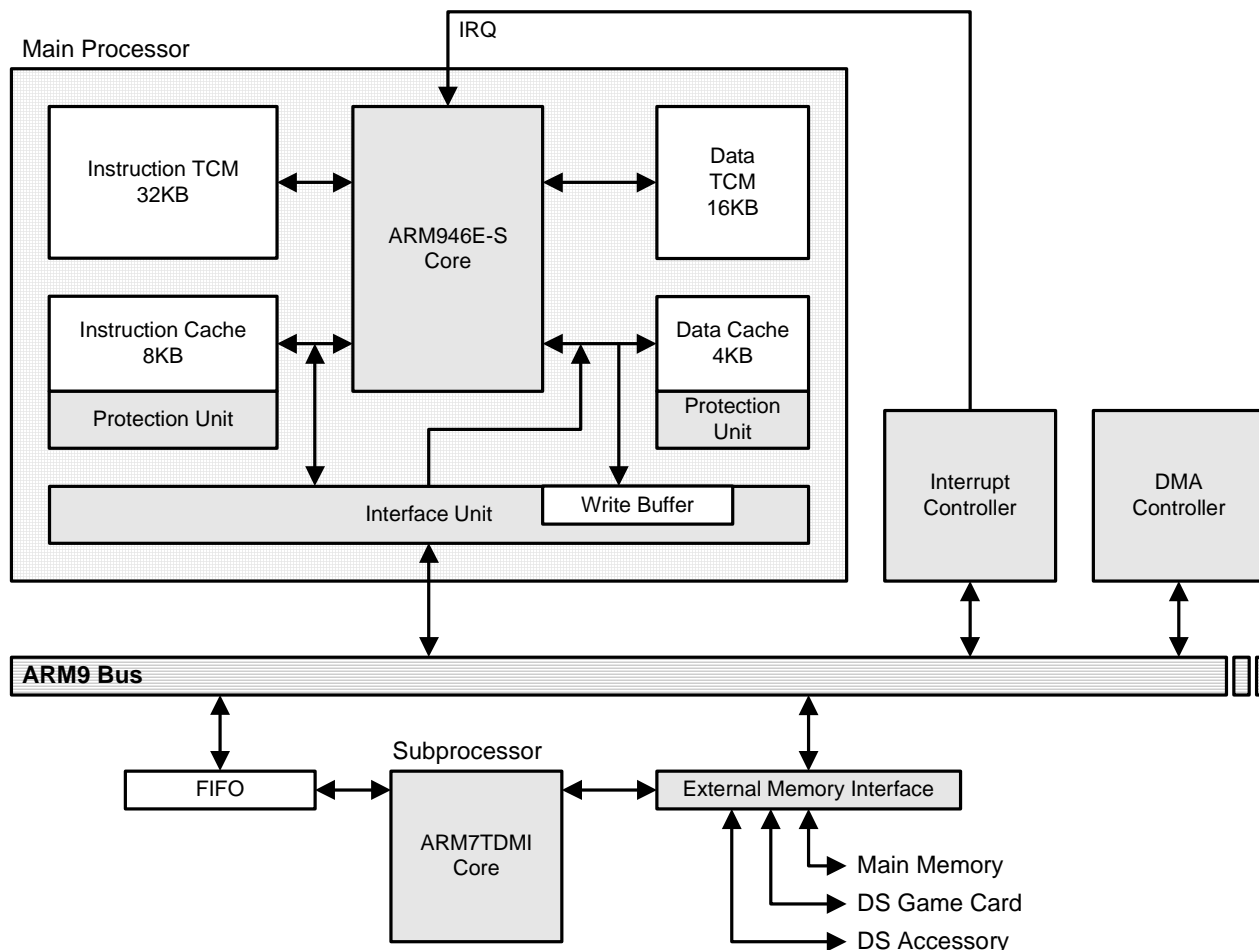




### 3 Main Processor Core (ARM946E-S)

Figure 3-1 is a block diagram of the main processor core.

Figure 3-1 : Block Diagram of the Main Processor Core



#### 3.1 Protection Unit

The *Protection Unit* protects memory by configuring the read/write attributes and enabling/disabling the use of the cache and Write buffers in each protection region.

Up to eight memory regions can be configured from the 4 GB of background (the entire address space accessible from the CPU). Protection regions with higher region numbers have higher priority. (If regions have duplicate protection region settings, the setting with the higher protection region number is given priority.) Each protection region can be configured separately to enable/disable the cache and Write buffers and allow/disallow reading/writing.

- **Debug Version and Release Version**

In addition to the production version of NITRO, there is a debug version that comes with the main memory expanded to 8 MB. Though part of the additional 4 MB of main memory is used as the debugger region, the remainder can be used during application development. Because the debug and release versions of NITRO have different memory structures, the Protection Unit settings are also different.

## 3.2 Tightly Coupled Memory (TCM)

TCM is high-speed memory that is directly connected to the ARM9 core. It can be used as independent Work RAM. Because TCM does not use the ARM9 bus, the ARM9 can use TCM to conduct processes even while the ARM9 bus is being used (such as by DMA).

As a result, TCM can boost performance by storing frequently used programs and data that you want to access during DMA transfers.

There are two types of TCM: one for instructions and one for data. Note that TCM cannot be accessed via DMA.

### 3.2.1 Instruction TCM

This is 32 KB of high-speed memory, mapped from ARM9 address 0x01000000.

Instruction TCM is a good place to hold fast-running programs or programs for which you want to fix the operation clock count. Examples include graphic libraries and routines that branch on interrupts.

Because Instruction TCM does not require the ARM9 bus while fetching instructions, Instruction TCM is also an effective place to store programs you want to execute while DMA is using the ARM9 bus. Examples include routines that generate display lists and computation routines.

You can store data in Instruction TCM memory also, but stalls occur when data access collides with instruction fetches.

### 3.2.2 Data TCM

This is 16 KB of high-speed memory. Data TCM can be set to any address location in the memory map.

Data TCM is a good place to store data for fast reading and writing. Examples include stacks and frequently accessed tables.

Because Data TCM does not require the ARM9 bus during access, Data TCM can be used for creating the next display list even while the current display list is being transferred via DMA from main memory to the geometry engine.

However, transfers from TCM must be conducted via the ARM9 core. In addition, instructions cannot be placed in Data TCM. But that means there is no stalling of data access due to collisions with instruction fetches.

## 3.3 Cache Memory

When the ARM9 bus references memory, it takes 32 bytes of data from that vicinity (the data in a range corresponding to the upper 27 bits of the referenced address) and loads that data into the cache so the data is accessible at the fast speeds of cache memory the next time that range is referenced.

If the ARM9 hits data that is in the cache, the data can be read quickly.

If the ARM9 does not hit data in the cache, the contents of memory are read in units of cache lines (line fetching). During this process, the contents of the cache lines are replaced according to the replacement algorithm.

Note that the Protection Unit must be enabled to use the cache.

Table 3-1 lists the specifications for the cache.

**Table 3-1 : Cache Specifications**

<b>Capacity</b>	Instruction cache: 8 KB Data cache: 4 KB
<b>Configuration</b>	4-way set associative method
<b>Cache Line</b>	8 words (32 bytes)
<b>Write-miss Operation</b>	Read-allocate method (see note 1)
<b>Replacement Algorithm</b>	Can choose either round-robin (see note 2) or pseudo-random (see note 3)
<b>Bus Snoop Feature (see note 4)</b>	None
<b>Other Features (Instruction Cache)</b>	Lockdown Instruction prefetch
<b>Other Features (Data Cache)</b>	Lockdown Write-through mode / Write-back mode

**Note 1:** Read-allocate methods

In this method, when there is a write miss, the write is performed only for the memory (or for the Write buffer, if it is enabled). Data is not loaded to the cache. (For the Write-allocation method, a write miss is handled as a write hit after data is loaded into the cache.)

**Note 2:** Round-robin (recommended)

In this method, cache lines are replaced in order. Performance is stable in the worst-case scenario.

**Note 3:** Pseudo-random

In this method, cache lines are replaced randomly. This increases peak-time performance, but lowers worst-case performance.

**Note 4:** Bus Snoop feature

By monitoring the bus, this feature detects whether another processor or DMA writes to the memory region stored in the cache. Because the ARM946E-S does not have this feature, you must be careful that there is coherency between memory and cache. (See ["3.5 Ensuring Coherency"](#) on page 43.)

**3.3.1 Instruction Cache**

This is 8 KB of high-speed memory, dedicated to instruction code.

The ARM9 bus is not used during cache hits, so a program in the Instruction cache can run even when a non-ARM9 bus master (DMA or subprocessor) has possession of the ARM9 bus.

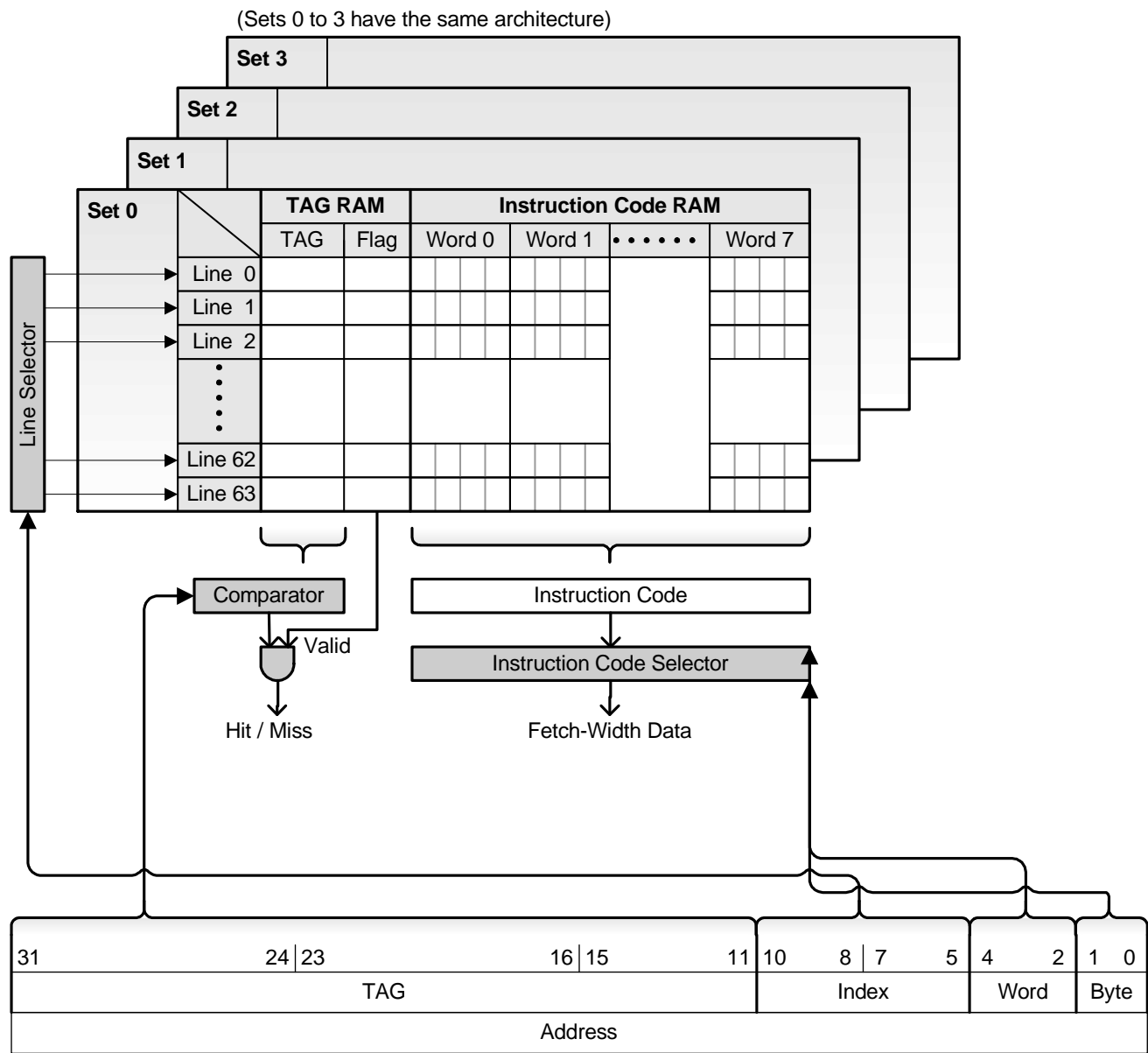
**3.3.1.1 Determining Hits and Misses**

When the ARM9 bus fetches instruction code from memory, the Instruction Cache Controller extracts the Index bit and TAG bit from the memory addresses and compares the contents of TAG RAM in the  $n$ th cache line (defined by the Index number) with the TAG bit at that memory address. Because the ARM946E-S cache has a four-set structure, the comparison is made on four cache lines. If the comparison finds a match for any of these four cache lines and the valid bit is enabled, it is determined that the cache line contains the targeted instruction code (a hit). If not, it is considered a miss.

For a cache hit, the intended instruction code in Data RAM is identified from the address' Word or Byte and can be accessed quickly. For a miss, the cache lines are fetched from memory via the ARM9 bus.

Figure 3-2 shows the structure and actions of the Instruction cache.

Figure 3-2 : Structure and Actions of the Instruction Cache



Address of instruction that ARM9 will fetch

About Instruction Cache TAG RAM

- TAG  
The upper 21 bits of the memory address of the data in the cache line are stored here.
- Flag  
Generated from the *valid* bit, which indicates whether cache lines are enabled or disabled.

3.3.2 Data Cache

This is 4 KB of high-speed memory. It is data-only.

The ARM9 bus is not used during cache hits, so the data in this cache can be accessed even when a non-ARM9 bus master (DMA or subprocessor) has possession of the ARM9 bus.

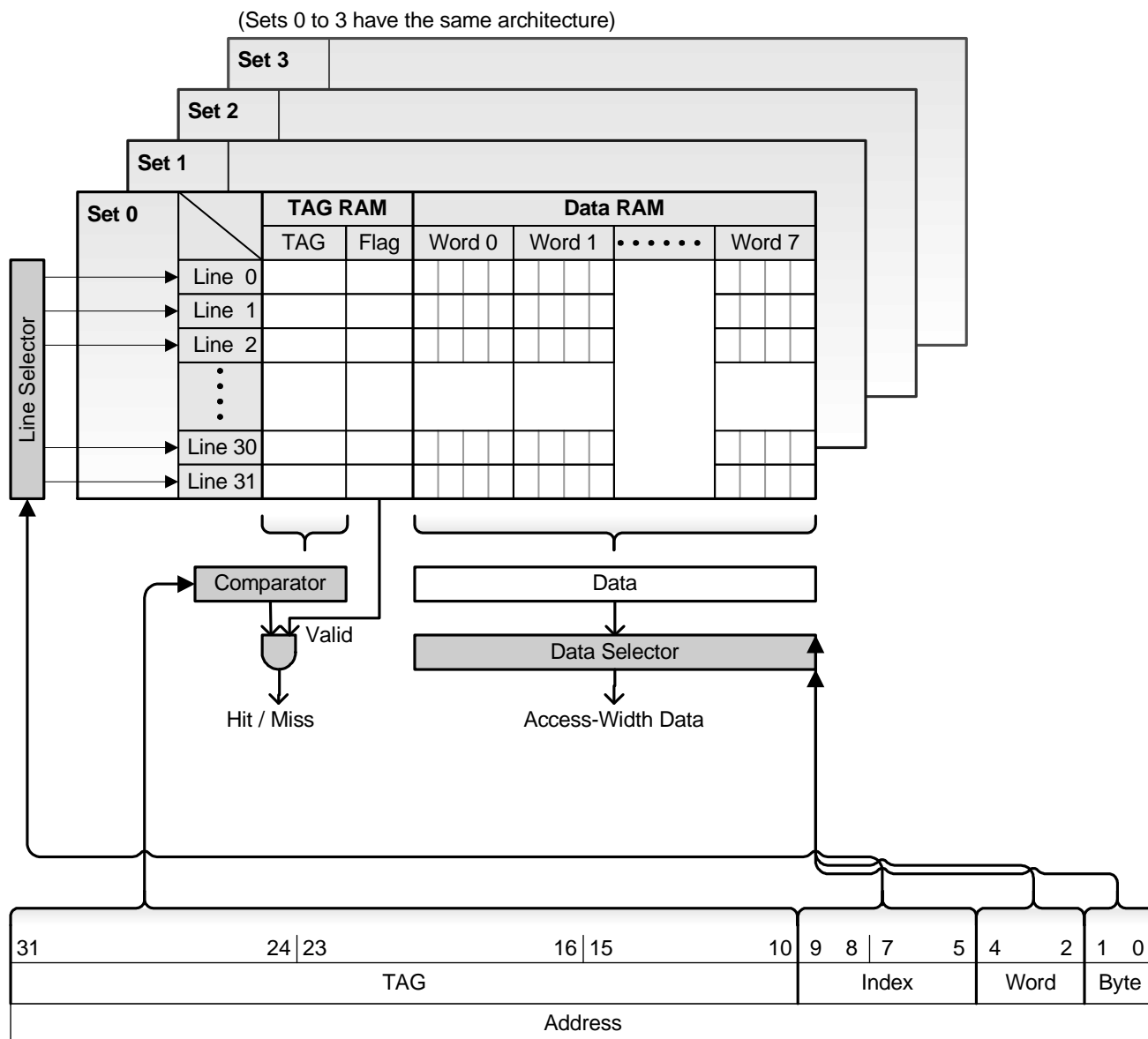
### 3.3.2.1 Determining Hits and Misses

When ARM9 loads data into memory, the Data Cache Controller extracts the Index bit and TAG bit from the memory address and compares the contents of TAG RAM in the  $n$ th cache line (defined by the index number) with the TAG bit at the memory address. The ARM946E-S cache has a four-set structure, so the comparison is made on four cache lines. If the comparison finds a match for any of these four cache lines and the valid bit is enabled, it is determined that the cache line contains the targeted data (hit). If not, it is considered a miss.

For a cache hit, the targeted data in data RAM is identified from the address' Word and Byte and can be quickly accessed. For a miss, the data is accessed from memory via the ARM9 bus.

Figure 3-3 shows the structure and actions of the Data cache.

**Figure 3-3 : Structure and Actions of the Data Cache**



**Address of data that ARM9 will read/write**

(Differs from Instruction cache in that the Index is 5 bits)

## About Data Cache TAG RAM

- TAG

The upper 22 bits of the memory address of the data in the cache line are stored here.

- Flag

Generated from the *valid* bit, which indicates whether cache lines are enabled or disabled, the *dirty* bit for the first half of the line, and the *dirty* bit for the second half of the line, which indicate whether the contents of the cache and memory are the same.

When the contents of the cache match those in memory, the cache is said to be *clean*. When the contents do not match, the cache is *dirty*.

The dirty bits are used only in write-back mode. To read more about this mode, see ["3.4 Write Buffer"](#) on page 42.

The dirty bits are referenced when the contents of a line are to be written back to memory. If the flag is dirty, a write-back occurs. If the flag is clean, no write-back occurs (because it is unnecessary).

### 3.3.3 Cache Operations

Table 3-2 shows the operations conducted on the cache. Some operations act on the entire cache, while other operations act on specified cache sets, specified cache lines, or a particular cache line specified by a memory address.

**Table 3-2 : Cache Operations**

Cache Operation	Instruction Cache	Data Cache
<b>Overall Direct Operations</b>	- Enable/Disable - Invalidate	- Enable/Disable - Invalidate
<b>Direct Operations on Sets</b>	- Lockdown	- Lockdown
<b>Direct Operations on Lines</b>	---	- Clean - Clean & Invalidate
<b>Operations on Cache Lines Corresponding to Memory Addresses</b>	- Prefetch  - Invalidate	- Clean - Invalidate - Clean & Invalidate

The *Clean*, *Invalidate*, and *Clean & Invalidate* operations can cause changes in the usage state of cache. To read more about these changes, see ["3.5 Ensuring Coherency"](#) on page 43.

#### About Each Operation

- Clean

Writes back dirty data in the cache to memory. The data remains in the cache. If the Write buffer is enabled, the actual write back to memory is delayed.

- Invalidate

Invalidates the data in the cache. The next time this memory region is read, a read miss occurs and a line fetch from memory to the cache is performed. The data moved into the cache by the line fetch is treated as valid.

- Clean and Invalidate

Writes back dirty data in the cache to memory and invalidates the data in the cache. As a result, the next time this memory region is accessed, the data will be line-fetched from memory to the cache.

**Caution:** If the write buffer (refer to ["3.4 Write Buffer"](#) on page 42 for details) is full and the Clean and Invalidate command is issued, caches that are already clean will not be invalidated.

- **Instruction Prefetch (prefetch)**  
Takes instruction code that has not yet been fetched by the program and preloads it into the Instruction cache. It uses coprocessor instructions.
- **Lockdown**  
By locking down one set of cache, no line in that set can be replaced by line fetches and the whole set can be used as a block of Work RAM. However, this reduces the cache region by the same amount and increases the miss rate.
- **Enable/Disable**  
When using the cache, in addition to enabling the cache, you must also enable the Protection Unit.

You can enable/disable the Instruction cache and the Data cache for each protection region. To read about protection regions and their settings, see ["3.1 Protection Unit"](#) on page 35.

**Caution:** Data Preload (preload) is a feature that preloads data that has not been accessed by the program and normally operates by using the PLD instruction. However, due to ARM946E-S specifications, no operation will be performed even when a PLD instruction is recognized. Therefore, the data preload feature does not work with ARM946E-S.

### 3.3.4 Optimizing the Cache

The cache uses the memory reference locality of most programs to accelerate memory access.

*Temporal locality*      The high probability that data, once referenced, will be referenced again soon.

*Spatial locality*        The high probability that data near referenced data will also be referenced.

Programs with a higher reference locality have higher cache hit rates and faster average access speeds.

Depending on how a program is pieced together, you can manually increase the reference locality to some extent. For example, when constructing a loop to handle a two-dimensional (or higher) array, you can boost the spatial locality of the array by handling addresses in consecutive order.

#### Examples:

Loop Example A	Loop Example B
<pre>for( j=0; j&lt;0x100; j++) {     for( i=0; i&lt;0x100; i++) {         RESULT += TEST[i, j];     } }</pre>	<pre>for( i=0; i&lt;0x100; i++) {     for( j=0; j&lt;0x100; j++) {         RESULT += TEST[i, j];     } }</pre>

In loop example A, the TEST array is referenced every 0x100 addresses inside the loop, so there is no cache hit during the first iteration. Lines are fetched one after another in the first iteration. However, because the capacity of the Data cache is 32 lines x 4 sets for a total of 128 lines (0x80 lines), the entire loop does not fit in the Data cache, and hits occur only half the time in the second and subsequent iterations.

In contrast, in loop example B, the TEST array is referenced in the same order as the addresses inside the loop, enabling the maximum hit rate.

### 3.4 Write Buffer

The Write buffer is 32 bytes x 16 layers of FIFO memory that integrates addresses and data.

The type of entry is determined by the address/data flag. Address entries have an appended data size.

Writing data to the high-speed Write buffer instead of memory keeps ARM9 from stalling during the write process. However, ARM9 will stall when writing to the Write buffer when it is full.

When the Protection Unit is enabled, you can select among the access modes shown in Table 3-3 for writing data. The selection is made by configuring the Data cache and the Write buffer for each protection region as shown.

To read more about protection regions, see ["3.1 Protection Unit"](#) on page 35.

**Table 3-3 : Access Modes When the Data is Being Written**

Data Cache Setting	Write Buffer Setting	Access Mode
Disable	Disable	NCNB mode <sup>a</sup> (Data cache and Write buffer are both disabled)
Disable	Enable	NCB mode (Data cache disabled; Write buffer enabled)
Enable	Disable <sup>b</sup>	Write-through mode
Enable	Enable	Write-back mode

a. In NCNB mode, the contents of the write buffer are output when writing and are accessed ahead of the Write buffer output when reading.

b. In write-through mode, the Write buffer is disabled, but it is used.

- Write-back mode (recommended)

If there is a hit during the data write, data is written only to the cache and not to the Write buffer. Therefore, the contents of the cache may not be the same as in memory, but writing is fast.

A *Clean* operation must be performed in order for the data rewritten by the CPU to be reflected in memory.

Further, when a read-miss occurs while the cache is full and the cache lines to be emptied by the replacement algorithm are dirty, they are written back to the Write buffer.

- Write-through mode

If there is a hit during the data write, the data is written to the Write buffer at the same time that it is written to the cache. Therefore, the cache line does not become dirty as a result of writing to ARM9, but writing is slow.

Further, if the cache is full when a read-miss occurs, the replacement algorithm overwrites the cache line.

In both of these modes, data is written only to the Write buffer when a write-miss occurs.

Also, if a line is being fetched, the contents of the Write buffer are discharged first to maintain data coherency.

**Note:** Be careful about the access width of memory when writing in write-through mode. (For example, you cannot use an access width of 8 bits with VRAM.) To read about the access width for each memory type, see ["2 Memory"](#) on page 9.

To read about cache state transitions and control in either of these modes, see ["3.5 Ensuring Coherency"](#) on page 43.



### 3.4.1 Write Buffer Operations

- Wait for the Write buffer to empty

The ARM9 bus can stall until all data in the Write buffer is written to memory.

Use this operation to make certain that all content rewritten by the CPU is reflected in memory.

Note that this operation is not necessary when data is written to I/O registers or other regions where cache and buffers are disabled because, in these cases, the CPU stops until the Write buffer is empty.

To read why this operation is necessary in other cases, see ["3.5 Ensuring Coherency"](#) on page 43.

## 3.5 Ensuring Coherency

Be careful when using the cache to make sure no inconsistencies arise between the contents of the cache and memory.

The cache state is managed in each cache line by the flag in TAG RAM.

This flag includes one *valid* bit and two *dirty* bits. The dirty bits indicate the state of the first half and second half of the cache line. All three bits are used for the Data cache in the write-back mode, but only the valid bit is used in the write-through mode or for the Instruction cache.

### 3.5.1 Write-Back Mode

Table 3-4 shows how the flag states define the cache line state in write-back mode.

**Table 3-4 : Cache Line States (Write-Back Mode)**

State	Flag State		Explanation
	Valid	Dirty	
<b>Dirty</b>	1	1	Cache line is valid, contents differ from memory
<b>Clean</b>	1	0	Cache line is valid, contents match memory
<b>Invalid</b>	0	*	Cache line is invalid

### Operations Managed Automatically by the Cache Controller

As shown in Figure 3-4, read misses/hits and write misses/hits on access from the ARM9 as well as state transitions by the replacement algorithm are performed automatically.

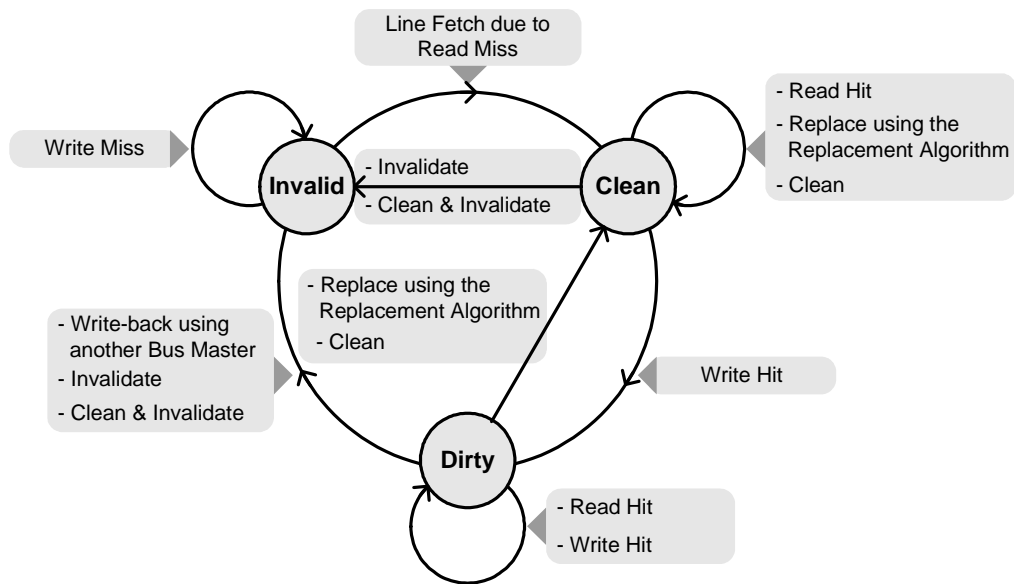
When the replacement algorithm replaces valid, but dirty, lines, the lines are first written back to memory (or to the Write buffer, if enabled). Because the process is actually conducted on the first and second halves of the line, and not on the entire line, the volume of data written back can be 0, 16, or 32 bytes.

### Operations that Must be Managed by the User

Because the ARM946E-S lacks the bus snoop feature, when cached memory is accessed by a bus master other than ARM9 (such as the subprocessor or DMA), the cache lines must be operated manually.

When data are written to memory by a bus master other than ARM9, invalidate the appropriate cache lines.

Also, when memory is read by a bus master other than ARM9, you should clean the cache line beforehand and perform the Wait for Write buffer to empty operation.

**Figure 3-4 : Cache Line State Transitions (Write-Back Mode)****Example**

- If the program in main memory is overwritten by an overlay, etc.  
Invalidate the Instruction cache in the appropriate region.

**3.5.2 Write-Through Mode**

Table 3-5 shows how the flag states define the cache line state in write-through mode.

**Table 3-5 : Cache Line States (Write-Through Mode)**

State	Flag State		Explanation
	Valid	Dirty	
Clean	1	*	Cache line is valid, contents match memory
Invalid	0	*	Cache line is invalid

**Operations Managed Automatically by the Cache Controller**

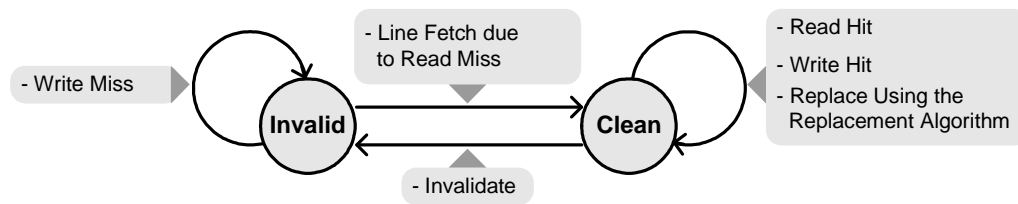
As shown in Figure 3-5, read misses/hits and write misses/hits for access from the ARM9 as well as state transitions by the replacement algorithm are performed automatically.

**Operations that Must be Managed by the User**

Because the ARM946E-S lacks the bus snoop feature, when cached memory is accessed by a bus master other than ARM9 (such as the subprocessor, DMA, etc.), the cache lines must be operated manually.

When data are written to memory by a bus master other than ARM9, you should invalidate the appropriate cache lines.

Also, when memory is read by a bus master other than ARM9, perform the Wait for Write buffer to empty operation.

**Figure 3-5 : Cache Line State Transitions (Write-Through Mode)****Example**

- If the program in main memory is overwritten by an overlay, etc.  
Invalidate the Instruction cache in the appropriate region.



## 4 Display

### 4.1 Display System

The display system block diagram is shown in "[Figure 4-1 : Display System Block Diagram](#)" on page 48.

Each selector in the block diagram can be controlled using the register selection flags in Table 4-1.

**Table 4-1 : Selector and Register Selection Flag Map**

Selector Name	Register Name	Flag Name
SEL DISP	DISPCNT	Display mode selection
SEL BG0	DISPCNT	2D/3D display selection for BG0
SEL DISP VRAM	DISPCNT	Display VRAM Selection
SEL A	DISPCAPCNT	Capture source A selection
SEL B	DISPCAPCNT	Capture source B selection
SEL CAP	DISPCAPCNT	Capture mode selection
SEL CAP VRAM	DISPCAPCNT	Capture data write destination VRAM selection
SEL LCD	POWCNT	LCD output destination switch

After selecting the graphics display, VRAM display, or main memory display using SEL DISP, the image output becomes Image Output A.

Similarly, the image output of the 2D graphics engine B becomes Image Output B.

Image Outputs A and B each go through the Master Brightness Up/Down A and B, respectively, and become the Display Output A and Display Output B that are sent to the LCD.

When finally output to the LCD, these display outputs cannot be layered.

Choose one of the following:

- Send Display Output A to the Upper Screen LCD and send Display Output B to the Lower Screen LCD
- Send Display Output A to the Lower Screen LCD and send Display Output B to the Upper Screen LCD

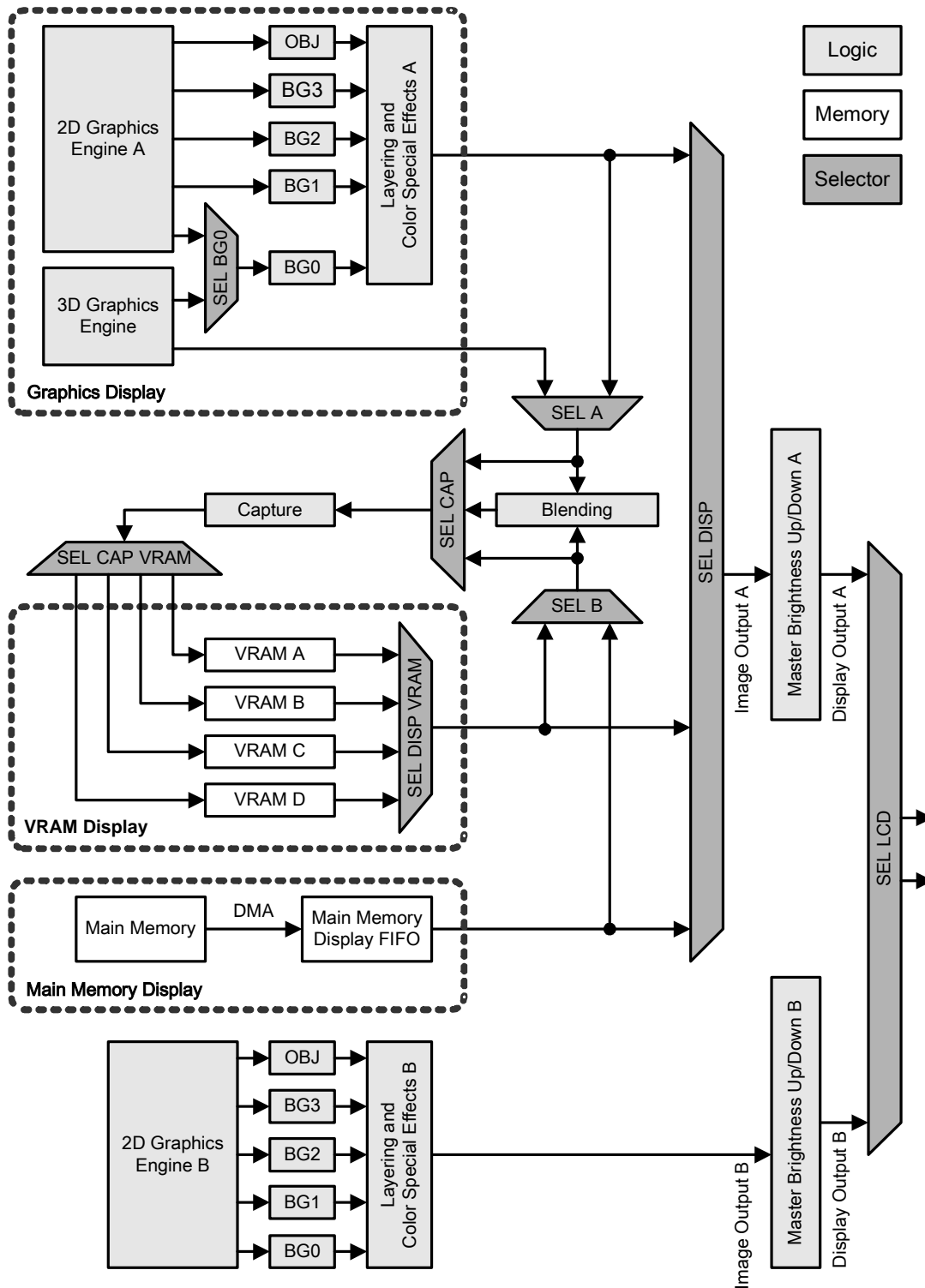
For games that require only one LCD, disable the LCD display you do not use.

For further details, see "[Power Management](#)" on page 285.

Image Output A allows you to blend and display 2D graphics and 3D graphics in the graphics display.

See "[Display](#)" on page 47 for information on the hardware block for 3D image creation.

Figure 4-1 : Display System Block Diagram



## 4.2 LCD

The specifications for the two LCD controllers included on the DS are shown below.

The top and bottom LCDs have the same specifications, but the directions differ on the Nintendo DS. So, the order of RGB pixel arrays differs.

### 4.2.1 LCD Controller Specifications

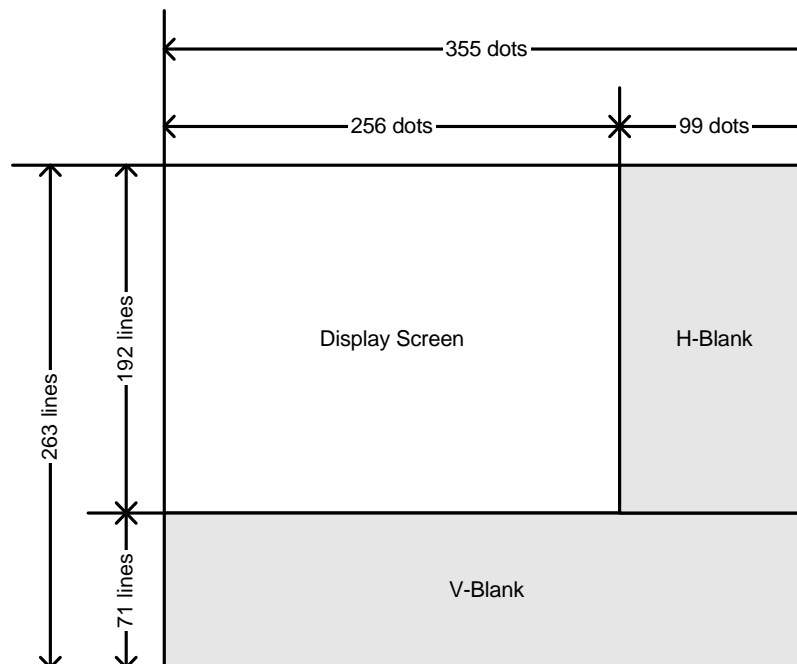
The LCD clock specifications of the LCD controller are shown in Table 4-2, the LCD scan timing is shown in Figure 4-2, and the specifications for the LCD scan timing are shown in "[Table 4-3 : LCD Scan Timing Specifications](#)" on page 50.

**Table 4-2 : LCD Clock Specifications**

LCD Clock	Frequency (time)
Image Processing Clock	33.513982 Mhz (29.838293 ns)
Dot Clock	5.585664 Mhz (179.029757 ns) (see note)

Note: The Dot Clock is 1/6 of the Image Processing Clock.

**Figure 4-2 : LCD Scan Timing**



**Table 4-3 : LCD Scan Timing Specifications**

Item		Spec	Period	Reference: AGB Value
Display Screen Size	Horizontal Dot Count	256 dots	45.8316 $\mu$ s	240 dots (57.221 $\mu$ s)
	Vertical Line Count	192 lines	12.2027 ms	160 lines (11.749 ms)
Total Dot Count	Horizontal Dot Count	355 dots	63.5556 $\mu$ s	308 dots (73.433 $\mu$ s)
	Vertical Line Count	263 lines	16.7151 ms	228 lines (16.743 ms)
Blanking	H-Blank Dot Count	99 dots	17.7239 $\mu$ s	68 dots (16.212 $\mu$ s)
	V-Blank Line Count	71 lines	4.5124 ms	68 lines (4.994 ms)
Scan Cycle	H-Cycle	15.7343 KHz	63.5556 $\mu$ s	13.618 KHz (73.443 $\mu$ s)
	V-Cycle	59.8261 Hz	16.7151 ms	59.727 Hz (16.743 ms)

The V-Blank cycle for the 3D rendering engine consists of 23 lines: 191–213. For details, see ["Rendering Engine" on page 226](#).



### 4.3 Display Status

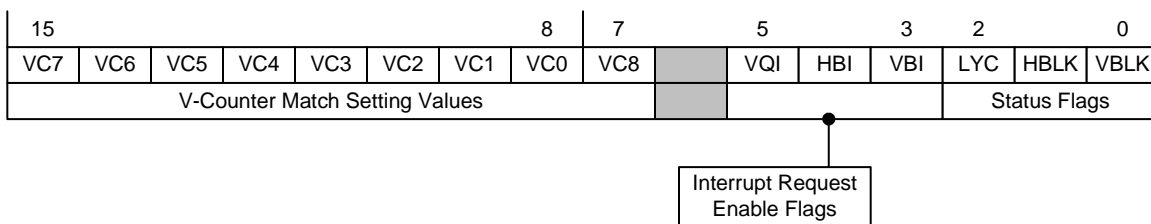
#### DISPSTAT: Display Status Register

Name: DISPSTAT

Address: 0x04000004

Attribute: R/W

Initial value: 0x0000



- [d15–d07] : V-Counter Match Setting Values

Note that VC8 is located in d07 (for compatibility with AGB). Values between 0 and 262 can be set. Proper operation is not guaranteed for a value of 263 or higher.

- [d05–d03] : Interrupt Request Enable Flags

- VQI[d05] : V-Counter match interrupt request enable flag

0	Disable
1	Enable

- HBI[d04] : H-Blank interrupt request enable flag

0	Disable
1	Enable

When enabled, H-Blank interrupts are permitted with the Interrupt Enable Register (IE). H-Blank interrupts can be made during the display interval, and also during any of the 263 vertical lines (Line 0 – 262) on the LCD, including V-Blank intervals.

- VBI[d03] : V-Blank interrupt request enable flag

0	Disable
1	Enable

- [d02–d00] : Status Flag

- LYC[d02] : V-Counter match detection flag

0	Outside a matching interval
1	During a matching interval

- HBLK[d01] : H-Blank detection flag

0	Outside H-Blank interval
1	During H-Blank interval

- VBLK[d00] : V-Blank detection flag

0	Outside V-Blank interval
1	During V-Blank interval

**Note:** V-Blank detection flag is set to 1 at the moment it reaches Line 192, and is set to 0 when it reaches Line 262. This is because the OBJ rendering circuitry accesses OBJ-VRAM and OAM starting at Line 262, which is one line before the actual display. In addition, the timing that ends access to OBJ-VRAM and OAM depends on whether the OBJ process is performed during H-Blank. On the other hand, BG-VRAM, BG Palette RAM, and OBJ Palette RAM begin access at Line 0, and end at Line 191. This is summarized in Table 4-4.

**Table 4-4 : Period when Graphics Engines Access Memory**

Memory Accessed by Graphics Engines		V-Counter Value
Access period when OBJ rendering circuitry accesses OBJ-VRAM and OAM	Perform OBJ process during H-Blank	0-191, 262
	Does not perform OBJ process during H-Blank	0-190, 262
Access period when rendering circuitry accesses BG-VRAM, BG Palette RAM, and OBJ Palette RAM		0-191
(Reference) Period when the V-Blank deflection flag becomes 1		192-261

**VCOUNT: V-Counter Register**

Name: VCOUNT

Address: 0x04000006

Attribute: R/W

Initial value: 0x0000

15							8	7								0	
							V8	V7	V6	V5	V4	V3	V2	V1	V0		
							V-Counter Values										

- [d08–d00] : V-Counter Values

Unlike the bit arrangement of the V-Counter Match Setting Values in the DISPSTAT register, these bits are in a normal arrangement.

**1. When reading values**

Can read which of the LCD's total 263 lines is currently displayed. The readout value is between 0 and 262.

If the readout value is between 0 and 191, images are being drawn. If the value is between 192 and 262, it is a V-Blank period.

To learn about the LCD's display timing, see "[LCD](#)" on page 49.

**2. When writing values**

Written values are reflected when the hardware's V-Counter is updated.

By using this register, you can synchronize all NITRO V cycles by adjusting the V-count value when communicating among multiple DS devices.

Confirm that the current value of the V-Counter is between 202 and 212, and write values only between 202 and 212. Proper operation of the 3D engine is not guaranteed when writing values outside this range.

**Note:** When there is a conflict between the access to the display circuit VRAM and the access to VRAM from the CPU, the display circuit VRAM takes precedence.

Because the dot clock of the LCD controller is 1/6 of a cycle of the image processing clock and the system clock, the timing for the LCD controller to access the VRAM is once every six cycles.

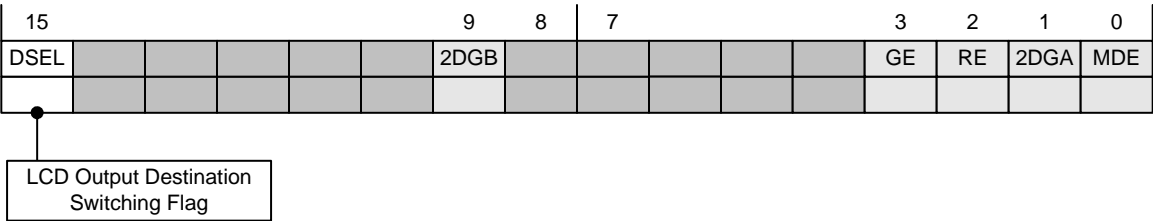
With this timing, when simultaneously accessing from the CPU, the CPU access must wait for one cycle.

4.4 Display Control

4.4.1 Top LCD/Bottom LCD Output Switching

POWCNT: Power Control Register

Name: POWCNT      Address: 0x04000304      Attributes: R/W      Initial Value: 0x0000



- DSEL[d15] : LCD Output Switching Flag

0	Send Display Output A to the Lower Screen LCD Send Display Output B to the Upper Screen LCD
1	Send Display Output A to the Upper Screen LCD Send Display Output B to the Lower Screen LCD

You can switch the LCD output destination with no delay by configuring the Power Control Register.

## 4.4.2 Display Control of 2D Graphics Engine A

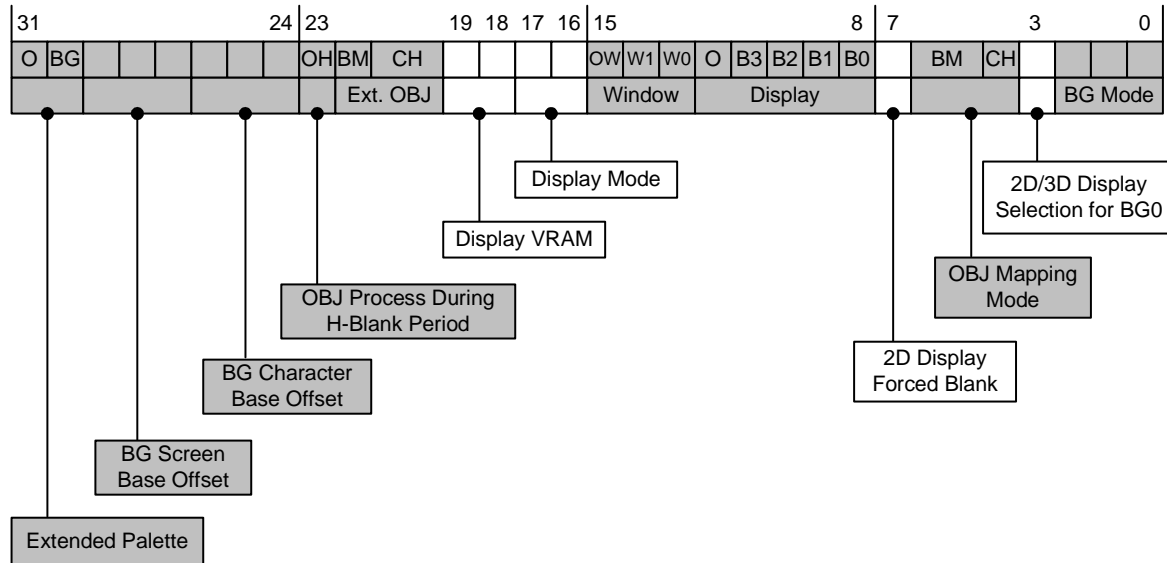
### DISPCNT: Display Control Register

Name: DISPCNT

Address: 0x04000000

Attribute: R/W

Initial value: 0x00000000



- [d19–d18] : Display VRAM

Selects the VRAM block to display when in VRAM Display Mode (see “d17–d16 Display mode”).

00	VRAM-A
01	VRAM-B
10	VRAM-C
11	VRAM-D

- [d17–d16] : Display mode

When the Display mode is OFF, the 2D/3D graphics, VRAM display, and main memory display are not selected and appear white.

Graphics display mode displays both 2D and 3D graphics.

VRAM display mode displays the bitmap data stored in VRAM.

Main memory display mode displays the bitmap data stored in main memory (requires a DMA setting).

For details, see the appropriate sections.

0	Display OFF
1	Graphics Display
2	VRAM Display
3	Main Memory Display

- [d07] : 2D Display Forced Blank

The 2D graphics display is forcibly halted by the CPU. Because 2D display is halted, 3D graphics using BG0 are not displayed either.

During a forced blank, the 2D graphics circuitry does not access VRAM, and the LCD screen is white.

However, even during a forced blank, the internal HV synchronization counter continues to run.

If the forced-blank setting is changed from ON to OFF during a display period of the internal HV synchronization counter, the effect takes place immediately; if it is changed from OFF to ON, the switch takes place at the start after three lines.

- [d03] : 2D/3D Display Selection for BG0

This bit determines whether to use one of the BG screens (BG0) for 2D graphics or for 3D graphics.

When 3D graphics are selected, the 2D graphics features for BG0 are limited and the specifications for color special effects change. See ["6.4 2D Graphics Features you can Apply to the 3D Screen after Rendering" on page 268](#).

0	Display 2D graphics
1	Display 3D graphics

- Other bits

The bits in the DISPCNT register not covered above are explained in the following sections:

Bits related to the display control of 2D graphics features are explained in ["5 2D Graphics" on page 73](#).

Bits related to BG are explained in ["5.2 BG" on page 77](#).

Bits related to OBJ are explained in ["5.3 OBJ" on page 109](#).



#### 4.4.4 Display Modes

As indicated in Table 4-5, on the Display Output A side (the 2D Graphics Engine A), there are modes that display the bitmap data in the VRAM and main memory in addition to the mode that displays the images generated by the graphics circuit.

**Table 4-5 : Overview of the Display Modes (2D Graphics Engine A)**

Display Mode Number	Display Mode	Display Size	Frame Rate	Features			
				3D Display	Character BG Display	Bitmap BG Display	OBJ Display
0	Display OFF	-	-	-	-	-	-
1	Graphics Display	256x192	60 fps	X	X	X	X
2	VRAM Display	256x192	60 fps			X	
3	Main Memory Display	256x192	60 fps			X	

As indicated in Table 4-6, on the Display Output B side (the 2D graphics engine B), the only mode selection is graphics display ON or OFF.

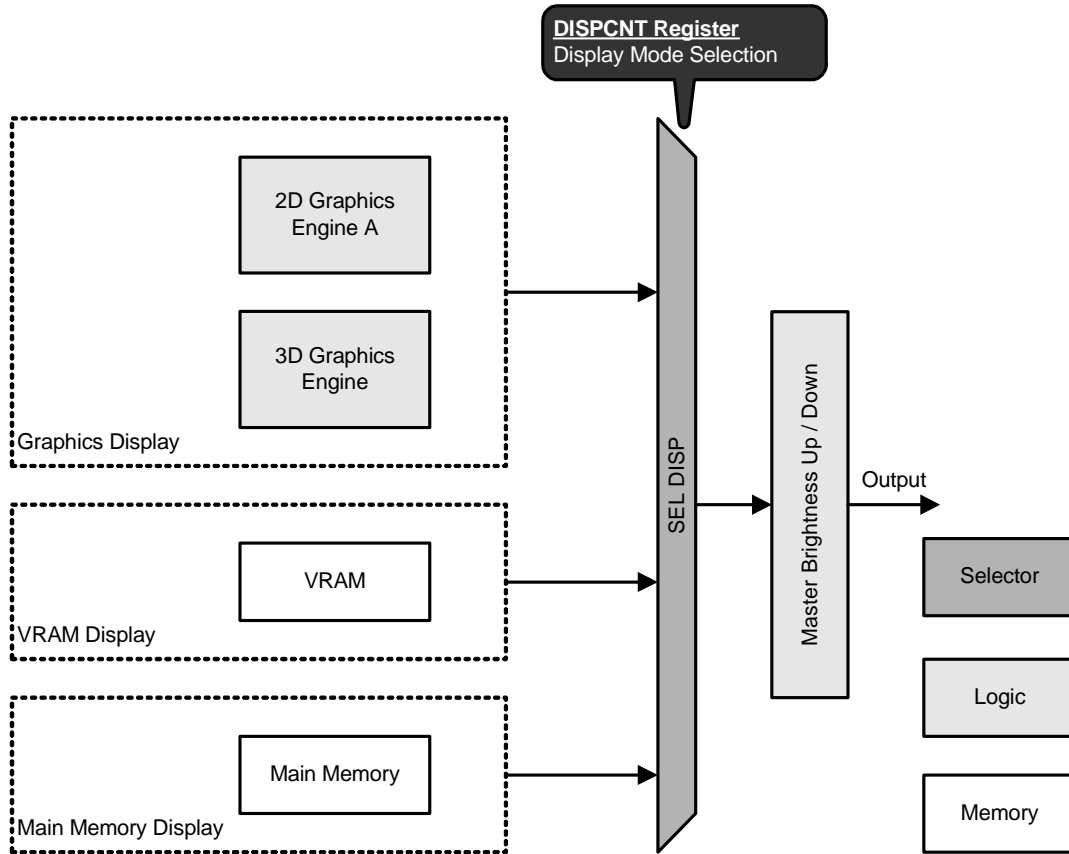
**Table 4-6 : Overview of the Display Modes (2D Graphics Engine B)**

Display Mode Number	Display Mode	Display Size	Frame Rate	Feature		
				Character BG Display	Bitmap BG Display	OBJ Display
0	Display OFF	-	-	-	-	-
1	Graphics Display	256x192	60 fps	X	X	X

"[Figure 4-3 : Display Mode Selection \(Display Output A Side Only\)](#)" on page 59 is a simplified version of the display mode for Display Output A in "[Figure 4-1 : Display System Block Diagram](#)" on page 48.



**Figure 4-3 : Display Mode Selection (Display Output A Side Only)**



#### 4.4.4.1 Graphics Display Mode

This mode displays images generated with the 2D and 3D graphics features.

See "[Figure 4-1 : Display System Block Diagram](#)" on page 48 for information about the entire display system.

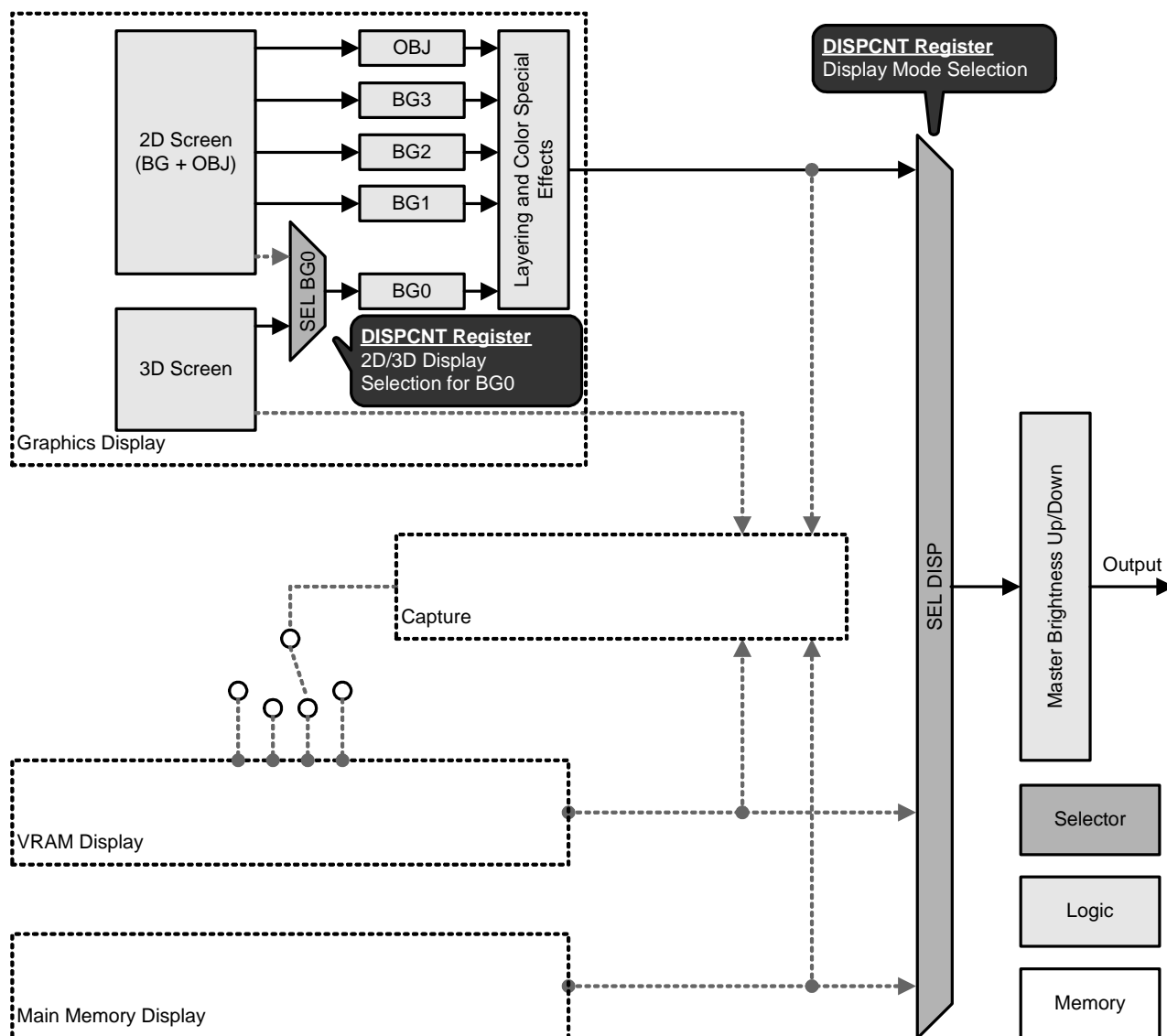
To read about the various graphic features, see "[5 2D Graphics](#)" on page 73 and "[6 3D Graphics](#)" on page 153.

- **Graphics display mode – Example 1**

The example in Figure 4-4 shows how the results of 3D rendering are layered with the 2D screen and displayed.

Although 3D display is handled as the BG0 screen, the 2D graphics features of BG0 are limited, and the color special effect specifications have changed. See "[6.4 2D Graphics Features you can Apply to the 3D Screen after Rendering](#)" on page 268.

**Figure 4-4 : Display Mode Selection (Display Output A Side Only)**



- **Graphics display mode – Example 2**

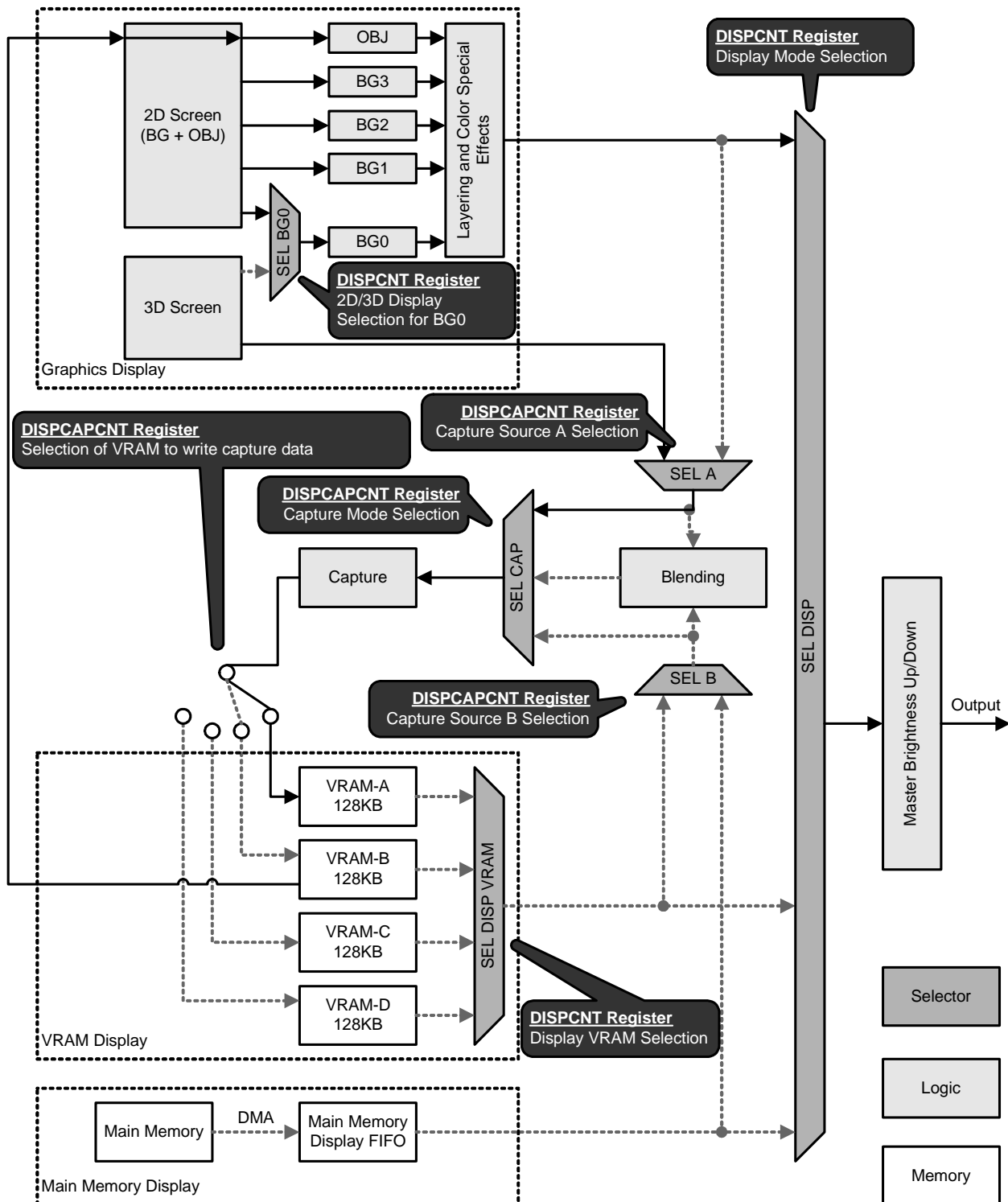
In the example shown in "[Figure 4-5 : Example of Displaying the Bitmap OBJ Results of 3D Rendering](#)" on page 62, the results of 3D rendering are pasted in a bitmap OBJ and displayed.

The rendering engine's clear alpha value is set to 0, and the 3D rendering result is captured. Then, in the next frame, the VRAM is assigned to a bitmap OBJ, according to the value of the RAM Bank Control register. This enables the 3D rendering result to be displayed as an OBJ.

At this moment in the sequence, alpha value segments that remain 0 in the 3D alpha-blending process are transparent. (See "[6.3.7 Alpha-Blending](#)" on page 258 for the capture feature and the rendering engine.)

In this example, double buffering occurs by alternately assigning VRAM-A and VRAM-B to LCDC and OBJ-VRAM.

Figure 4-5 : Example of Displaying the Bitmap OBJ Results of 3D Rendering



#### 4.4.4.2 VRAM Display Mode

When the DISPCNT register is set for VRAM display mode, one frame of bitmap data stored in a VRAM block is shown from the start of the next display. The DISPCNT register can specify which VRAM block to use.

VRAM is displayed using a different system than the 2D circuitry and the 3D circuitry, so when the mode is set to VRAM display mode, images can be created by the graphics circuitry and captured to VRAM at the same time that images are being displayed. (See "[Figure 4-1 : Display System Block Diagram](#)" on page 48).

You can specify the same VRAM block for display and for capturing images.

For details about capturing images, see "[4.5 Display Capture](#)" on page 67.

The pixel data format for VRAM display mode is shown below.

**VRAM Display Mode Data Format**

15	14		10	9	8	7		5	4		0
		BLUE			GREEN				RED		
		Pixel Color Data									

Figure 4-6 shows the VRAM address map of the LCD pixels.

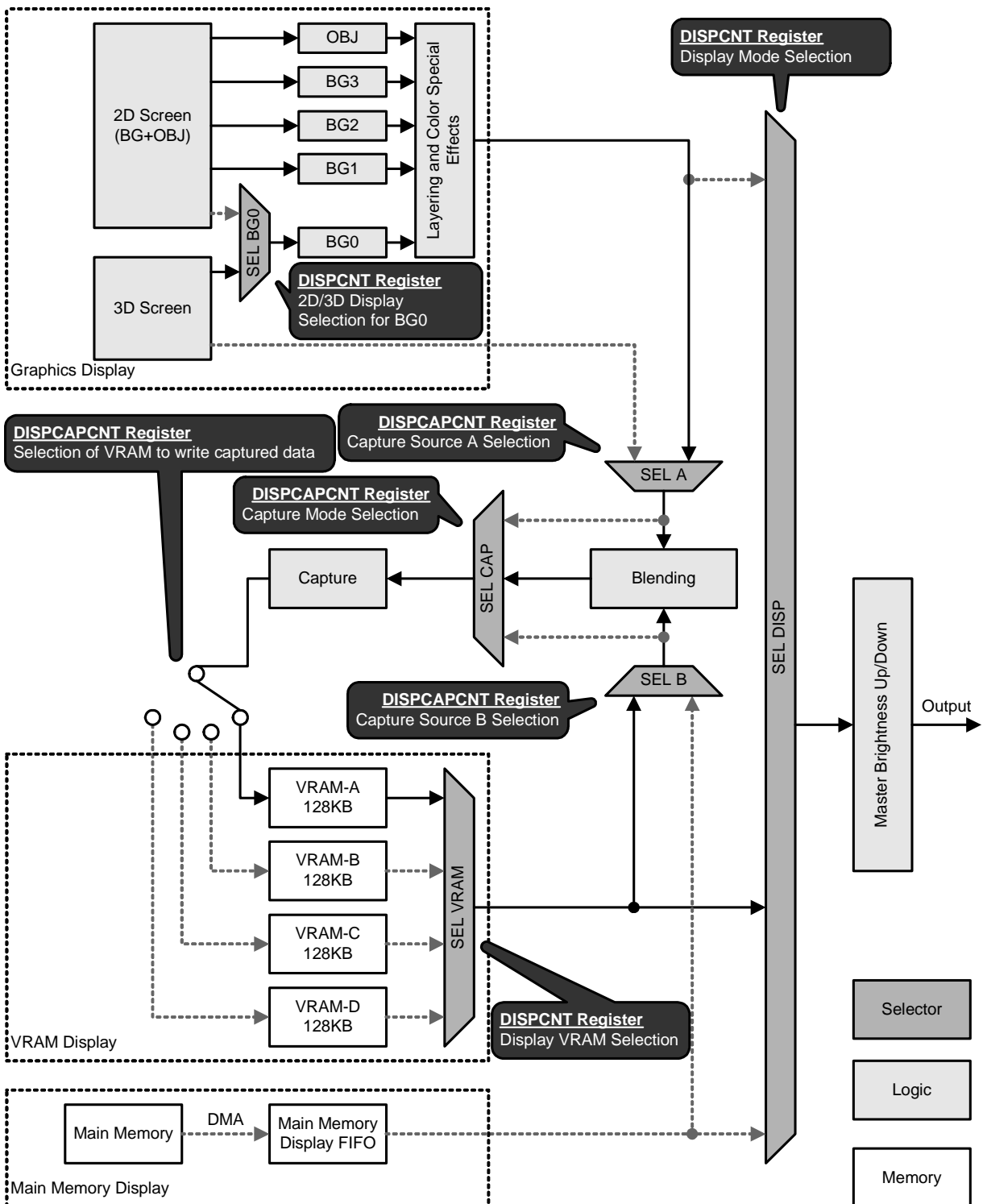
**Figure 4-6 : VRAM Address Map of the LCD Pixels**

	Dot 0	1	2	3		253	254	255
Line 0	0h	2h	4h	6h		1FAh	1FCh	1FEh
1	200h	202h	204h	206h		3FAh	3FCh	3FEh
2	400h	402h					5FCh	5FEh
3	600h	602h					7FCh	7FEh
4	800h							9FEh
187	17600h							177FEh
188	17800h	17802h					179FCh	179FEh
189	17A00h	17A02h					17BFCh	17BFEh
190	17C00h	17C02h	17C04h	17C06h		17DFAh	17DFCh	17DFEh
191	17E00h	17E02h	17E04h	17E06h		17FFAh	17FFCh	17FFEh

- VRAM display mode - Example

In "[Figure 4-7 : Example of the Motion Blur Effect that Uses the Display Capture](#)" on page 64, the image created by the graphic circuitry is put into VRAM using the capture feature, and then the image is displayed with the mode set to VRAM Display. When the image is captured, a motion blur effect is achieved by blending with the display-use VRAM.

Figure 4-7 : Example of the Motion Blur Effect that Uses the Display Capture



#### 4.4.4.3 Main Memory Display Mode

This mode enables the display of bitmap data held in main memory. When the DISPCNT register is set to main memory display mode, the data held in the main memory display FIFO is transferred to the LCD module at the beginning of the next display. A data request is sent to DMA for every data transfer.

There is a four-layer FIFO between the main memory display FIFO register and the LCD module, and the LCD module takes four words at a time. For this reason, you should write four layers of data at a time to the main memory display FIFO register.

To be more specific, after setting the DMA transfer bit width to 32 bits and the word count to 4, set the DMA startup mode to main memory display mode. For this mode, be sure to set the DMA source address to the main memory region.

Table 4-7 shows the DMA configuration when using the main memory display mode.

### Table 4-7 : DMA Configuration when Using the Main Memory Display Mode

Setting	Value
Source Address	Main memory
Transfer Bit Width	32 bits
Word Count	4

For details about the DMA configuration, see ["7 DMA" on page 273](#).

Data from main memory is displayed using a system other than the 2D and 3D circuitry, so when the mode is set to main memory display, images can be created by the graphics circuitry and captured to VRAM at the same time that images are being displayed. (See "[Figure 4-1 : Display System Block Diagram](#)" on page 48.)

## Main Memory Display FIFO Register

Name: DISP\_MMEM\_FIFO

Address: 0x04000068

Attribute: R/W

Initial value: 0x00000000

31	30				26	25	24	23		21	20			16	15	14			10	9	8	7		5	4			0
		BLUE				GREEN				RED						BLUE				GREEN				RED				
ODD														EVEN														

"[Figure 4-8 : LCD Pixel EVEN/ODD Map of the Main Memory Display FIFO Register](#)" on page 66 shows the LCD pixel EVEN/ODD map of the main memory display FIFO register.

**Figure 4-8 : LCD Pixel EVEN/ODD Map of the Main Memory Display FIFO Register**

	Dot 0	1	2	3		253	254	255
Line 0	EVEN	ODD	EVEN	ODD		ODD	EVEN	ODD
1	EVEN	ODD	EVEN	ODD		ODD	EVEN	ODD
2	EVEN	ODD					EVEN	ODD
3	EVEN	ODD					EVEN	ODD
4	EVEN							ODD
187	EVEN							ODD
188	EVEN	ODD					EVEN	ODD
189	EVEN	ODD					EVEN	ODD
190	EVEN	ODD	EVEN	ODD		ODD	EVEN	ODD
191	EVEN	ODD	EVEN	ODD		ODD	EVEN	ODD



## 4.5 Display Capture

This feature enables 2D and 3D graphics and image output from VRAM and main memory to be read into VRAM.

Only the image output on the Display Output A side (the 2D graphics engine) can be captured.

It also enables images from two sources to be blended, and then captured.

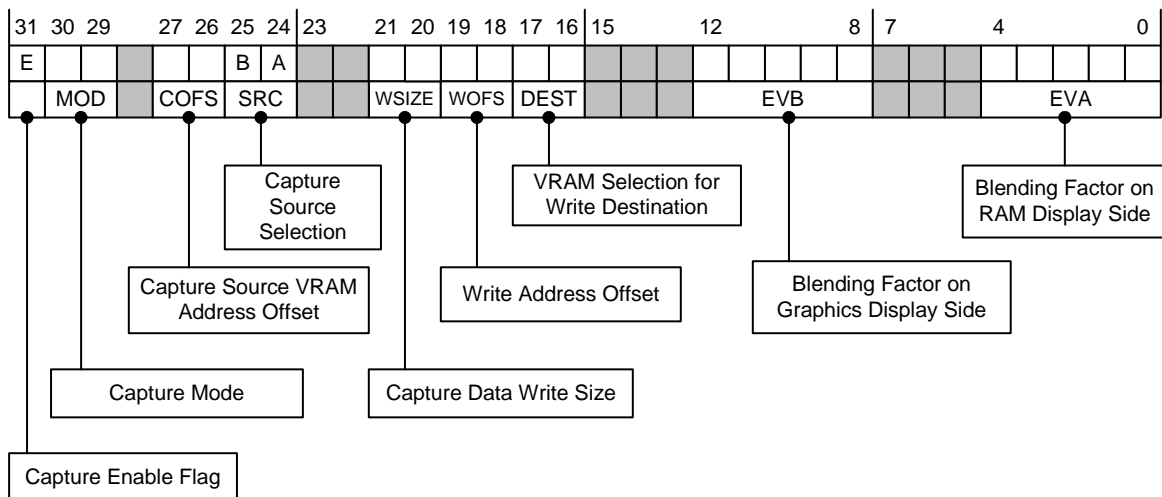
### DISPCAPCNT: Display Capture Control Register

Name: DISPCAPCNT

Address: 0x04000064

Attribute: R/W

Initial value: 0x00000000



- E[d31] : Display Capture Enable Flag

When the flag is set to 1, one screen of data is captured from the next 0 line, and then the flag is set to 0.

0	Disable
1	Enable

- MOD[d30–d29] : Capture Mode

00	Capture data from source A
01	Capture data from source B
10	Capture the result of blending data from sources A and B
11	

- COFS[d27–d26] : Read Address Offset for Capture Data Source VRAM

Invalid in VRAM display mode.

If the offset exceeds 0x20000 during reading, the reading continues after wrapping to address 0x00000.

00	0x00000
01	0x08000
10	0x10000
11	0x18000

- SRC[d25–d24] : Capture Data Source Selection

B

0	VRAM
1	Main Memory

A

0	Graphics display screen (after 3D/2D blending)
1	3D screen

- WSIZE[d21–d20] : Capture Size

Specifies the size when writing the capture data. With RAM captures, one line is always read as a 256-dot image, so you cannot blend and then capture (see Capture mode above) when the setting is 128x128 dots.

00	128x128 dots (0x08000 bytes)
01	256x64 dots (0x08000 bytes)
10	256x128 dots (0x10000 bytes)
11	256x192 dots (0x18000 bytes)

- WOFS[d19–d18] : Address Offset for Capture Data Write

This can specify the offset value for the address where data is written in the specified VRAM. If the offset exceeds 0x20000 during writing, the writing continues after wrapping to address 0x00000.

00	0x00000
01	0x08000
10	0x10000
11	0x18000

- DEST[d17–d16] : Capture Data Write Destination VRAM Selection

The write destination VRAM must be allocated to the LCDC.

00	VRAM-A
01	VRAM-B
10	VRAM-C
11	VRAM-D

- EVB[d12–d08], EVA[d04–d00] : Blending Factors

Sets the blending factors for capture sources A and B. See below for the calculation method.

In VRAM display mode, you can set the same VRAM block for display VRAM and for writing the captured image data.

### Capture Data Format

15	14	10	9	8	7	5	4	0
A	BLUE			GREEN			RED	
$\alpha$	Pixel Color Data							

Although 3D graphics are output in R:G:B=6:6:6 color, because capture occurs in R:G:B=5:5:5 color (employing the upper 5 bits), the image gradient becomes a little coarse.

Figure 4-9 shows the LCD pixel map of the captured data when the capture size is 256 x 192 dots.

**Figure 4-9 : LCD Pixel Map of the Capture Data (When the Capture Size is 256 x 192 Dots)**

	Dot 0	1	2	3		253	254	255
Line 0	0h	2h	4h	6h		1FAh	1FCh	1FEh
1	200h	202h	204h	206h		3FAh	3FCh	3FEh
2	400h	402h					5FCh	5FEh
3	600h	602h					7FCh	7FEh
4	800h							9FEh
187	17600h							177FEh
188	17800h	17802h					179FCh	179FEh
189	17A00h	17A02h					17BFCh	17BFEh
190	17C00h	17C02h	17C04h	17C06h		17DFAh	17DFCh	17DFEh
191	17E00h	17E02h	17E04h	17E06h		17FFAh	17FFCh	17FFEh

- **How to calculate data to write**

1. For data captured from source A:

$$CAP = Ca$$

Capture source A's alpha value is used for the alpha value.

2. For data captured from source B:

$$CAP = Cb$$

Capture source B's alpha value is used for the alpha value

3. For capturing data blended from sources A and B:

$$CAP = \frac{(Ca \times Aa \times EVA) + (Cb \times Ab \times EVB)}{16}$$

The alpha value is 1 when EVA is non-zero and capture source A's alpha value is 1, or when EVB is non-zero and capture source B's alpha value is 1. In all other circumstances, the alpha value is 0.

CAP: The color to write (calculation results are rounded to the nearest integer)

Ca: A's capture source data color, EVA: Blending factor for A

Cb: B's capture source data color, EVB: Blending factor for B

Aa: A's alpha value: A's capture source alpha value.

Determined as shown below.

Capture Source A Selection	3D Screen Alpha Value	Aa
0	-	1
1	0	0
	1 - 31	1

Ab: alpha value of B: alpha value of B's capture source

**Note:** When a conflict occurs between access to the display circuit VRAM and access to VRAM from the CPU, the display circuit VRAM access takes precedence.

Because the dot clock of the LCD controller is 1/6 of a cycle of the image processing clock and the system clock, the timing for the LCD controller to access the VRAM is once every six cycles.

If the VRAM of the capture is being displayed while display capturing, the frequency at which the display circuit accesses the VRAM is doubled, and the VRAM is accessed with a timing of once every three cycles.

With this timing, when simultaneously accessing from the CPU, the CPU access must wait one cycle.

## 4.6 Master Brightness

The brightness up/down process for Image Output A is handled by the configuration of the MASTER\_BRIGHT register, while the brightness up/down process for Image Output B is handled by the configuration of the DB\_MASTER\_BRIGHT register.

### MASTER\_BRIGHT: Master Brightness Up/Down Register

Name: MASTER\_BRIGHT Address: 0x0400006C Attribute: R/W Initial value: 0x0000

15	14									4					0
E_MOD															E_VALUE
Mode															Factor

### DB\_MASTER\_BRIGHT: Master Brightness Up/Down B Register

Name: DB\_MASTER\_BRIGHT Address: 0x0400106C Attribute: R/W Initial value: 0x0000

15	14									4					0
E_MOD															E_VALUE
Mode															Factor

The MASTER\_BRIGHT and DB\_MASTER\_BRIGHT registers share identical configuration details.

- E\_MOD [d15–d14] : Mode

Sets the mode for processing brightness up/down.

Setting	Process
00	No change in brightness
01	Increase brightness
10	Decrease brightness
11	Setting prohibited

- E\_VALUE [d04–d00] : Factor

Sets the factors as calculated below.

#### 1. Brightness up computation

$$R_{out} = R_{in} + (63 - R_{in}) \times (E\_VALUE/16)$$

$$G_{out} = G_{in} + (63 - G_{in}) \times (E\_VALUE/16)$$

$$B_{out} = B_{in} + (63 - B_{in}) \times (E\_VALUE/16)$$

#### 2. Brightness down computation

$$R_{out} = R_{in} - R_{in} \times (E\_VALUE/16)$$

$$G_{out} = G_{in} - G_{in} \times (E\_VALUE/16)$$

$$B_{out} = B_{in} - B_{in} \times (E\_VALUE/16)$$

The result of the Brightness Up and Down computation ( $R_{out}$ ,  $G_{out}$ , and  $B_{out}$ ) is rounded to the nearest integer.



## 5 2D Graphics

NITRO has two 2D graphics engines: 2D Graphics Engine A and 2D Graphics Engine B. 2D Graphics Engine A can use 3D Graphics BG and large-screen 256-color Bitmap BG, but 2D Graphics Engine B cannot. In the following sections, 2D Graphics Engine A is sometimes referred to as 2D\_A and 2D Graphics Engine B as 2D\_B. Where register names differ for 2D Graphics Engine A and 2D Graphics Engine B, the register name for 2D Graphics Engine B is given inside square brackets [].

### 5.1 Controlling the 2D Display

The display for each 2D graphics feature can be controlled and turned on or off independently. Control register settings differ for 2D Graphics Engine A and 2D Graphics Engine B.

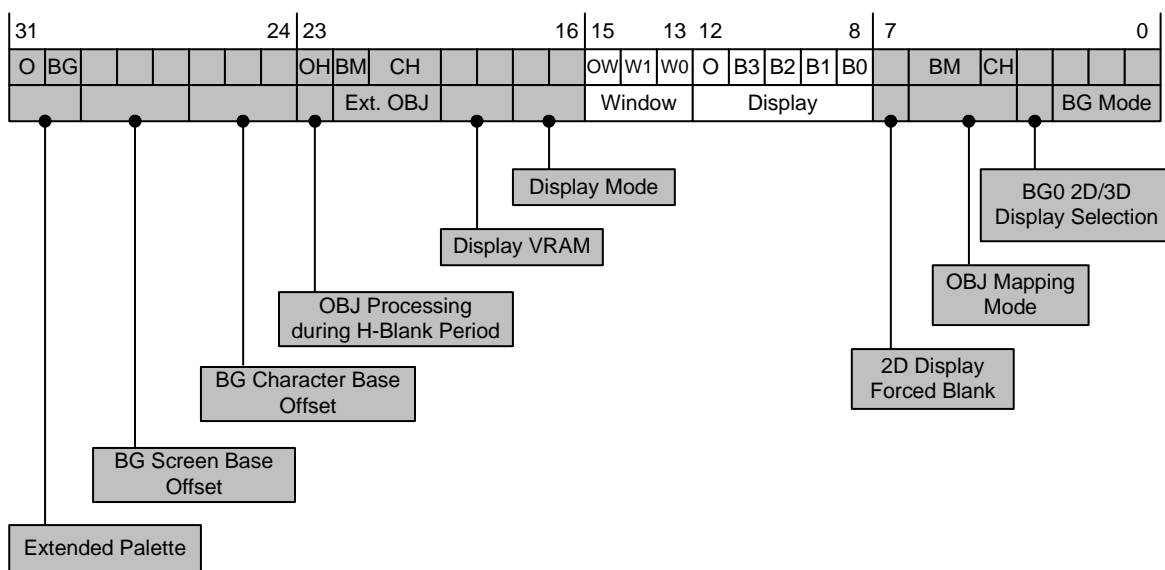
#### DISPCNT: Display Control Register (2D Graphics Engine A)

Name: DISPCNT

Address: 0x04000000

Attribute: R/W

Initial Value: 0x00000000



- [d15–d13] : Window Display Enable Flag

See "[5.6 Windows](#)" on page 142 to read about the window features.

- OW [d15] : OBJ Window Display Enable Flag

0	Disable display
1	Enable display

To display the OBJ Window requires enabling both the OBJ Window Display Enable Flag and the OBJ Display Enable Flag.

- W1 [d14] : Window 1 Display Enable Flag

0	Disable display
1	Enable display

- W0 [d13] : Window 0 Display Enable Flag

0	Disable display
1	Enable display

- [d12–d08] : Display Selection Flag

- O [d12] : OBJ Display Enable Flag

0	Disable display
1	Enable display

- B3 [d11] : BG3 Display Enable Flag

0	Disable display
1	Enable display

- B2 [d10] : BG2 Display Enable Flag

0	Disable display
1	Enable display

- B1 [d09] : BG1 Display Enable Flag

0	Disable display
1	Enable display

- B0 [d08] : BG0 Display Enable Flag

0	Disable display
1	Enable display



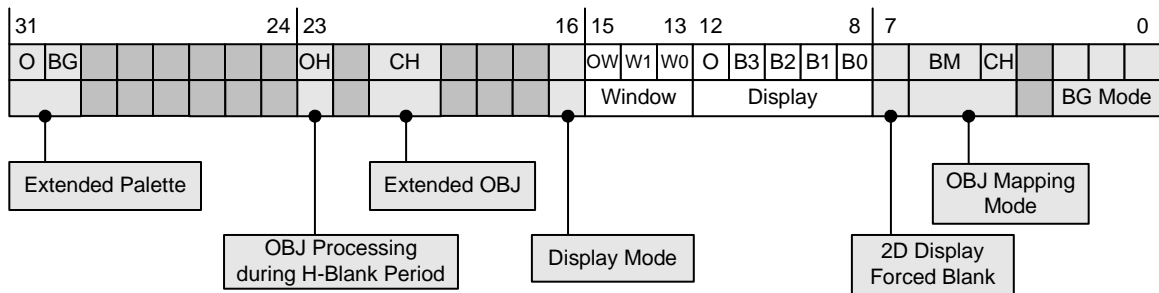
**DB\_DISPCNT: Display Control Register 1 (2D Graphics Engine B)**

Name: DB\_DISPCNT

Address: 0x04001000

Attribute: R/W

Initial Value: 0x00000000



- [d15–d13] : Window Display Enable Flag

See "[5.6 Windows](#)" on page 142 to read about the window features.

- OW [d15] : OBJ Window Display Enable Flag

0	Disable display
1	Enable display

To display the OBJ Window requires enabling both the OBJ Window Display Enable Flag and the OBJ Display Enable Flag.

- W1 [d14] : Window 1 Display Enable Flag

0	Disable display
1	Enable display

- W0 [d13] : Window 0 Display Enable Flag

0	Disable display
1	Enable display

- [d12–d08] : Display Selection Flag

- O [d12] : OBJ Display Enable Flag

0	Disable display
1	Enable display

- B3 [d11] : BG3 Display Enable Flag

0	Disable display
1	Enable display

- B2 [d10] : BG2 Display Enable Flag

0	Disable display
1	Enable display

- B1 [d09] : BG1 Display Enable Flag

<b>0</b>	Disable display
<b>1</b>	Enable display

- B0 [d08] : BG0 Display Enable Flag

<b>0</b>	Disable display
<b>1</b>	Enable display

## 5.2 BG

### 5.2.1 BG Mode

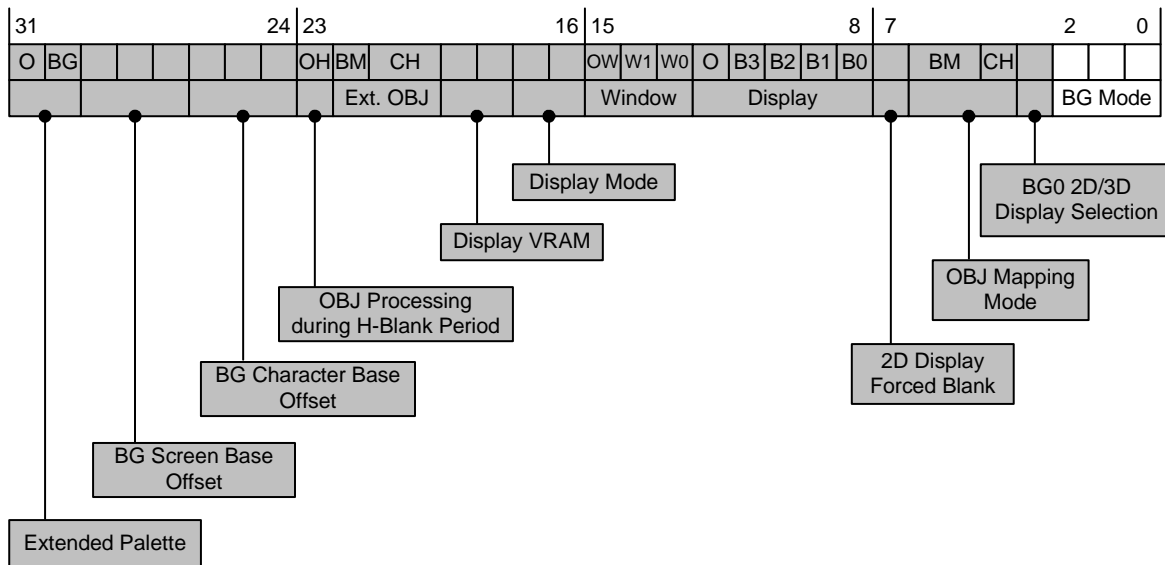
The BG modes that can be set for 2D Graphics Engine A and 2D Graphics Engine B are different.

#### 5.2.1.1 2D Graphics Engine A

With 2D Graphics Engine A, BG0 can be displayed as either 2D or 3D. In addition, Large-Screen 256-Color Bitmap BG can be selected as the BG type for BG2.

#### DISPCNT: Display Control Register (2D Graphics Engine A)

Name: DISPCNT      Address: 0x04000000      Attribute: R/W      Initial Value: 0x00000000



- [d02–d00] : BG Mode

These bits set the BG mode number. The BG mode selects the BG types that can be used. See Table 5-1 for a list of BG modes for 2D Graphics Engine A.

The DISPCNT register can be used to select either Text BG or 3D BG as the BG type for BG0. See chapter ["6 3D Graphics" on page 153](#) for details on 3D BG display.

**Table 5-1: List of BG Modes (2D Graphics Engine A)**

BG Mode Number	BG0	BG1	BG2	BG3
0	Text BG/3D BG	Text BG	Text BG	Text BG
1	Text BG/3D BG	Text BG	Text BG	Affine BG
2	Text BG/3D BG	Text BG	Affine BG	Affine BG
3	Text BG/3D BG	Text BG	Text BG	Affine Extended BG
4	Text BG/3D BG	Text BG	Affine BG	Affine Extended BG
5	Text BG/3D BG	Text BG	Affine Extended BG	Affine Extended BG
6	3D BG	—	Large-Screen 256-Color Bitmap BG	—
7	Prohibited Setting			

### 5.2.1.2 2D Graphics Engine B

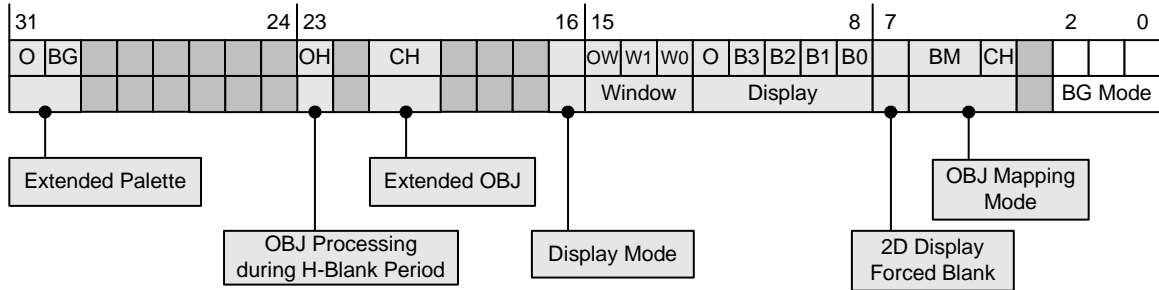
#### DB\_DISPCNT: Display Control Register 1 (2D Graphics Engine B)

Name: DB\_DISPCNT

Address: 0x04001000

Attribute: R/W

Initial Value: 0x00000000



- [d02–d00] : BG Mode

These bits set the BG mode number. The BG mode selects the BG types that can be used. See Table 5-2 for a list of BG modes for 2D Graphics Engine B.

**Note:** Unlike 2D Graphics Engine A, Large-screen 256-Color Bitmap BG cannot be set as the BG type for BG2. Furthermore, 3D BG display cannot be set as the BG type for BG0.

**Table 5-2 : List of BG Modes (2D Graphics Engine B)**

BG Mode Number	BG0	BG1	BG2	BG3
0	Text BG	Text BG	Text BG	Text BG
1	Text BG	Text BG	Text BG	Affine BG
2	Text BG	Text BG	Affine BG	Affine BG
3	Text BG	Text BG	Text BG	Affine Extended BG
4	Text BG	Text BG	Affine BG	Affine Extended BG
5	Text BG	Text BG	Affine Extended BG	Affine Extended BG
6	Prohibited Setting			
7	Prohibited Setting			

### 5.2.1.3 Basic Features for Each Type of BG

Each BG type has its own special features described in Table 5-3.

**Table 5-3 : Basic Features of BG Types**

BG Type	Features
<b>3D BG</b>	This type can display images generated by the 3D graphics engine. It can be displayed with other BG screens according to alpha-blending and priority settings. 2D Graphics Engine B cannot use this type.
<b>Text BG</b>	This type is a character format BG. Text BG is the only BG type that can handle characters defined in 16 colors and control VRAM consumption, but it cannot accommodate affine transformations.
<b>Affine BG</b>	This type is the character format BG that can accommodate affine transformations. It cannot perform character-unit processes (such as HV Flips).
<b>Affine Extended BG</b>	Three types are available: <ul style="list-style-type: none"> <li>• Character BG that can use 256 colors x 16 palettes</li> <li>• 256-Color Bitmap BG</li> <li>• Direct Color Bitmap BG that can specify color directly</li> </ul>
<b>Large-Screen 256-Color Bitmap BG</b>	This type is the Large-Screen Bitmap BG. Because one screen makes full use of the maximum capacity of BG-VRAM (512 KB), it cannot be used together with other BGs. However, it can be used together with a 3D screen. 2D Graphics Engine B cannot use this type.

### 5.2.1.4 Specifications for Different BG Types

The specifications for the different types of BG are shown in Table 5-4.

**Table 5-4 : Specifications for BG Types**

Category	Character BG				Bitmap BG		
BG Type Specs	3DBG	Text BG	Affine BG	Affine Extended BG			Large-Screen 256-Color Bitmap BG
				256-Color x 16 Palettes	256-Color	Direct Color	
Screen Size	256x192	256x256 512x256 256x512 512x512	128x128 256x256 512x512 1,024x1,024	128x128 256x256 512x512 1,024x1,024	128x128 256x256 512x256 512x512	128x128 256x256 512x256 512x512	512x1,024 1,024x512
Specifiable Character Count	—	1,024	256	1,024	—	—	—
Number of Colors/Palettes	262,144	16/16 256/1 256/16	256/1	256/16	256/1	32,768	256/1
Affine			X	X	X	X	X
HV Flip		X		X			
H Scroll	X	X	X	X	X	X	X
V Scroll		X	X	X	X	X	X
Mosaic		X	X	X	X	X	X
Fade-in/Fade-out	X	X	X	X	X	X	X
Alpha Blending	X	X	X	X	X	X	X
Priority	X	X	X	X	X	X	X

**Note:** 2D Graphics Engine B cannot set 3D BG and Large-Screen 256-Color Bitmap BG as BG types.

**Note:** Because the allocation of BG-VRAM to 2D Graphics Engine B is limited, the following settings cannot be used:

- 256-Color Bitmap: 512x512
- Direct Color Bitmap: 512x256 and 512x512

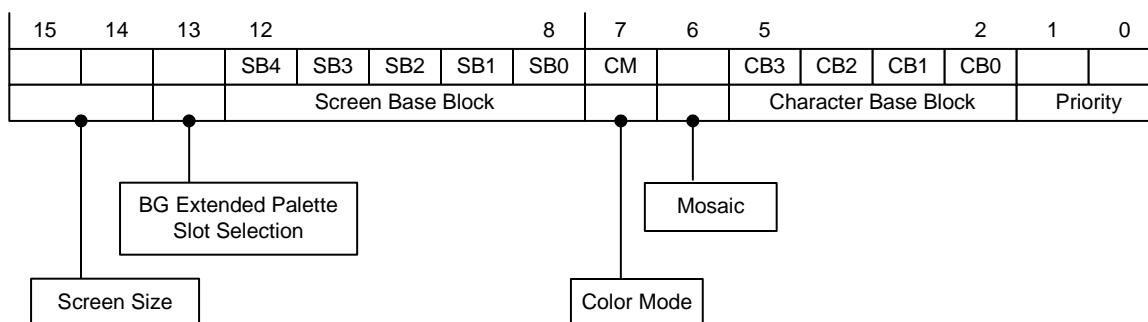
## 5.2.2 BG Control

There are four BG control registers that correspond to the number of BG screens. With the 2D Graphics Engine A, the BG screens are controlled with the BG0CNT, BG1CNT, BG2CNT, and BG3CNT registers. With the 2D Graphics Engine B, the BG screens are controlled with the DB\_BG0CNT, DB\_BG1CNT, DB\_BG2CNT, and DB\_BG3CNT registers.

**Note:** 2D Graphics Engine A and 2D Graphics Engine B use different register names as well as different methods to calculate base address values for BG screen data and BG character data.

### BGx(x=0, 1) Control Register

Name	Address	Attribute	Initial Value
(2D_A) BGxCNT(x=0, 1)	0x04000008, 0x0400000A	R/W	0x0000
(2D_B) DB_BGxCNT(x=0, 1)	0x04001008, 0x0400100A	R/W	0x0000



- [d15–d14] : Screen Size

Screen Size Settings	Text BG	
	Screen Size	Screen Data Size
00	256x256	2 KB
01	512x256	4 KB
10	256x512	4 KB
11	512x512	8 KB

- [d13] : BG Extended Palette Slot Selection

This bit specifies the Extended Palette Slot Number used when BG extended palettes are enabled. The settings differ for BG0 and BG1. Extended palettes are enabled/disabled with the DISPCNT [DB\_DISPCNT] register. See "[2.2.1 VRAM](#)" on page 17 for more information on the palette slot memory map.

1. BG0CNT [DB\_BG0CNT]

0	Slot 0
1	Slot 2

2. BG1CNT [DB\_BG1CNT]

0	Slot 1
1	Slot 3

- SB4-SB0 [d12–d08] : Screen Base Block

These bits specify (in 2-KB units) the starting block in VRAM where screen data is stored. When screen data is actually referenced, the starting address is calculated as follows.

### 2D Graphics Engine A

The starting address is the sum of the DISPCNT register's screen base offset value with a 64-KB offset and the screen base block with a 2-KB offset.

$$(\text{screen base offset} \times 0x10000) + (\text{screen base block} \times 0x800)$$

### 2D Graphics Engine B

The starting address is the screen base block with a 2-KB offset.

$$(\text{screen base block} \times 0x800)$$

- CM [d07] : Color Mode

This bit specifies whether the screen data references BG character data in 16-color or 256-color format.

0	16-color mode
1	256-color mode

- [d06] : Mosaic

This bit controls whether the mosaic process for BG is on or off. Set the mosaic size with the MOSAIC [DB\_MOSAIC] register.

- CB3-CB0 [d05–d02] : Character Base Block

These bits specify (in 16-KB units) the starting block in VRAM for storing character data. When character data is actually referenced, the starting address is calculated as follows.

### 2D Graphics Engine A

The starting address is the sum of the DISPCNT register's character base offset value with a 64-KB offset and the character base block with a 16-KB offset.

$$(\text{character base offset} \times 0x10000) + (\text{character base block} \times 0x4000)$$

### 2D Graphics Engine B

The starting address is the character base block with a 16-KB offset.

$$(\text{character base block} \times 0x4000)$$

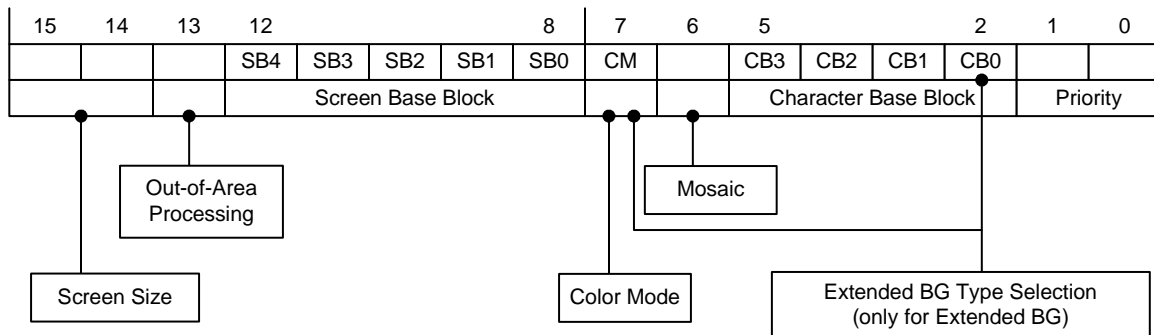
- [d01–d00] : Priority

The default order of priority among the BG screens is BG0 > BG1 > BG2 > BG3 (when priority settings are the same). However, this order can be changed. Priorities of 0 (highest) to 3 (lowest) can be set. Be careful of the pixel specifications to which color special effects are applied when changing BG priorities.



**BGx(x=2, 3) Control Register**

Name	Address	Attribute	Initial Value
(2D_A) BGxCNT(x=2, 3)	0x0400000C, 0x0400000E	R/W	0x0000
(2D_B) DB_BGxCNT(x=2, 3)	0x0400100C, 0x0400100E	R/W	0x0000



The bit definitions for Color Mode, Mosaic, Character Base Block, and Priority are the same as for the BGx (x=0, 1) Control Registers described above. BG2 uses Extended Palette Slot 2 and BG3 uses Extended Palette Slot 3 when extended palettes are enabled. The extended palette numbers used by BG2 and BG3 cannot be changed. The extended palettes can be enabled/disabled with the DISPCNT [DB\_DISPCNT] register.

- [d15–d14] : Screen Size

The screen sizes that can be configured depend on the BG type and are described in Table 5-5 and "[Table 5-6 : Screen Sizes \(2D Graphics Engine B\)](#)" on page 84.

**Note:** 2D Graphics Engine A and 2D Graphics Engine B accommodate different combinations of screen sizes and BG types that can be configured. 2D Graphics Engine B cannot set Large-Screen 256-Color Bitmap BG as the BG type. In addition, because a maximum of 128 KB of BG-VRAM can be allocated to 2D Graphics Engine B, screen sizes exceeding 128 KB are prohibited.

**Table 5-5 : Screen Sizes (2D Graphics Engine A)**

Screen Size Settings	Text BG	Affine BG	Affine Extended BG			Large-Screen 256-Color Bitmap BG
			256-Color x 16-Palette Character BG	256-Color Bitmap BG	Direct-Color Bitmap BG	
<b>00</b>	256x256 (2 KB)	128x128 (256 bytes)	128x128 (512 bytes)	128x128 <16 KB >	128x128 <32 KB >	512x1024 <512 KB >
<b>01</b>	512x256 (4 KB)	256x256 (1 KB)	256x256 (2 KB)	256x256 <64 KB >	256x256 <128 KB >	1024x512 <512 KB >
<b>10</b>	256x512 (4 KB)	512x512 (4 KB)	512x512 (8 KB)	512x256 <128 KB >	512x256 <256 KB >	—
<b>11</b>	512x512 (8 KB)	1024x1024 (16 KB)	1024x1024 (32 KB)	512x512 <256 KB >	512x512 <512 KB >	—

**Note:** The screen size is enclosed in parentheses () and the bitmap data size in angle brackets <>.

**Table 5-6 : Screen Sizes (2D Graphics Engine B)**

Screen Size Settings	Text BG	Affine BG	Affine Extended BG		
			256-Color x 16-Palette Character BG	256-Color Bitmap BG	Direct-Color Bitmap BG
<b>00</b>	256x256 (2 KB)	128x128 (256 bytes)	128x128 (512 bytes)	128x128 <16 KB >	128x128 <32 KB >
<b>01</b>	512x256 (4 KB)	256x256 (1 KB)	256x256 (2 KB)	256x256 <64 KB >	256x256 <128 KB >
<b>10</b>	256x512 (4 KB)	512x512 (4 KB)	512x512 (8 KB)	512x256 <128 KB >	Prohibited Setting
<b>11</b>	512x512 (8 KB)	1024x1024 (16 KB)	1024x1024 (32 KB)	Prohibited Setting	Prohibited Setting

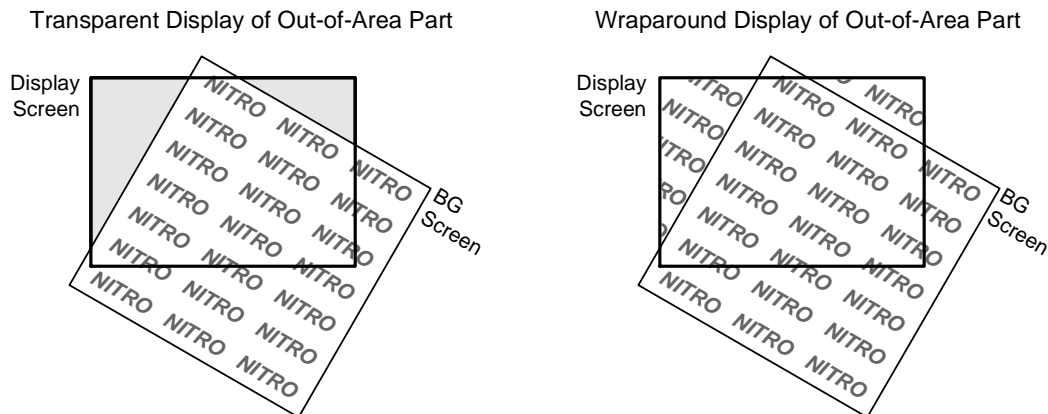
**Note:** The screen size is enclosed in parentheses () and the bitmap data size in angle brackets <>.

- [d13] : Out-of-Area Processing

This bit selects either to make out-of-area regions transparent or to wrap around and display when the BG screen does not lie entirely within the display screen because of affine transformations.

<b>0</b>	Transparent display
<b>1</b>	Wraparound display

The difference between the two Out-of-Area processing methods is shown in Figure 5-1.

**Figure 5-1 : Out-of-Area Processing Method Differences**

- SB4-SB0 [d12–d08] : Screen Base Block

Calculation of the base address for the screen base block differs according to the BG mode.

### 1. Character BG

These bits specify (in 2-KB units) the starting block in VRAM where screen data is stored. When screen data is actually referenced, the starting address is calculated as follows.

## 2D Graphics Engine A

The starting address is the sum of the DISPCNT register's screen base offset value with a 64-KB offset and the value of the screen base block with a 2-KB offset.

$$(\text{screen base offset} \times 0x10000) + (\text{screen base block} \times 0x800)$$

## 2D Graphics Engine B

The starting address is the screen base block with a 2-KB offset.

$$(\text{screen base block} \times 0x800)$$

### 2. 256-Color Bitmap BG and Direct Color Bitmap BG

These bits specify (in 16-KB units) the offset address in BG-VRAM where the bitmap data is stored. Because there is no relation to the screen base offset value in the DISPCNT register, the same calculation for the BG bitmap data starting address is used for both 2D Graphics Engine A and 2D Graphics Engine B.

#### 2D Graphics Engine A and 2D Graphics Engine B

$$(\text{screen base block} \times 0x4000)$$

### 3. Large-Screen 256-Color Bitmap BG

The screen base block value is invalid for 2D Graphics Engine A.

The BG mode cannot be set to large-screen 256-color bitmap BG for 2D Graphics Engine B.

- [d07, d02] : Affine Extended BG Type Selection (only with Affine Extended BG)

CM	CB0	Affine Extended BG Type
0	(See note)	256-color x 16-palette Character BG
1	0	256-color bitmap BG
1	1	Direct-color bitmap BG

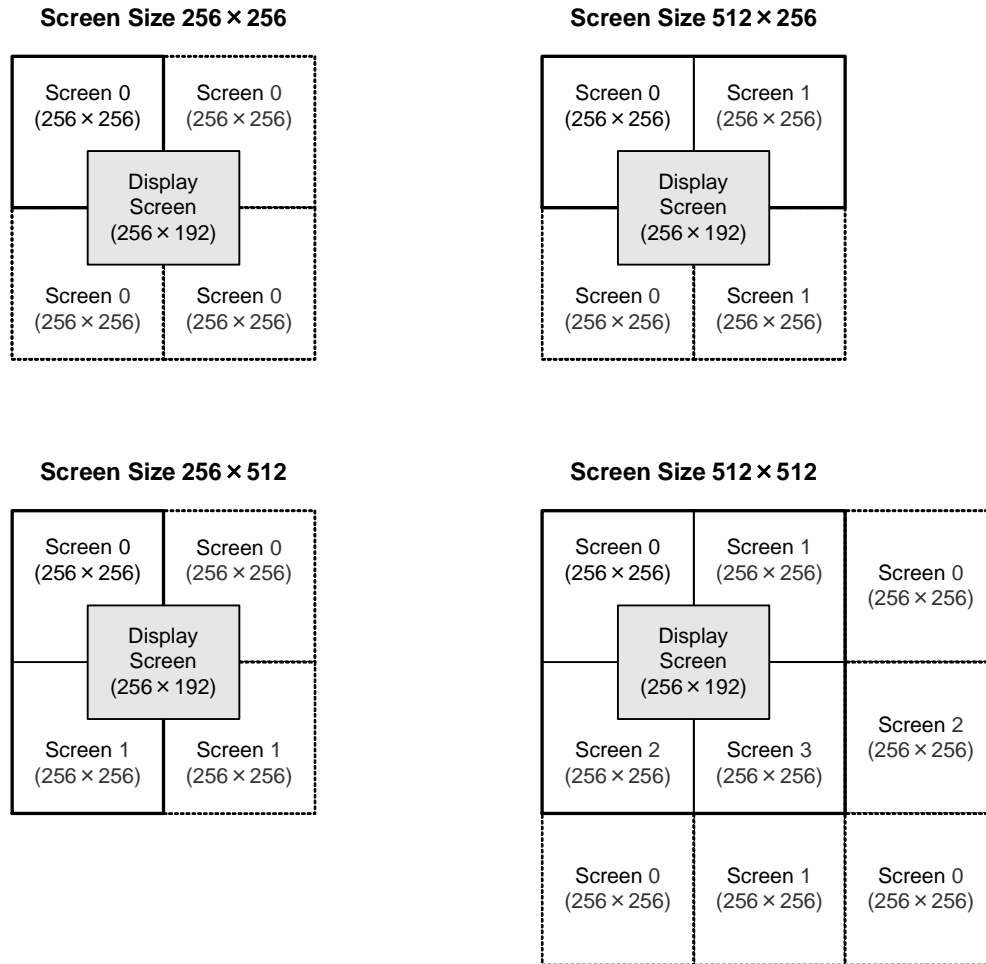
**Note:** When CM = 0, a unique 256-color x 16-palette Character BG is used and CB3-CB0 are handled as normal character base blocks.

## 5.2.2.1 Screen Sizes and Display Screens

### 5.2.2.1.1 Text BG

Text BG screen sizes are shown in Figure 5-2.

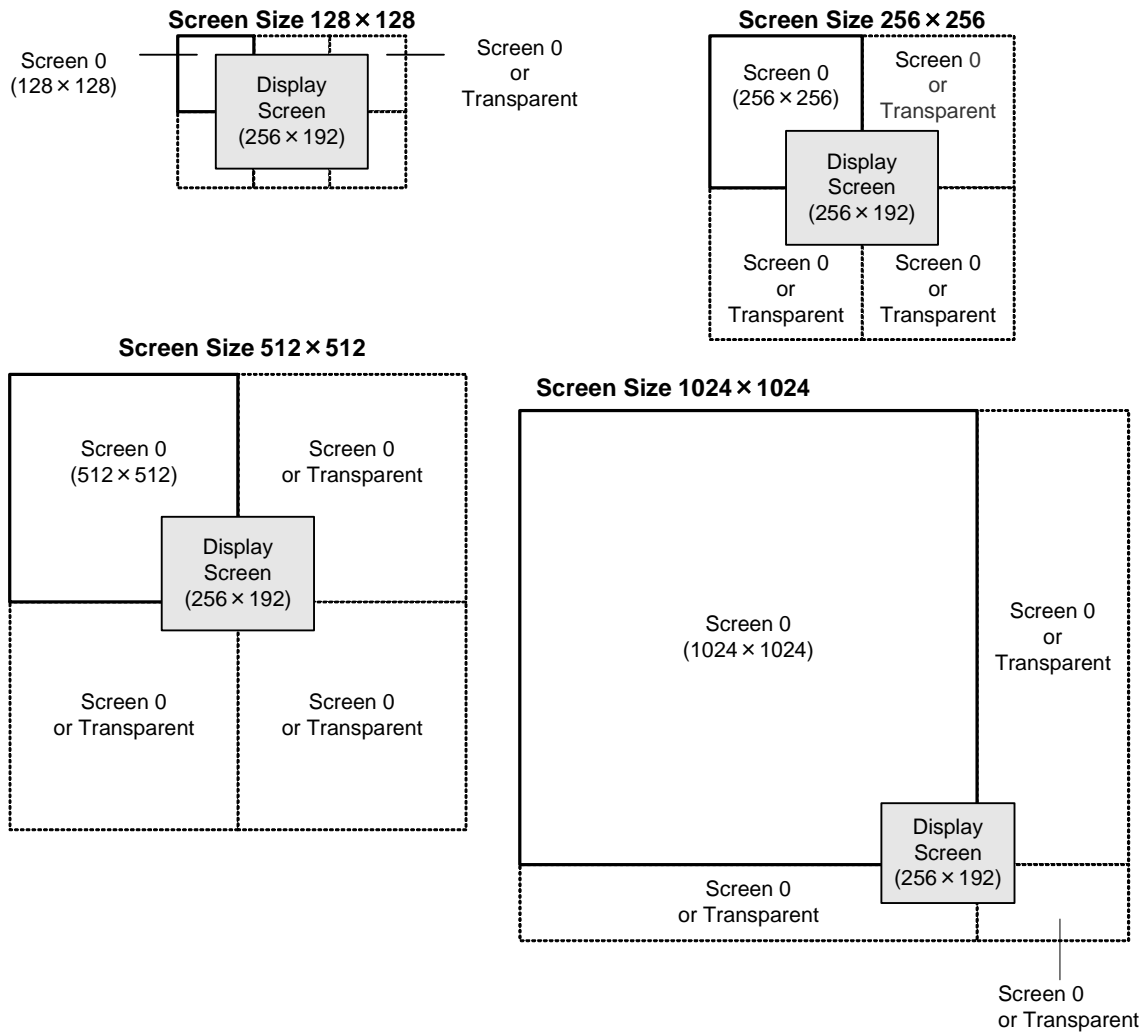
**Figure 5-2 : Text BG Screen Size**



### 5.2.2.1.2 Affine BG

Affine BG screen sizes are shown in Figure 5-3.

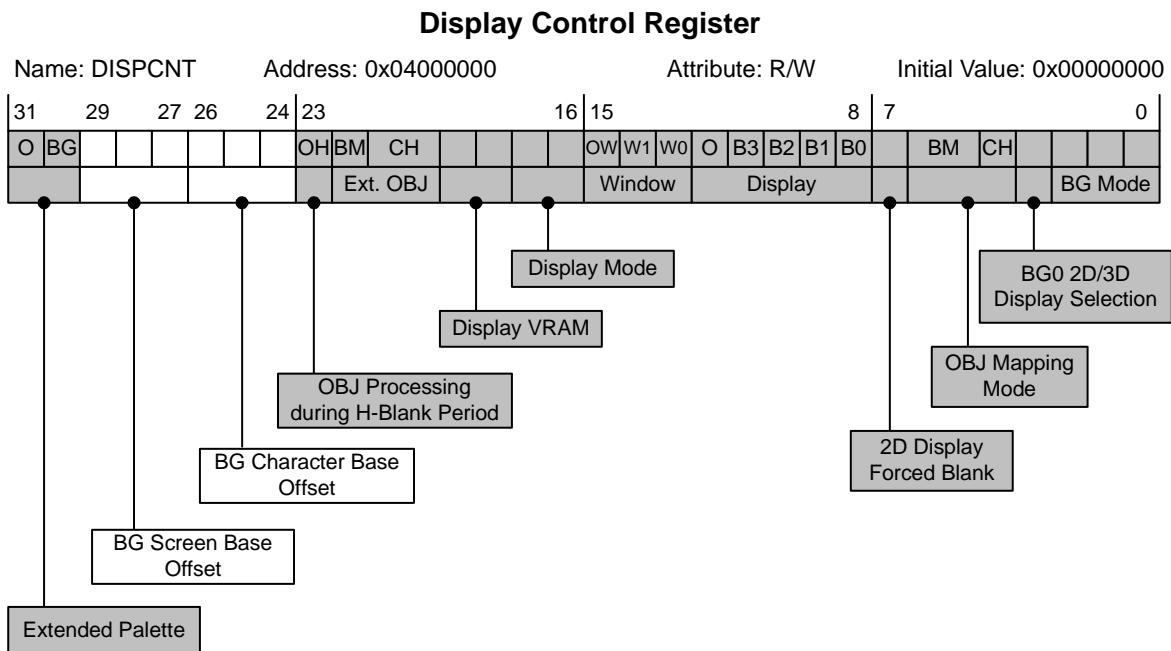
**Figure 5-3 : Affine BG Screen Size**



### 5.2.3 Character BG

For character BG, BG screen composition elements are treated as characters of 8 x 8 dots (8x8-dot). Accordingly, character data is required to display the BG. In addition, character index data for each 8x8-dot unit is required; this character index data is called *screen data*.

**Note:** 2D Graphics Engine B differs from 2D Graphics Engine A in that there are no settings for the BG screen base offset and BG character base offset.



- [d29–d27] : BG Screen Base Offset

These bits offset (in 64-KB units) the base address of the screen data set with the BG Control Register.

Accordingly, the base address of the BG screen data is calculated as follows:

The value set in the BG Control Register + (BG screen base offset x 0x10000)

An arbitrary base address can be specified from a maximum 512 KB of BG-VRAM space.

- [d26–d24] : BG Character Base Offset

These bits offset (in 64-KB units) the base address of the screen data set with the BG Control Register.

Accordingly, the base address of the BG character data is calculated as follows:

The value set in the BG Control Register + (BG character base offset x 0x10000)

An arbitrary base address can be specified from a maximum 512 KB of BG-VRAM space.

### 5.2.3.1 VRAM Maps of BG Data

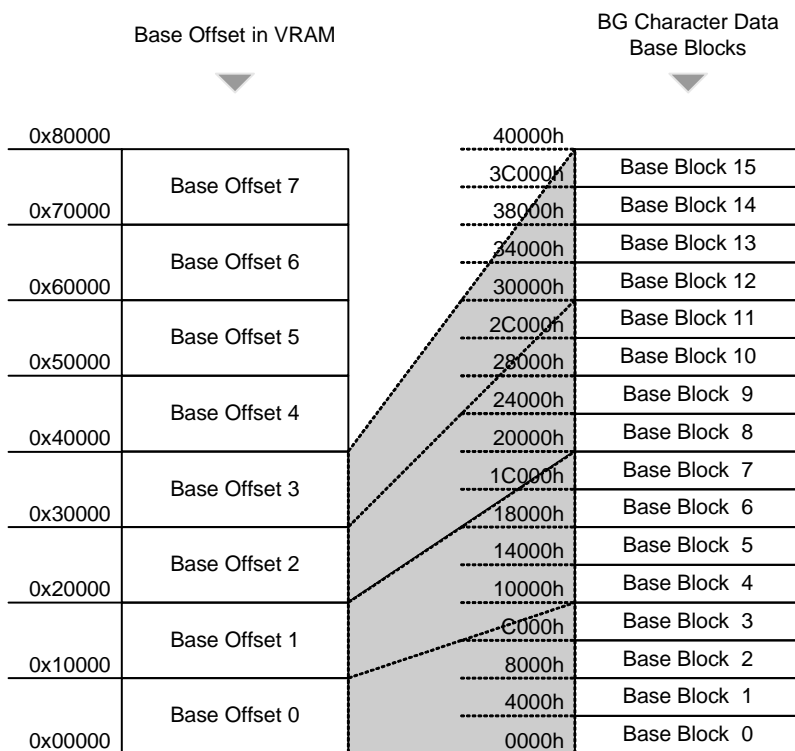
Character BG requires both BG screen data and BG character data. Store both the BG screen data and the BG character data in VRAM that was allocated to BG-VRAM by the RAM Bank Control Register. BG-VRAM can be assigned up to a maximum of 512 KB with 2D Graphics Engine A and up to a maximum of 128 KB with 2D Graphics Engine B.

#### 1. BG Character Data

With 2D Graphics Engine A, the starting address for referencing BG character data can be set by specifying the DISPCNT register's character base offset and the BG Control register's character base block. With 2D Graphics Engine B, there is no setting for character base offset. The VRAM offset for BG character data is shown in Figure 5-4.

The volume of data depends on the amount of registered character data and the format (Color Mode: 256 or 16 colors).

**Figure 5-4 : VRAM Offset for BG Character Data**



**Note:** For 2D Graphics Engine A, the maximum amount of VRAM that can be used for BG character data is 256 KB because the DISPCNT register's base offset cannot be set for each BG screen.

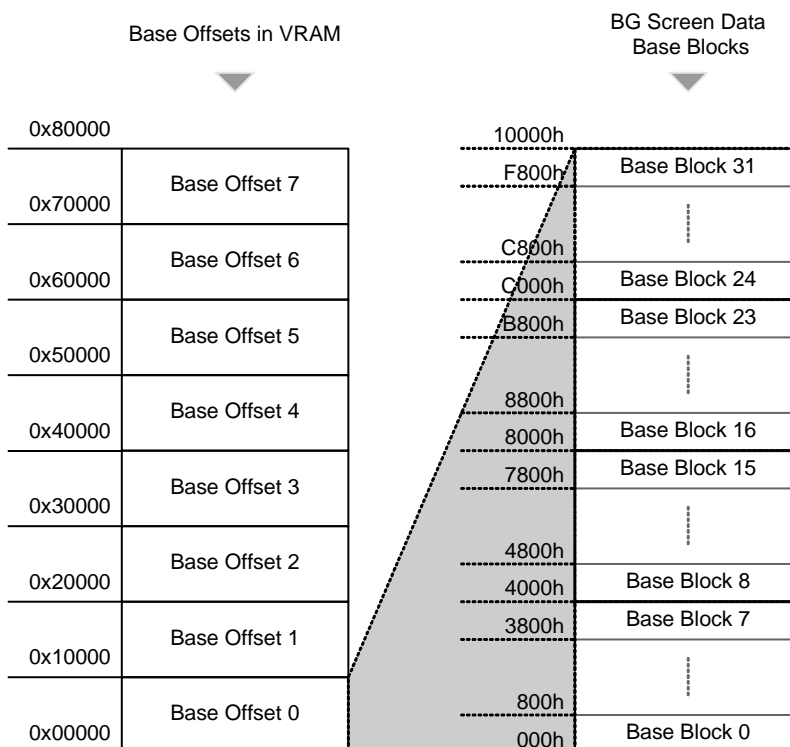
For 2D Graphics Engine B, the maximum amount of VRAM that can be used for BG character data is 128 KB because there are limitations on the size of BG-VRAM that can be allocated.

## 2. BG Screen Data

For 2D Graphics Engine A, the starting address for referencing BG screen data can be set by specifying the DISPCNT register's screen base offset and the BG Control register's screen base block. With 2D Graphics Engine B, there is no setting for the screen base offset. The VRAM offset for BG screen data is shown in Figure 5-5.

The volume of data depends on the BG type (Text BG or Affine BG) and the screen size.

**Figure 5-5 : VRAM Offset for BG Screen Data**



**Note:** For 2D Graphics Engine A, the maximum amount of VRAM that can be used for BG screen data is 64 KB because the DISPCNT register's base offset cannot be set for each BG screen. With 2D Graphics Engine B, the maximum amount of VRAM that can be used for BG screen data is 64 KB.



### 5.2.3.2 Text BG

#### 5.2.3.2.1 Screen Data Format

Store the BG screen data starting from the starting address of the BG screen base block specified by the BG Control Register. BG screen data for Text BG screens is configured using the following format:

**Text BG Screen Data**

15					12	11	10	9	8	7							0
						VF	HF										
Color Palette					Flip		Character Name										

- [d15–d12] : Color Palette

Palettes applied to characters are specified in the range of 0-15. The Color Palette specification is enabled with 256 colors x 16 palettes or 16 colors x 16 palettes, but it is disabled with 256 colors x 1 palette.

- [d11–d10] : Flip

- VF: Vertical Flip Flag    HF: Horizontal Flip Flag

<b>0</b>	Do not flip
<b>1</b>	Flip

- [d09–d00] : Character Name

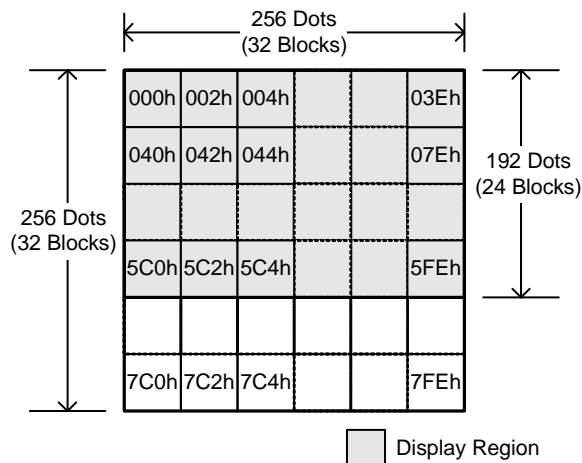
These bits specify the character number of the character that serves as the origin of the starting address for the character base block specified by the BG Control Register.

### 5.2.3.2.2 Screen Data Address Mapping

#### 1. 256x256-Dot Screen Size

Figure 5-6 shows the address map for screen data with a 256x256-dot screen size.

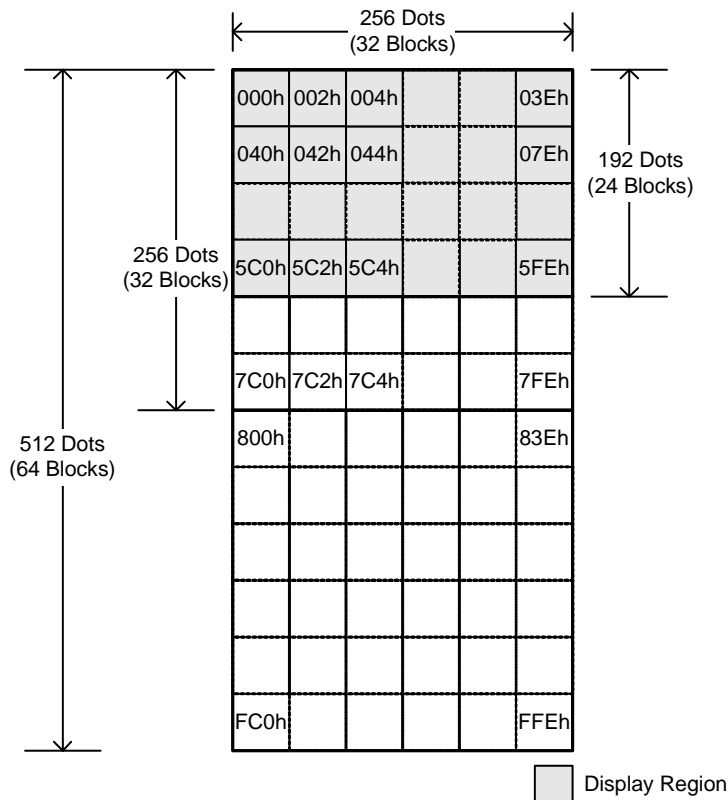
**Figure 5-6 : 256x256-Dot Address Mapping (Text BG)**



#### 2. 256x512-Dot Screen Size

Figure 5-7 shows the address map for screen data with a 256x512-dot screen size.

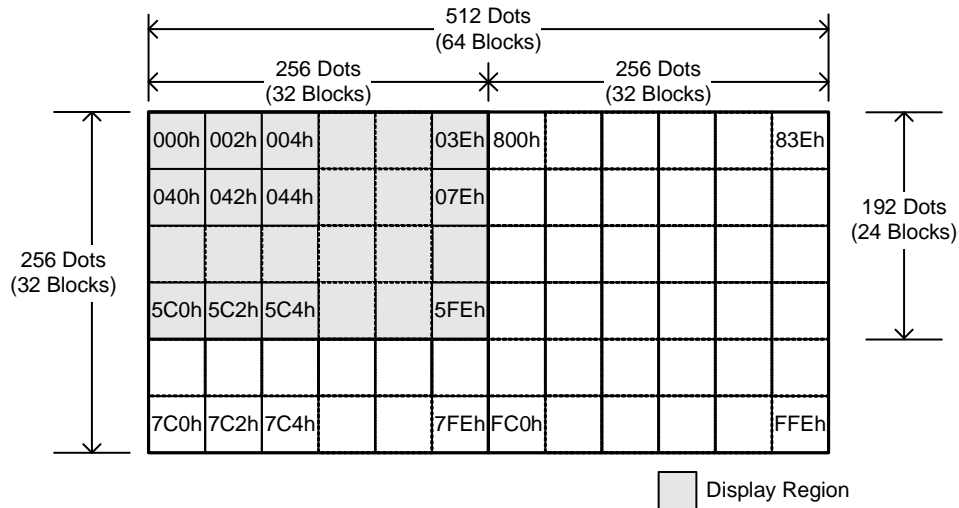
**Figure 5-7 : 256x512-Dot Address Mapping (Text BG)**



### 3. 512x256-Dot Screen Size

Figure 5-8 shows the address map for screen data with a 512x256-dot screen size.

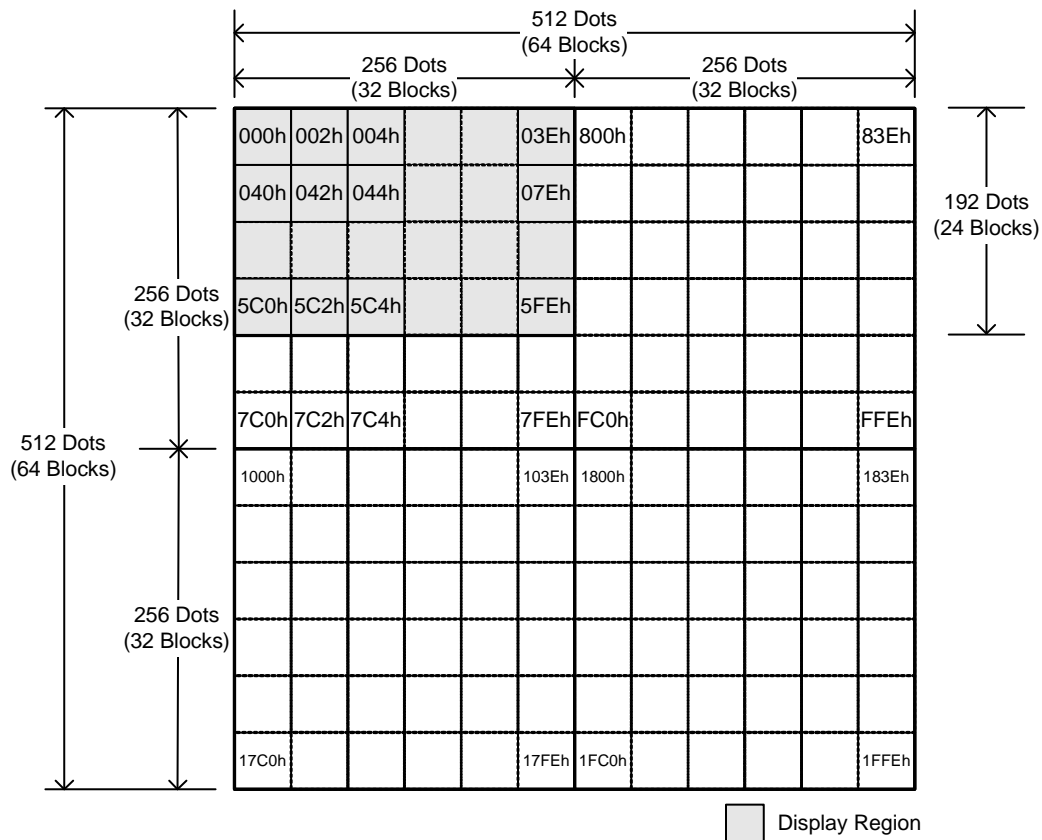
**Figure 5-8 : 512x256-Dot Address Mapping (Text BG)**



#### 4. 512x512-Dot Screen Size

Figure 5-9 shows the address map for screen data with a 512x512-dot screen size.

**Figure 5-9 : 512x512-Dot Address Mapping (Text BG)**



### 5.2.3.2.3 Character Data Formats

The character data formats for Text BG 16-color mode and Text BG 256-color mode are shown below. The Character Display table shows the case when an 8x8-dot character is defined.

#### 5.2.3.2.3.1 16-Color Mode

The character data format for 16-color mode, correspondence between character display and pixel data, and address mapping (Figure 5-10) are shown below.

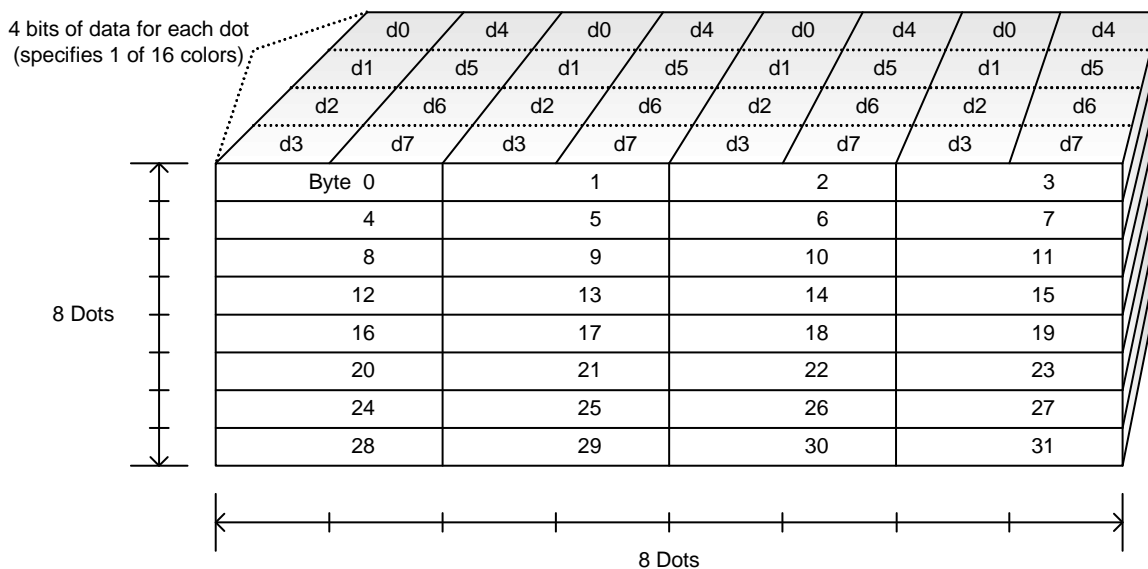
**16-Color Mode Character Data Format**

15	12	11	8	7	4	3	0
P3		P2		P1		P0	
4 pixels worth of data (4 bits/pixel)							

**Character Display**

P0	P1	P2	P3				

**Figure 5-10 : Character Data Address Mapping (Text BG 16-Color Mode)**



5.2.3.2.3.2 256-Color Mode

The character data format for 256-color mode, correspondence between character display and pixel data, and address mapping (Figure 5-11) are shown below.

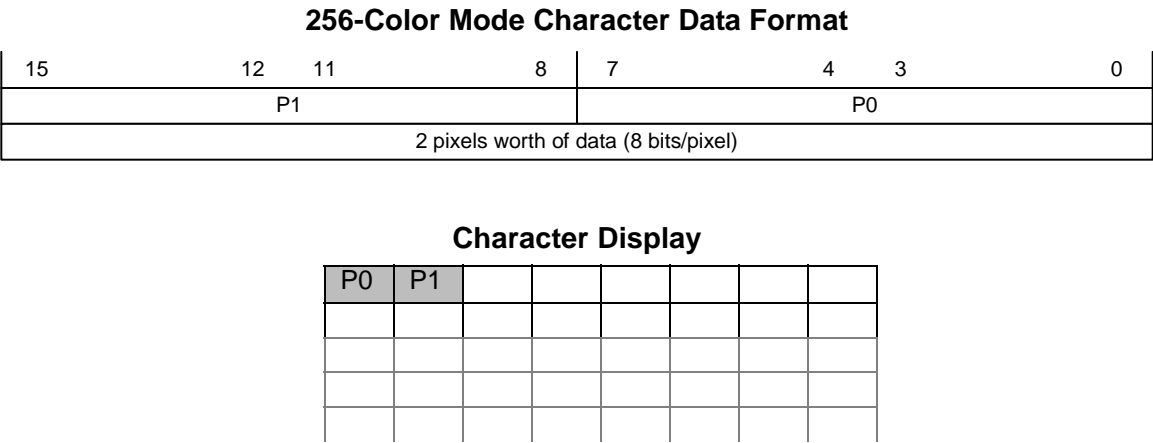
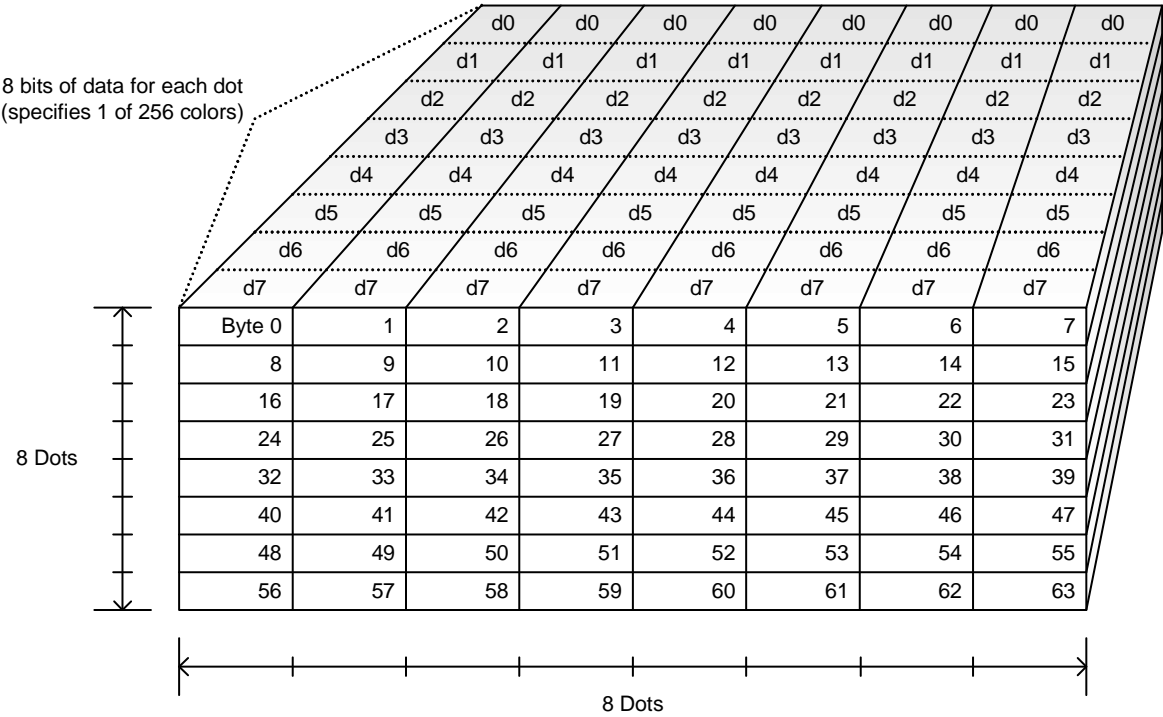


Figure 5-11 : Character Data Address Mapping (Text BG 256-Color Mode)



### 5.2.3.3 Affine BG

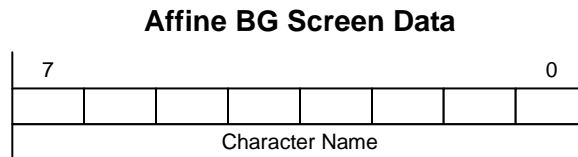
This character format BG type can be rotated and scaled.

**Note:** Affine BG can be set only with BG2 and BG3. The color mode for Affine BG screens is fixed to 256-color mode. Consequently, the BG Control Register's color-mode setting is disabled. Furthermore, horizontal and vertical flips cannot be performed on Affine BG.

#### 5.2.3.3.1 Screen Data Format

Store the BG screen data starting from the starting address of the BG screen base block specified by the BG Control register.

BG screen data for Affine BG screens is configured using the following format:



- [d07–d00] : Character Name

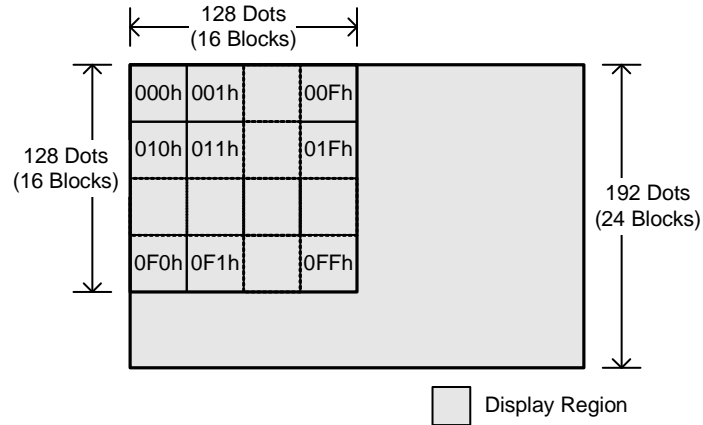
These bits specify the character number of the character that serves as the origin of the starting address for the character base block specified by the BG Control Register.

### 5.2.3.3.2 Screen Data Address Mapping

#### 1. 128X128-Dot Screen Size

Figure 5-12 shows the address map for screen data with a 128x128-dot screen size.

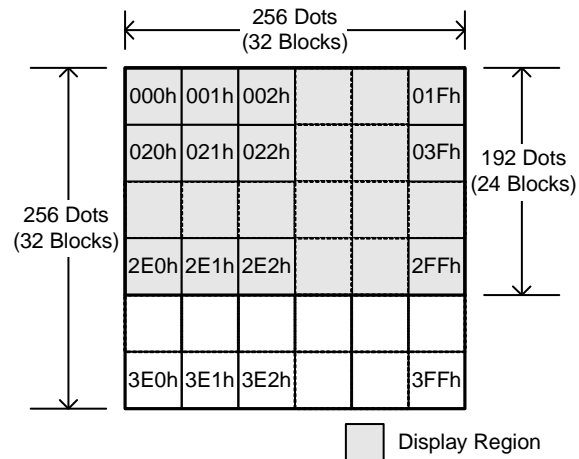
**Figure 5-12 : 128X128-Dot Address Mapping (Affine BG)**



#### 2. 256x256-Dot Screen Size

Figure 5-13 shows the address map for screen data with a 256x256-dot screen size.

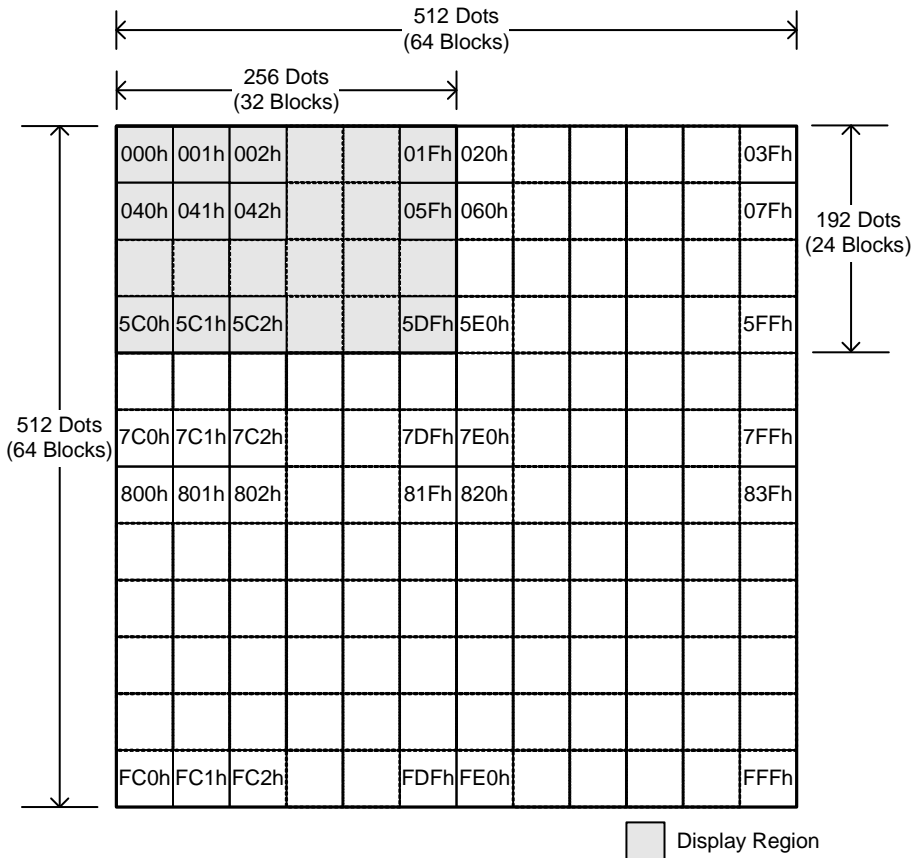
**Figure 5-13 : 256x256-Dot Address Mapping (Affine BG)**



3. 512x512-Dot Screen Size

Figure 5-14 shows the address map for screen data with a 512x512-dot screen size.

Figure 5-14 : 512x512-Dot Address Mapping (Affine BG)







5.2.3.3.3 Character Data Format

The character data format for Affine BG screens is shown below. The Character Display table shows the case when an 8x8-dot character is defined. Figure 5-16 shows the character data address mapping.

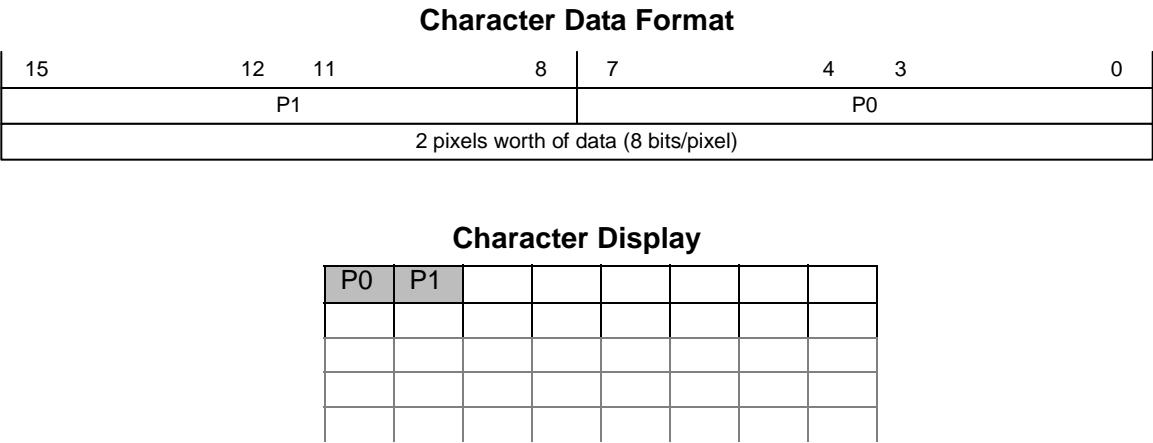
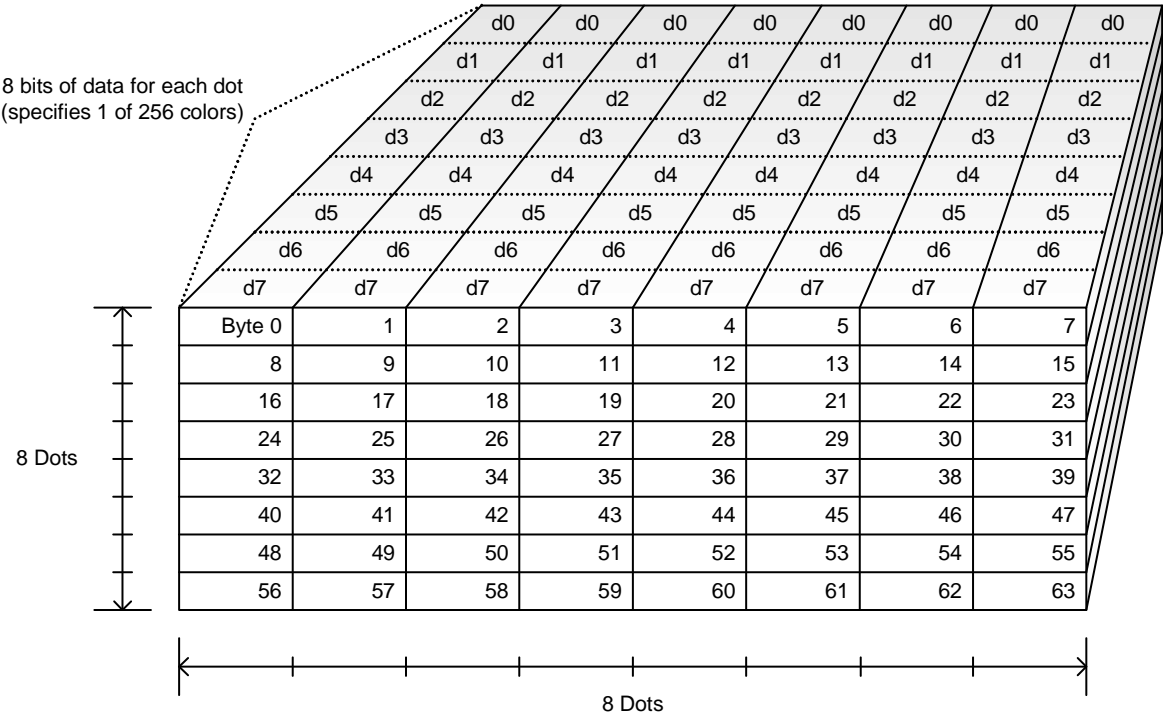


Figure 5-16 : Character Data Address Mapping (Affine BG)



### 5.2.3.4 256-Color x 16-Palette Character BG (Affine Extended BG)

Select 256-Color x 16-Palette Character BG with the BG Control Register. Select the 256-Color x 16-Palette Character BG by selecting Affine Extended BG as the BG type and setting the BG Control Register Color Mode to 0.

**Note:** 256-Color x 16-Palette Character BG can be set only for BG2 and BG3.

#### 5.2.3.4.1 Screen Data Format

**256-Color x 16-Palette BG Screen Data**

15				12		11		10		9		8		7													0
						VF		HF																			
Color Palette					Flip			Character Name																			

- [d15–d12] : Color Palette

When enabled, extended palettes applied to characters are specified in the range of 0-15. When extended palettes are disabled, standard palettes are used. Extended palettes are enabled/disabled with the DISPCNT [DB\_DISPCNT] Register.

- [d11–d10] : Flip

- VF: Vertical Flip Flag HF: Horizontal Flip Flag

<b>0</b>	Do not flip
<b>1</b>	Flip

- [d09–d00] : Character Name

These bits specify the character number of the character that serves as the origin of the starting address for the character base block specified by the BG Control Register.

5.2.3.4.2 Character Data Format

Character data format is the same as for 256-Color Mode Text BG.

Figure 5-17 shows the character data address mapping.

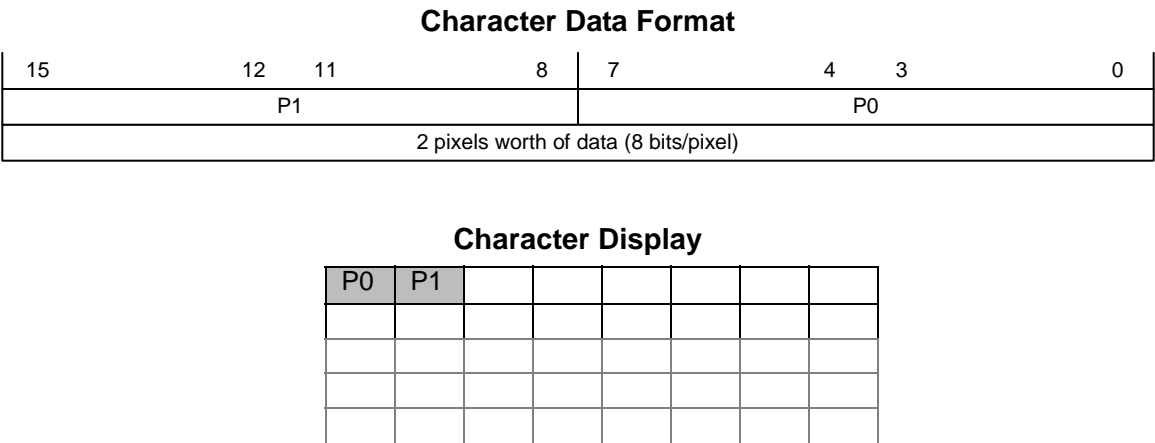
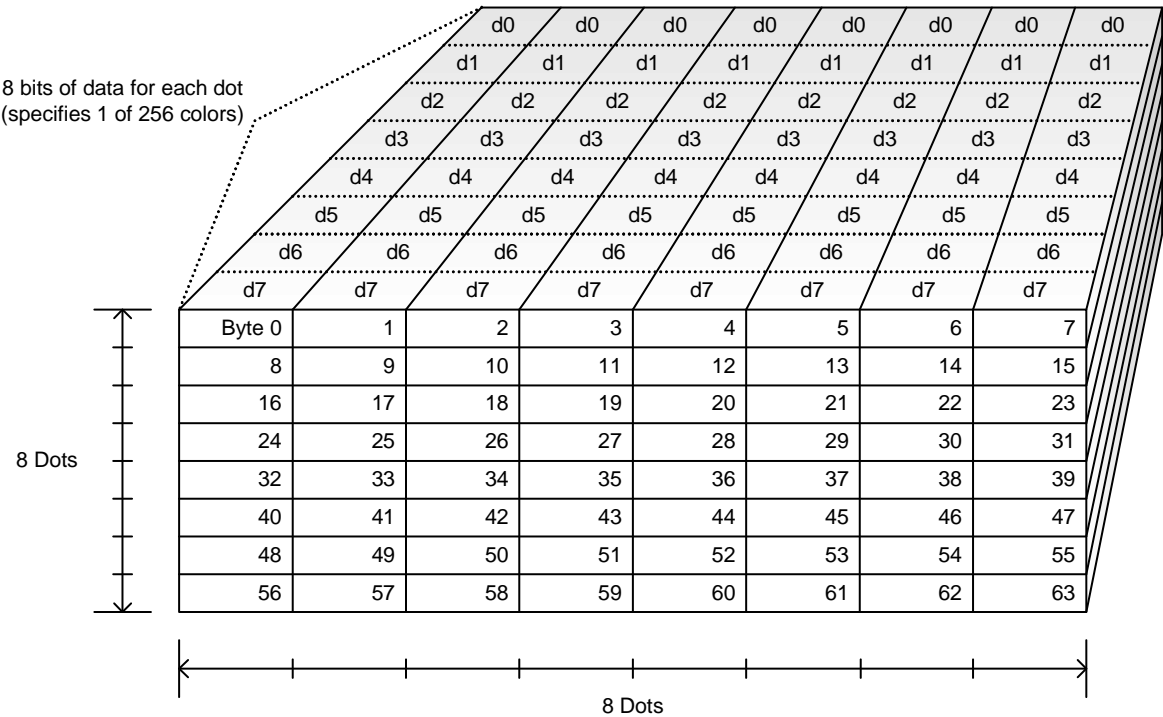


Figure 5-17 : Character Data Address Mapping (256-Color x 16-Palette Character BG)



## 5.2.4 Bitmap BG

For Bitmap BG, BG screen composition elements are treated as pixels and the contents of VRAM (frame buffer) are displayed as color data for each individual screen pixel.

**Note:** 256-Color Bitmap BG and Direct-Color Bitmap BG can be set only for BG2 and BG3. Large-Screen 256-Color Bitmap BG can be set only for BG2 with the 2D Graphics Engine A.

### 5.2.4.1 256-Color Bitmap BG (Affine Extended BG)

Select 256-Color Bitmap BG with the BG Control Register.

Select the 256-Color Bitmap BG by selecting Affine Extended BG as the BG type and setting the BG Control Register Color Mode to 1 and the Character Base Block CB0 to 0.

#### 5.2.4.1.1 Pixel Data Format

The pixel data format for 256-Color Bitmap BG is shown below.

**256-Color Bitmap BG Pixel Data Format**

7							0
Color Number							

#### 5.2.4.1.2 Pixel Data VRAM Map

The screen base address set in the BG Control Register specifies (in 16-KB units) the address in VRAM where the bitmap data is stored. The DISPCNT register's screen base offset value is invalid.

### 5.2.4.2 Direct-Color Bitmap BG (Affine Extended BG)

Select the Direct-Color Bitmap BG with the BG Control Register.

Direct-Color Bitmap BG can be selected by selecting Affine Extended BG as the BG type and setting the BG Control Register Color Mode to 1 and the Character Base Block CB0 to 1.

#### 5.2.4.2.1 Pixel Data Format

The pixel data format for Direct-Color Bitmap BG is shown below.

**Direct-Color Bitmap BG Pixel Data Format**

15	14				10	9	8	7		5	4				0
$\alpha$	BLUE					GREEN					RED				

#### 5.2.4.2.2 Pixel Data VRAM Map

The screen base address set in the BG Control Register specifies (in 16-KB units) the address in VRAM where the bitmap data is stored. The DISPCNT register's screen base offset value is invalid.

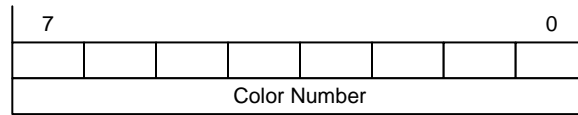
### 5.2.4.3 Large-Screen 256-Color Bitmap BG

The Large-Screen 256-Color Bitmap BG cannot be used with 2D Graphics Engine B.

#### 5.2.4.3.1 Pixel Data Format

The pixel data format for Large-Screen 256-Color Bitmap BG is shown below.

#### Large-Screen 256-Color Bitmap BG Pixel Data Format



#### 5.2.4.3.2 Pixel Data VRAM Map

The starting address for pixel data is fixed to the starting address of BG-VRAM (0x6000000). Both the BG Control Register's screen base address and the DISPCNT Register's screen base offset value are invalid.

## 5.2.5 BG Scroll

For each Text BG screen, the display screen and its offset value can be set in dots.

The Offset register is valid only for Text BG. Set the BG reference starting point (see "[5.2.6 BG Rotation and Scaling \(Affine Transformation\)](#)" on page 106) to display Affine BG and Bitmap Mode BG with an offset.

### BG Offset Setting Registers

Name	Address	Attribute	Initial Value
(2D_A) BGxOFS(x=0 - 3)	0x04000010, 0x04000014, 0x04000018, 0x0400001C	W	0x00000000
(2D_B) DB_BGxOFS(x=0 - 3)	0x04001010, 0x04001014, 0x04001018, 0x0400101C	W	0x00000000

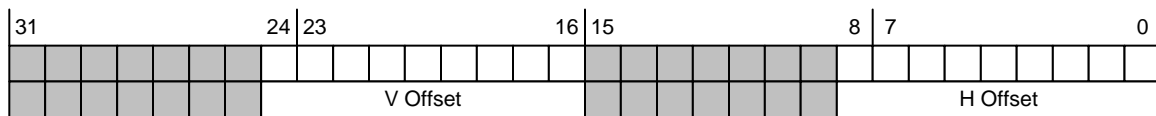
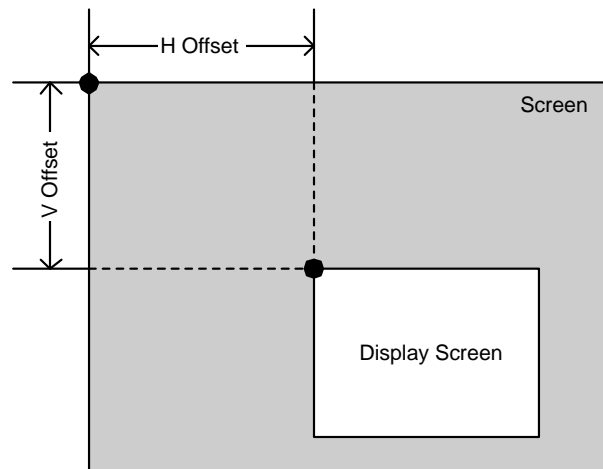


Figure 5-18 shows the offset for BG scrolling.

**Figure 5-18 : Offset Schematic**

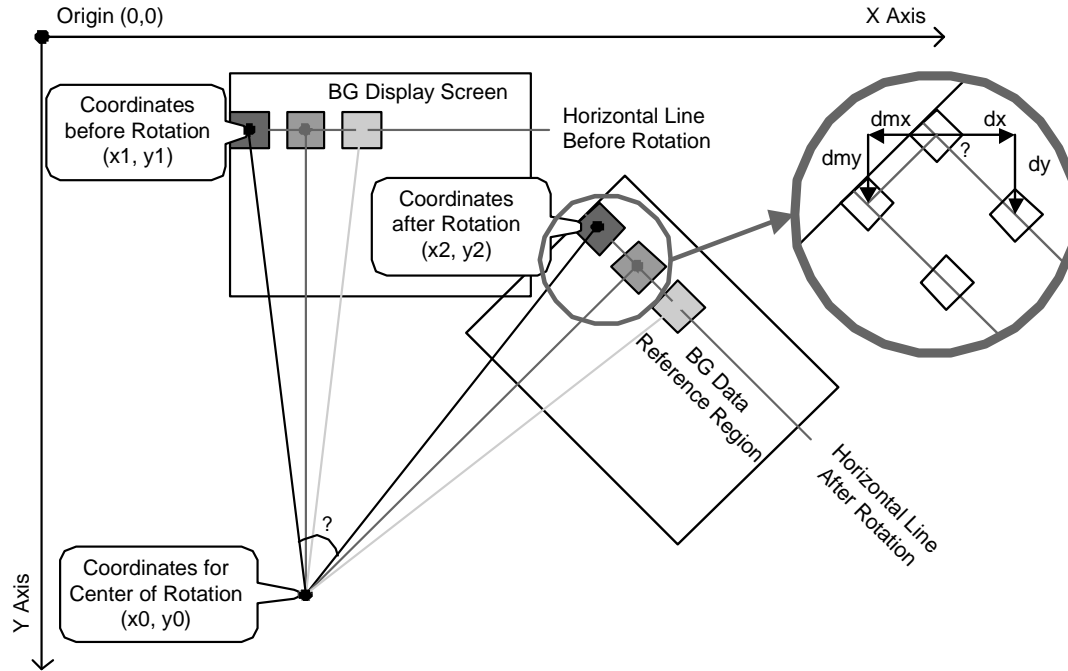


## 5.2.6 BG Rotation and Scaling (Affine Transformation)

The BG pixels are referenced horizontally in sequence from the top left when a BG is displayed, so a rotated BG can be displayed by rotating the reference direction.

Figure 5-19 shows the rotation and scaling process for a BG.

**Figure 5-19 : BG Rotation and Scaling**



$dx$  (reference distance in x-direction for same line) =  $(1/\alpha)\cos\theta$

$dy$  (reference distance in y-direction for same line) =  $-(1/\beta)\sin\theta$

$dmx$  (reference distance in x-direction for next line) =  $(1/\alpha)\sin\theta$

$dmy$  (reference distance in y-direction for next line) =  $(1/\beta)\cos\theta$

**Note:**  $\alpha$  is the scale ratio along the x-axis;  $\beta$  is the scale ratio along the y-axis.

The  $(x2, y2)$  coordinates correspond to the  $(x1, y1)$  coordinates after affine transformation and are calculated with the following formula:

$$\begin{bmatrix} x2 \\ y2 \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x1 - x0 \\ y1 - y0 \end{bmatrix} + \begin{bmatrix} x0 \\ y0 \end{bmatrix}$$

$$A = \frac{1}{\alpha}\cos\theta, B = \frac{1}{\alpha}\sin\theta, C = -\frac{1}{\beta}\sin\theta, D = \frac{1}{\beta}\cos\theta$$



- BG Rotation and Scaling Process

1. Using the equation above, calculate the results of affine transformation for the top-left coordinates of the display screen, and then set the results as the BG data reference start point in the following registers:

- 2D Graphics Engine A BGxX, BGxY Registers (x=2, 3)
- 2D Graphics Engine B DB\_BGxX, DB\_BGxY Registers (x=2, 3)

Also, refer to "[Figure 5-19 : BG Rotation and Scaling](#)" on page 106, and set the BG data reference direction in the following registers:

- 2D Graphics Engine A BGxPA, BGxPB, BGxPC, BGxPD Registers (x=2, 3)
- 2D Graphics Engine B DB\_BGxPA, DB\_BGxPB, DB\_BGxPC, DB\_BGxPD Registers (x=2, 3)

2. The image processing circuitry sums the cumulative increase in the x-direction (dx and dy) and calculates the x-direction coordinates in relation to the BG data reference start point set in these registers.
3. If the line advances, the rendering start point coordinates for the next line are calculated by summing the cumulative increase in the y-direction (dmx and dmy) in relation to the reference start point. Then the process in Step 2 is performed.
4. If the BG Data Reference Start Point Registers are overwritten during an H-Blank, the cumulative sum for the y-direction related to those registers is not computed. Use this mode to have the CPU change the affine transformation parameters and the center coordinates for each line.

## BG Data Reference Start Point Setting Registers

Name	Address	Attribute	Initial Value
(2D_A) BGxX (x=2, 3)	0x04000028, 0x04000038	W	0x00000000
(2D_B) DB_BGxX (x=2, 3)	0x04001028, 0x04001038	W	0x00000000

31					24	23					16	15					8	7					0								
					S	INTEGER_SX											DECIMAL_SX														
					x-Coordinate of the Reference Start Point (Affine Transformation Result)																										

Name	Address	Attribute	Initial Value
(2D_A) BGxY (x=2, 3)	0x0400002C, 0x0400003C	W	0x00000000
(2D_B) DB_BGxY (x=2, 3)	0x0400102C, 0x0400103C	W	0x00000000

31						24	23					16	15					8	7					0		
					S	INTEGER_SY														DECIMAL_SY						
					y-Coordinate of the Reference Start Point (Affine Transformation Result)																					

**BG Data Reference Direction Setting Registers**

Name	Address	Attribute	Initial Value
(2D_A) BGxPA (x=2, 3)	0x04000020, 0x04000030	W	0x0100
(2D_B) DB_BGxPA (x=2, 3)	0x04001020, 0x04001030	W	0x0100

15	8	7	0
S	INTEGER_DX	DECIMAL_DX	
Reference distance dx in x-direction for the same line			

Name	Address	Attribute	Initial Value
(2D_A) BGxPB (x=2, 3)	0x04000022, 0x04000032	W	0x0000
(2D_B) DB_BGxPB (x=2, 3)	0x04001022, 0x04001032	W	0x0000

15	8	7	0
S	INTEGER_DMX	DECIMAL_DMX	
Reference distance dmx in x-direction for the next line			

Name	Address	Attribute	Initial Value
(2D_A) BGxPC (x=2, 3)	0x04000024, 0x04000034	W	0x0000
(2D_B) DB_BGxPC (x=2, 3)	0x04001024, 0x04001034	W	0x0000

15	8	7	0
S	INTEGER_DY	DECIMAL_DY	
Reference distance dy in y-direction for the same line			

Name	Address	Attribute	Initial Value
(2D_A) BGxPD(x=2, 3)	0x04000026, 0x04000036	W	0x0100
(2D_B) DB_BGxPD(x=2, 3)	0x04001026, 0x04001036	W	0x0100

15	8	7	0
S	INTEGER_DMY	DECIMAL_DMY	
Reference distance dmy in y-direction for the next line			

### 5.3 OBJ

NITRO can handle two types of OBJ: Character OBJ and Bitmap OBJ. Table 5-7 summarizes the features for both Character OBJ and Bitmap OBJ.

**Table 5-7 : OBJ Overview**

Item	Character OBJ	Bitmap OBJ
<b>Number of Display Colors</b>	1. Standard Palette 16 colors x 16 palettes 256 colors x 1 palette  2. Extended Palette 16 colors x 16 palettes (Standard Palette) 256 colors x 16 palettes (Extended Palette)	32,768 colors
<b>Number of Characters (Converted to 8x8-Dot)</b>	1. One-dimensional Mapping 1,024 to 8,192 (16-color mode) 512 to 4,096 (256-color mode)  2. Two-dimensional Mapping 1,024 (16-color mode) 512 (256-color mode)	1. 1D Mapping 1,024 to 2,048  2. 2D Mapping 256
<b>Character Size</b>	8x8 dot to 64x64 dot (12 varieties)	
<b>Maximum Number Displayed on One Screen</b>	128 (converted to 64x64 dot)	
<b>Maximum Number Displayed on One Line</b>	128 (converted to 8x8 dot)	
<b>Features</b>	HV Offset, HV Flip, Affine Transformation, Translucence (see note), Mosaic, and Priority settings	

**Note:** See "[5.7 Color Special Effects](#)" on page 146 to learn about OBJ color effects.

- Other items

See "[5.6 Windows](#)" on page 142 to learn about OBJ windows.

- Number of OBJ that can be displayed on one line

Table 5-7 gives the capacity of OBJ that can be displayed on one line under the most efficient conditions. When display OBJ are positioned in series from the start of OAM, the number of OBJ that can be displayed on one line is calculated as follows:

$$(H \text{ dot count} \times 6 - 6) / \text{rendering cycle count} = \text{Number of OBJ displayable on 1 line (128 max.)}$$

*H dot count* is normally 355 dots, but it becomes 256 dots if the DISPCNT [DB\_DISPCNT] Register's OBJ Processing during H-Blank Period flag is set to 1 (see "[4.2 LCD](#)" on page 49).

*x 6* represents the number of cycles that the OBJ rendering circuitry can use per dot. - 6 represents the number of cycles needed for the OBJ rendering pre-process at the start of the H-line.

Table 5-8 shows the relation between the *rendering cycle count* and the number of OBJ that can be displayed on one line.

**Table 5-8 : Rendering Cycle Count and Number of OBJ Displayable on One Line**

OBJ H-Size	Render Cycle Count		Number of OBJ Displayable on One Line	
	Normal OBJ* <sup>1</sup>	Affine OBJ* <sup>2</sup>	Normal OBJ	Affine OBJ
<b>8</b>	8	26	128	81
<b>16</b>	16	42	128	50
<b>32</b>	32	74	66	28
<b>64</b>	64	138	33	15
<b>128 (Double-Sized 64)</b>	—	266	—	7

**Note 1:** A Normal OBJ has the OBJ Mode of OBJ Attribute 0 set to Normal OBJ.

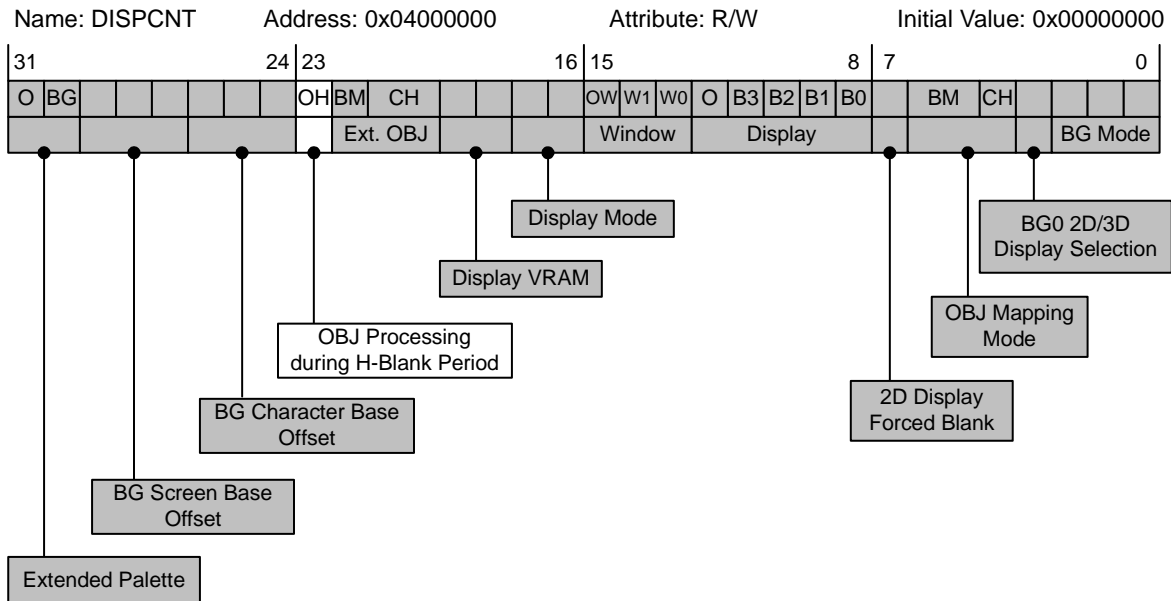
**Note 2:** An Affine OBJ has the Affine Enable Flag of OBJ Attribute 0 set to Enable.

Table 5-8 shows values under the most efficient conditions. Efficiency is actually lower because some OBJ in OAM are outside of the rendered line. Two cycles are lost for an OBJ outside of the rendered line.

### 5.3.1 OBJ Display Control

The overall configuration of OBJ features is performed with the DISPCNT Register for 2D Graphics Engine A and with the DB\_DISPCNT Register for 2D Graphics Engine B. The settings for an individual OBJ are configured with the OBJ Attribute Data stored in OAM. (This subject is touched on later.)

## Display Control Register (2D Graphics Engine A)

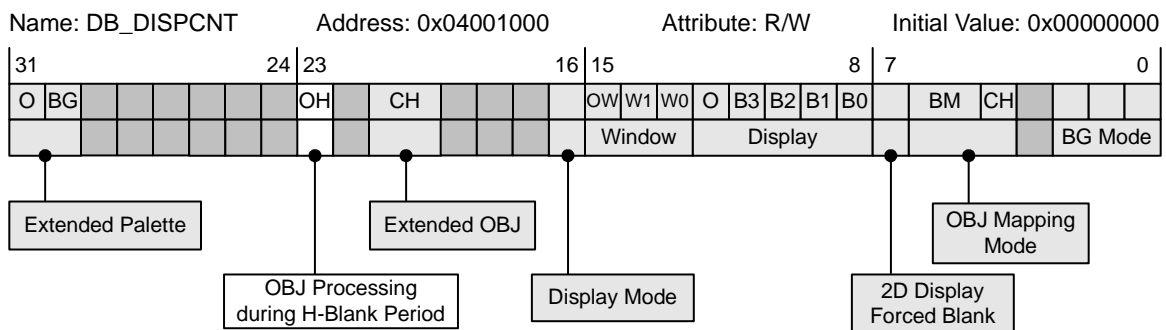


- OH [d23] : OBJ Processing during H-Blank Period Flag

When set to 0, the OBJ render process is performed during the entire H-line period (including the H-Blank period).

When set to 1, the OBJ render process is performed only during the display period, but not during the H-Blank period. In this case, the maximum number of OBJ cannot be displayed.

### Display Control Register 1 (2D Graphics Engine B)



- OH [d23] : OBJ Processing during H-Blank Period Flag

When set to 0, the OBJ render process is performed during the entire H-line period (including the H-Blank period).

When set to 1, the OBJ render process is performed only during the display period, but not during the H-Blank period. In this case, the maximum number of OBJ cannot be displayed.

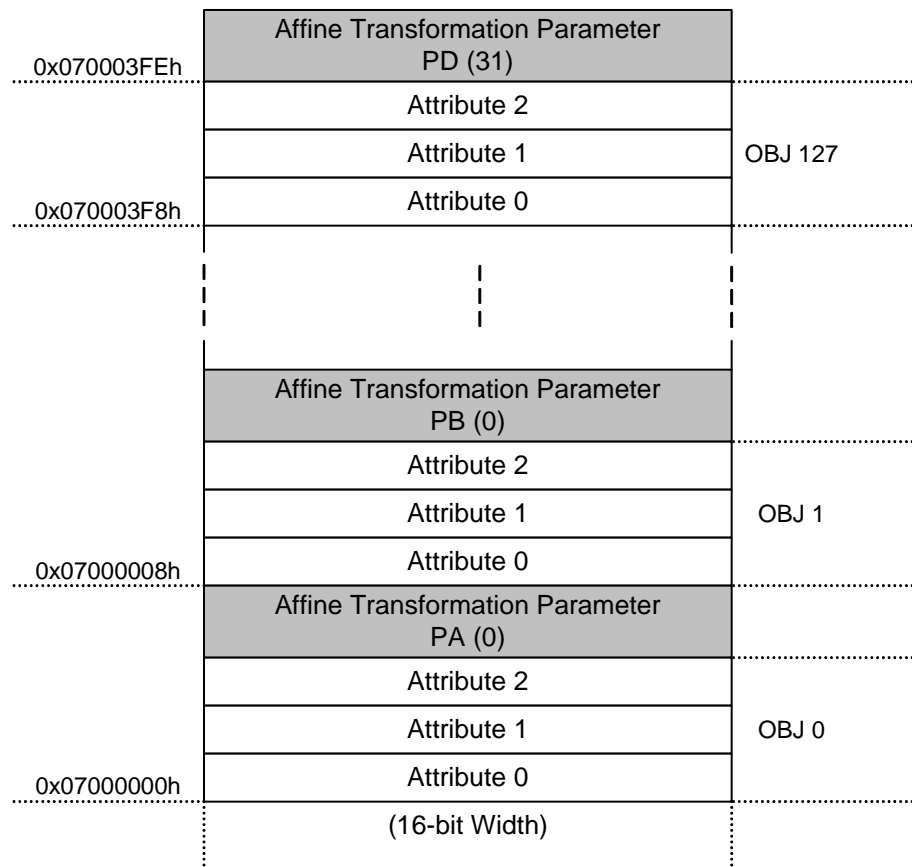
### 5.3.2 OAM

An OBJ is displayed by storing data in Object Attribute Memory (OAM). A total of 128 sets of OBJ data can be written to the NITRO Processor's internal OAM (1 KB from 07000000h – 070003FFh for 2D Graphics Engine A and 1 KB from 07000400h – 070007FFh for 2D Graphics Engine B). Accordingly, a total of 128 OBJ characters of any size can be displayed on the LCD.

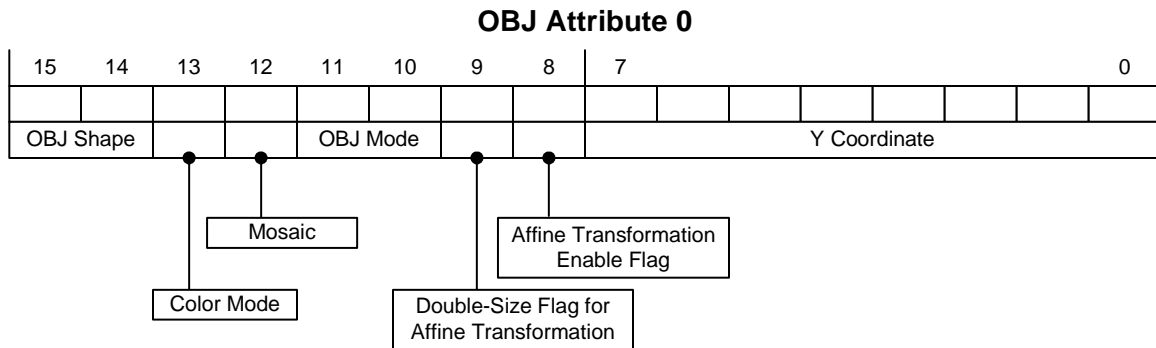
#### 5.3.2.1 Memory Map

48 bits x 128 sets of OBJ Attribute data can be written to OAM. If an OBJ is to be rotated and scaled, a total of 32 groups of affine transformation parameters PA, PB, PC, and PD can be written to OAM as shown in Figure 5-20.

**Figure 5-20 : OAM Memory Map (Add 0x400h to 2D Graphics Engine B Addresses)**



### 5.3.2.2 OAM Data Format



- [d15–d14] : OBJ Shape**

These bits set the shape of the OBJ. The number of dots in the OBJ's horizontal and vertical direction is determined by this setting and the OBJ size specification in OBJ Attribute 1. See OBJ Size under "[OBJ Attribute 1](#)" on page 116.

<b>00</b>	Square
<b>01</b>	Long rectangle
<b>10</b>	Tall rectangle
<b>11</b>	Prohibited code

- [d13] : Color Mode**

This bit sets whether the OBJ character data is referenced in 16-color format or 256-color format. Be sure to set Color Mode to 0 for Direct-Color Bitmap OBJ settings.

<b>0</b>	16-color mode
<b>1</b>	256-color mode

- [d12] : Mosaic**

<b>0</b>	Mosaic off
<b>1</b>	Mosaic on

- [d11–d10] : OBJ Mode**

The 00 to 10 settings specify Character OBJ. When 10 (the OBJ Window) is specified, data is not displayed as normal OBJ; if there are dots of non-zero character data, the data is handled as an OBJ window that can take any shape. See "[5.6 Windows](#)" on page 142 for display settings inside the OBJ window. The 11 setting specifies Bitmap OBJ.

<b>00</b>	Normal OBJ
<b>01</b>	Translucent OBJ
<b>10</b>	OBJ Window
<b>11</b>	Bitmap OBJ

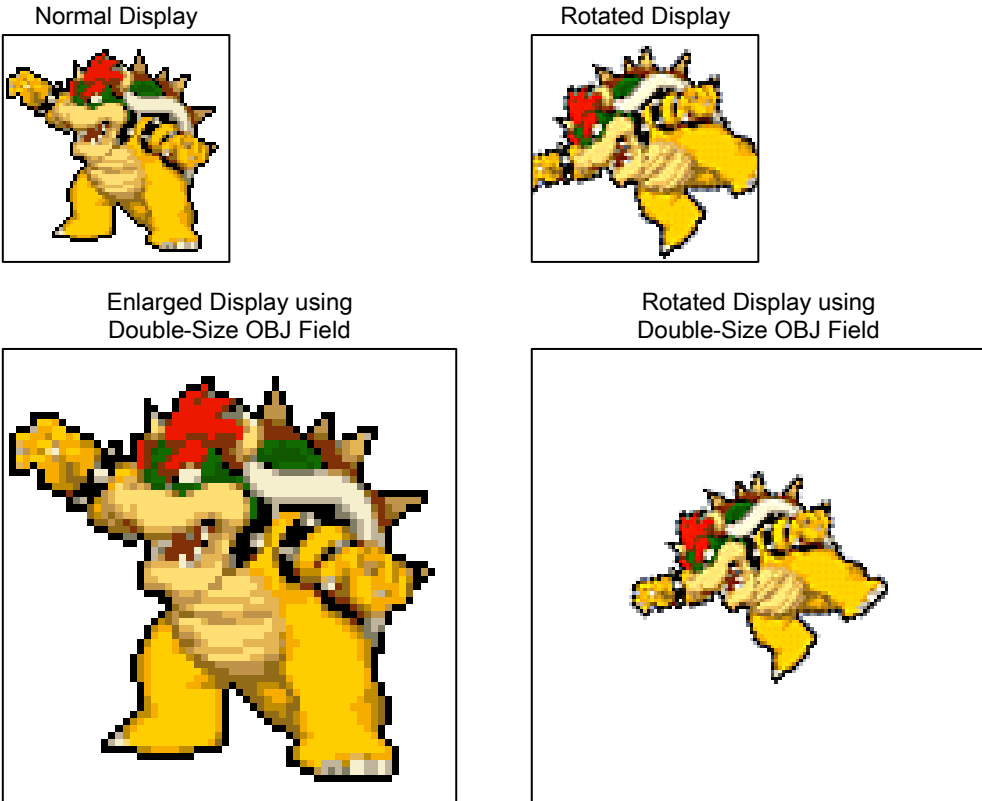
- [d09] : Double-Size Flag for Affine Transformation

0	Disable double-size
1	Enable double-size

When the d08 Affine Transformation Enable Flag is set to 1, the OBJ field can be doubled in size for display. Using a double-size OBJ field allows a rotated OBJ to be displayed in its entirety, without losing any sections. In addition, an OBJ can be enlarged up to double its original size and displayed without losing any sections (see Figure 5-21).

When the d08 Affine Transformation Enable Flag is set to 0 and this bit to 1, the OBJ is hidden.

**Figure 5-21 : Affine Transformation of Double-Size OBJ Field**



- [d08] : Affine Transformation Enable Flag

When this bit is enabled, the affine transformation parameters set in OBJ Attribute 1 are referenced.

When this bit is disabled and the d09 Double-Size Flag for Affine Transformation is set to 1, the OBJ is hidden.

0	Disable
1	Enable

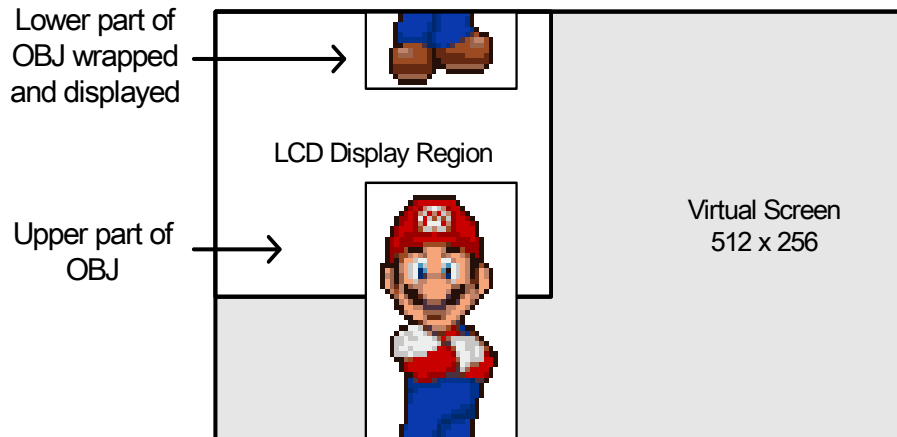
- [d07–d00] : Y Coordinate

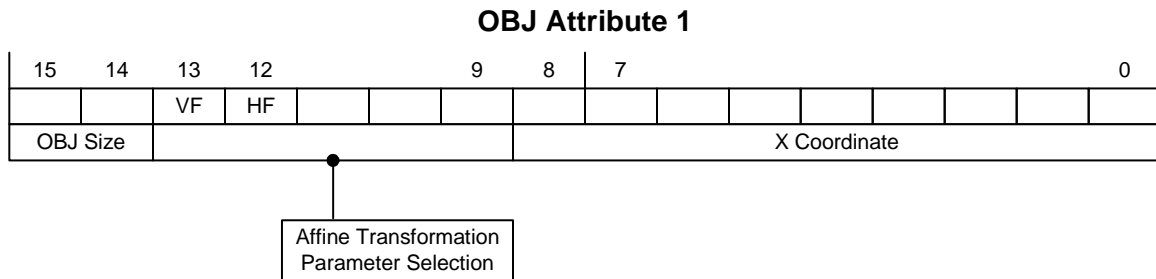
These bits specify the OBJ's Y coordinate in the display screen in the range of 0 to 255.



**Workaround for AGB Problem with OBJ Wrapping**

As shown in Figure 5-22, the AGB has a problem that prevents the upper part of the OBJ in the lower portion of the screen from displaying when the lower part of the OBJ was wrapped for display to the upper portion of the screen. This problem is corrected in NITRO (but the problem remains when NITRO is set to AGB-compatible mode).

**Figure 5-22 : Problem of OBJ Wrapping**



- [d15–d14] : OBJ Size

These bits set the OBJ size. The number of vertical and horizontal dots for an OBJ depends on this setting and the OBJ shape set in OBJ Attribute 0. Table 5-9 shows the relationship.

**Table 5-9 : OBJ Shape and OBJ Size Settings**

OBJ Attribute 1 OBJ size		OBJ Attribute 0 OBJ shape			
		00	01	10	11
00 (Square)		8x8	16x16	32x32	64x64
01 (Long rectangle)		16x8	32x8	32x16	64x32
10 (Tall rectangle)		8x16	8x32	16x32	32x64
11 (Prohibited setting)		Prohibited Setting			

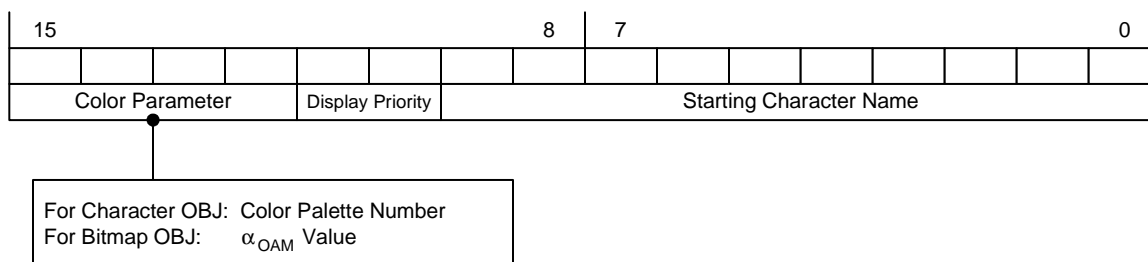
- [d13–d09] : Affine Transformation Parameter Selection

These bits specify which of the 32 sets of OAM PA – PD affine transformation parameters to reference.

When the OBJ Attribute 0 Affine Transformation Enable Flag is set to 0 (disabled), the [d13] VF bit is treated as the Vertical Flip Flag and the [d12] HF bit is treated as the Horizontal Flip Flag.

- [d08–d00] : x Coordinate

These bits specify the OBJ's x-coordinate in the display screen in the range of 0 to 511.

**OBJ Attribute 2**

- [d15–d12] : Color Parameter

What these bits specify depends on whether OBJ Mode in OBJ Attribute 0 is set to Character OBJ or Bitmap OBJ. These bits specify the Color Palette Number for Character OBJ. They specify the  $\alpha_{OAM}$  for Bitmap OBJ. The  $\alpha_{OAM}$  value is used as a factor for blending with the BG for the Bitmap OBJ (see "[5.3.4 Bitmap OBJ](#)" on page 128).

1. For Character OBJ: Color Palette Number

This specifies one of 16 palettes to apply to the character data.

This bit is invalid in 256-color mode when extended palettes are disabled (see "[OBJ Attribute 0](#)" on page 113).

Extended palettes are enabled/disabled with the DISPCNT [DB\_DISPCNT] register.

2. For Bitmap OBJ:  $\alpha_{OAM}$  value

The  $\alpha_{OAM}$  value is an element of the OBJ's transparency  $\alpha$ , where  $\alpha = \alpha_{BMP} \times (\alpha_{OAM} + 1)$ . Set  $\alpha_{BMP}$  using the Bitmap OBJ data (see "[5.3.4.1 Bitmap OBJ Data](#)" on page 130).

- [d11–d10] : Display Priority

These bits set the order of priority for display.

See "[5.9 Display Priority](#)" on page 151 to learn about the priority relation with BG.

The priority set with this bit is invalid when OBJ Mode in OAM Attribute 0 is set to OBJ Window. See "[5.6 Windows](#)" on page 142 to learn about the precedence of windows.

- [d09–d00] : Starting Character Name

The basic character number at the start of the OBJ character data mapped in OBJ-VRAM is written here. The specification in Bitmap OBJ mode is the same as for Character mode with 8x8-dot units.

### 2D Mapping Mode

When in 2D Mapping mode and 256-Color mode, the starting character name's lowest bit is fixed at 0. In addition, OBJ-VRAM references regions only up to 32 KB.

### 1D Mapping Mode

When in 1D Mapping mode, the capacity of OBJ-VRAM can be expanded (see the RAM Bank Control Registers 0 and 1 and the Display Control Register).

The boundary of the starting character name varies, as shown in Table 5-10 and Table 5-11, depending on the OBJ-VRAM capacity to allow the entire OBJ-VRAM region to be referenced with the setting region of the starting character name (10 bits).

**Table 5-10 : Character OBJ**

OBJ-VRAM Capacity	Starting Character Name Boundary
32 KB	32 bytes
64 KB	64 bytes
128 KB	128 bytes
256 KB	256 bytes

**Table 5-11 : Bitmap OBJ**

OBJ-VRAM Capacity	Starting Character Name Boundary
128 KB	128 bytes
256 KB	256 bytes

**Note:** The maximum capacity of OBJ-VRAM is 128 KB for 2D Graphics Engine B because of restrictions on VRAM allocation. Therefore, the Character OBJ and Bitmap OBJ capacity cannot be set to 256 KB.

**Affine Transformation Parameters**

See "[5.3.2.3 OBJ Rotation and Scaling \(Affine Transformation\)](#)" on page 120 for how to determine the OBJ's affine transformation parameters.

**Affine Transformation Parameter PA**

15	14	8	7	0
S_PA	INTEGER_PA		DECIMAL_PA	
Distance dx moved in x-direction on the same line				

Signed fixed-point decimal (sign + 7-bit integer + 8-bit decimal part)

**Affine Transformation Parameter PB**

15	14	8	7	0
S_PB	INTEGER_PB		DECIMAL_PB	
Distance dmx moved in x-direction on the next line				

Signed fixed-point decimal (sign + 7-bit integer + 8-bit decimal part)

**Affine Transformation Parameter PC**

15	14	8	7	0
S_PC	INTEGER_PC		DECIMAL_PC	
Distance dy moved in y-direction on the same line				

Signed fixed-point decimal (sign + 7-bit integer + 8-bit decimal part)

**Affine Transformation Parameter PD**

15	14	8	7	0
S_PD	INTEGER_PD		DECIMAL_PD	
Distance dmy moved in y-direction on the next line				

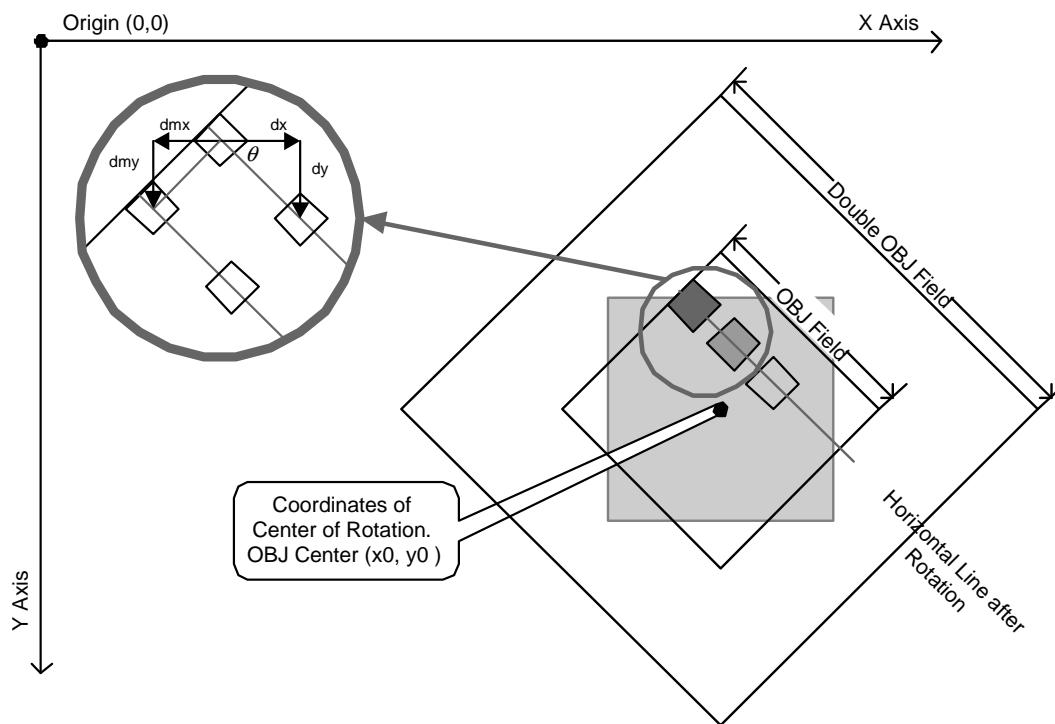
Signed fixed-point decimal (sign + 7-bit integer + 8-bit decimal part)

### 5.3.2.3 OBJ Rotation and Scaling (Affine Transformation)

The OBJ character data is referenced horizontally in sequence from the top left when an OBJ is displayed, so a rotated OBJ can be displayed by rotating the reference direction. The center of the rotation is fixed to the center of the OBJ field (dot boundary). If a reference point is outside the specified OBJ size, it becomes transparent.

Figure 5-23 shows the rotation and scaling process for an OBJ.

**Figure 5-23 : OBJ Rotation and Scaling**



$$dx \text{ (reference distance in x-direction for same line)} = (1/\alpha)\cos\theta$$

$$dy \text{ (reference distance in y-direction for same line)} = -(1/\beta)\sin\theta$$

$$dmx \text{ (reference distance in x-direction for next line)} = (1/\alpha)\sin\theta$$

$$dmy \text{ (reference distance in y-direction for next line)} = (1/\beta)\cos\theta$$

**Note:**  $\alpha$  is the scale ratio along the x-axis;  $\beta$  is the scale ratio along the y-axis.

- **OBJ Rotation and Scaling Process**

1. Affine transformation parameter numbers to be applied are specified in OBJ Attribute 1 registered in OAM. In addition, the affine transformation parameters PA, PB, PC, and PD to be applied are set in OAM using the information in Figure 5-23.
2. The image-processing circuitry calculates the coordinates in the x-direction in relation to the data reference start point that uses the center of the OBJ field as the center of rotation by summing the cumulative increase in the x-direction (dx and dy).
3. If the line advances, the rendering start point coordinates for the next line are calculated by summing the cumulative increase in the y-direction (dmx and dmy) in relation to the reference starting point. Then the process in Step 2 is performed.







5.3.3.1 Character Data Format

The character data format for Character OBJ is shown below. The Character Display table shows the case when an 8x8-dot character is defined.

5.3.3.1.1 16-Color Mode

The character data format for 16-color mode, correspondence between character display and pixel data, and address mapping (Figure 5-24) are shown below.

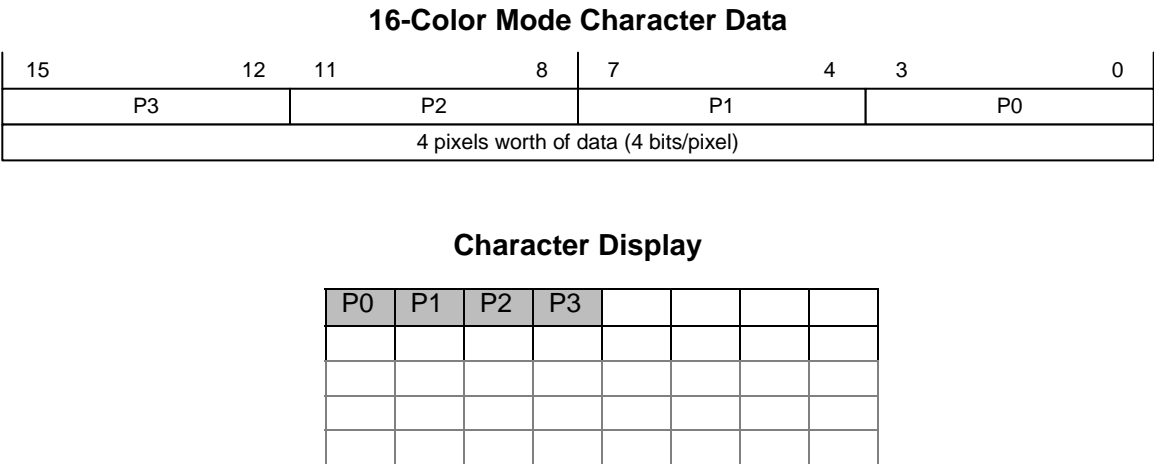
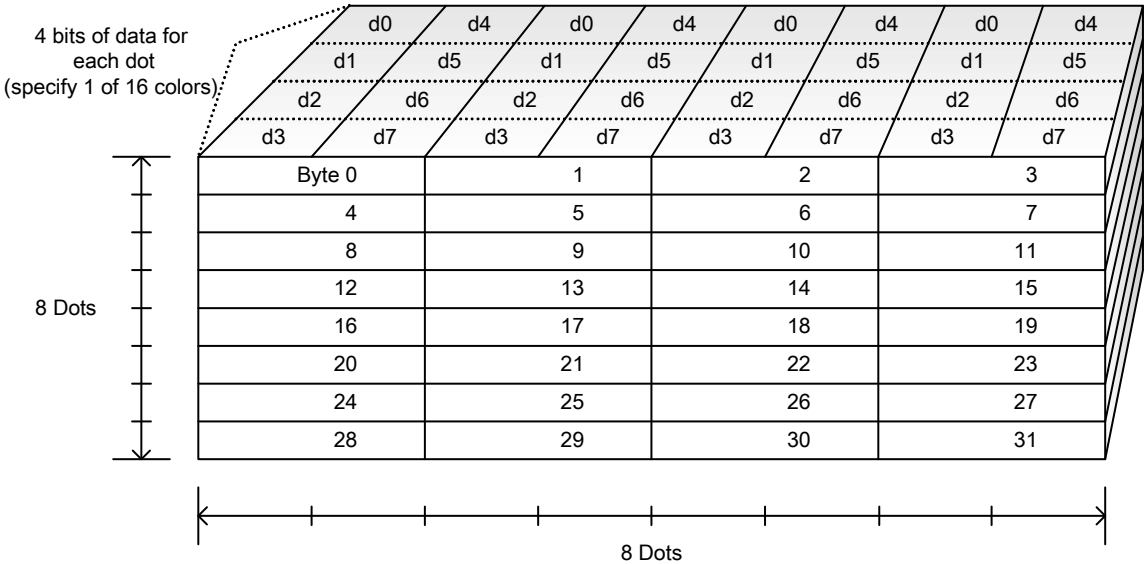


Figure 5-24 : Character Data Address Mapping (16-Color Mode Character OBJ)



5.3.3.1.2 256-Color Mode

The character data format for 256-color mode, correspondence between character display and pixel data, and address mapping (Figure 5-25) are shown below.

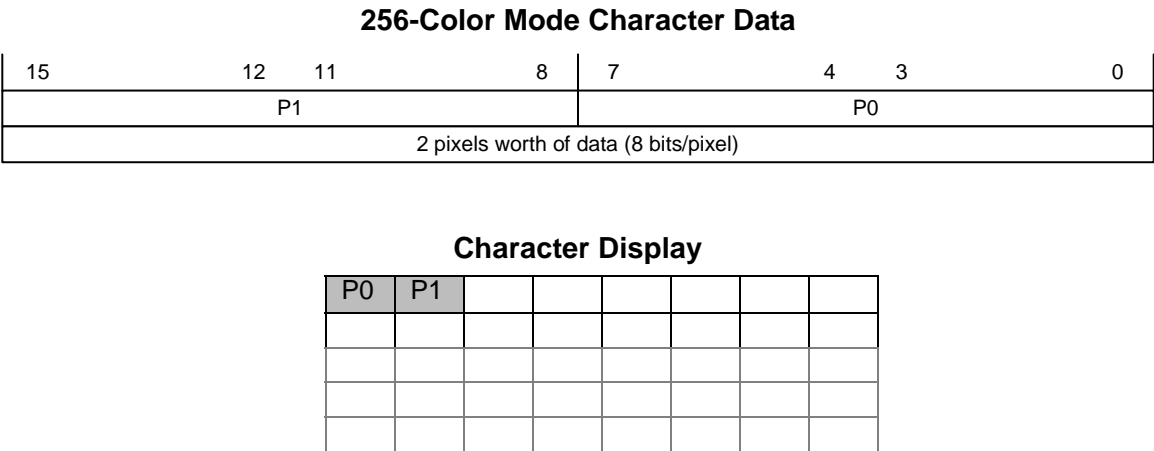
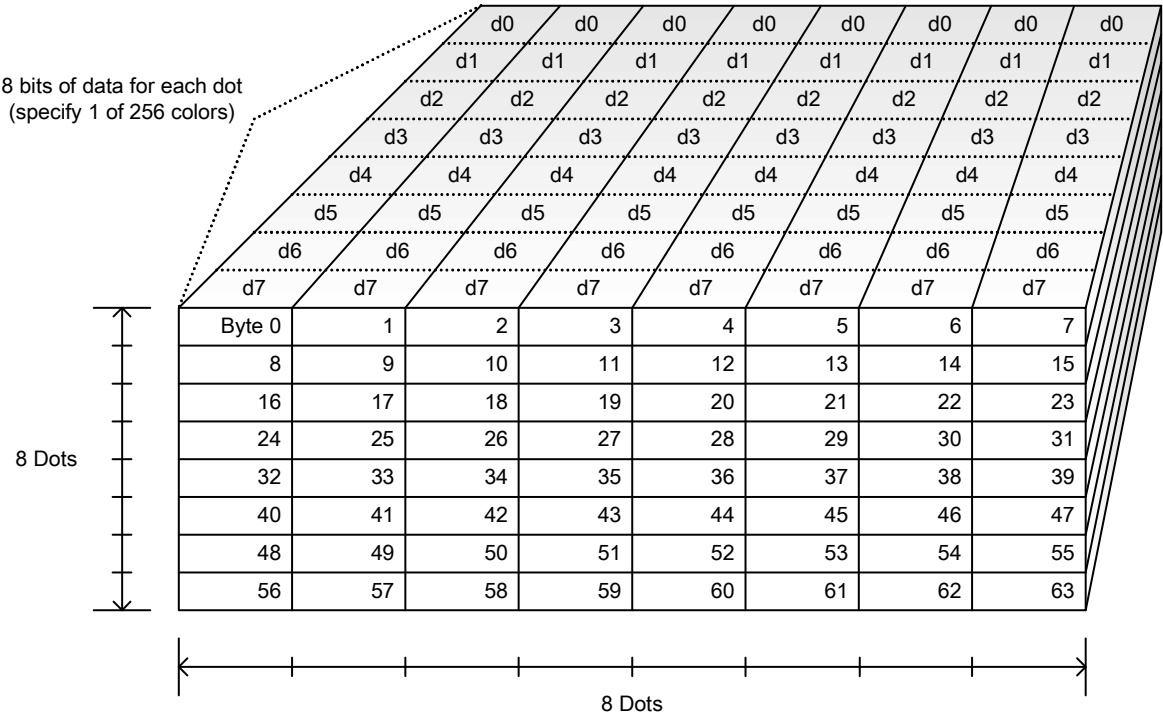


Figure 5-25 : Character Data Address Mapping (256-Color Mode Character OBJ)

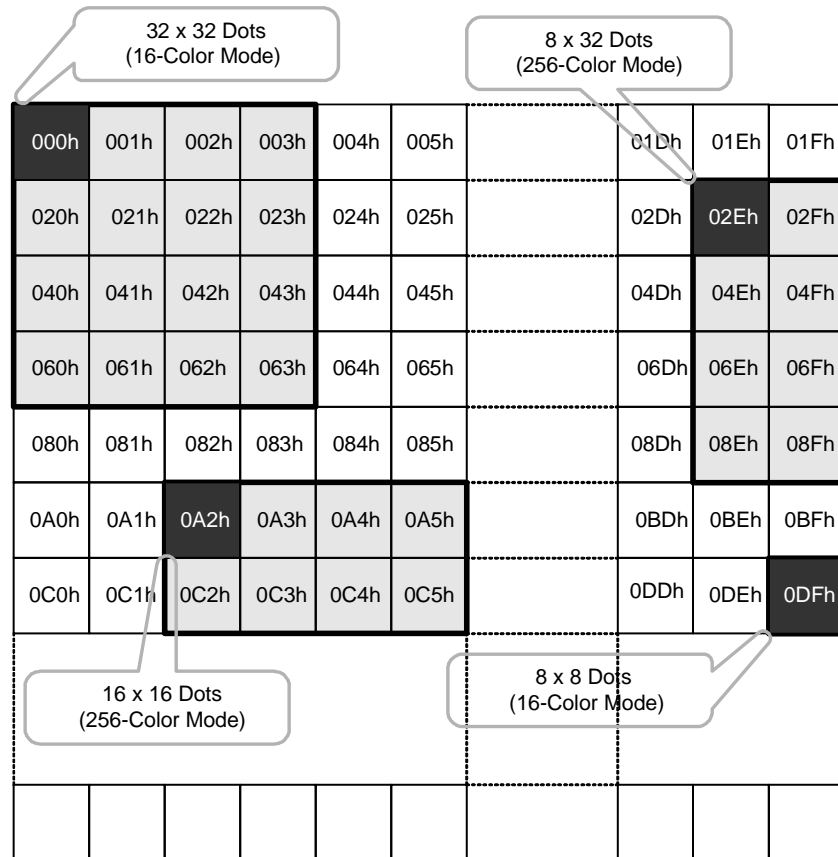


### 5.3.3.2 Mapping Modes for Character OBJ Data

#### 5.3.3.2.1 2D Mapping

When displaying 256-color x 1-palette characters in 2D mapping mode, the character name specification is limited to even numbers, as shown in Figure 5-26.

**Figure 5-26 : 2D Mapping**



### 5.3.3.2.2 1D Mapping

The address where the data that makes up the character is stored is consecutive for each character, as shown in Figure 5-27 and Figure 5-28.

**Figure 5-27 : 1D Mapping when Character Name Boundary is 32 Bytes**

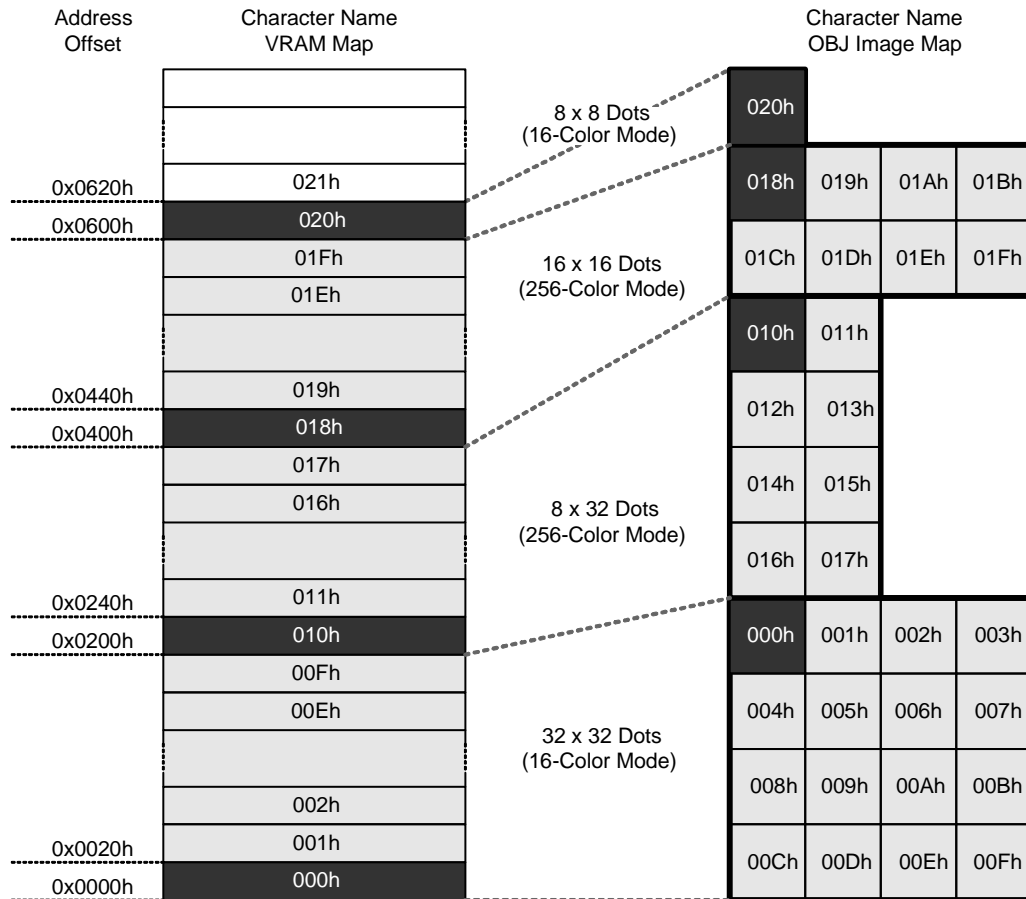
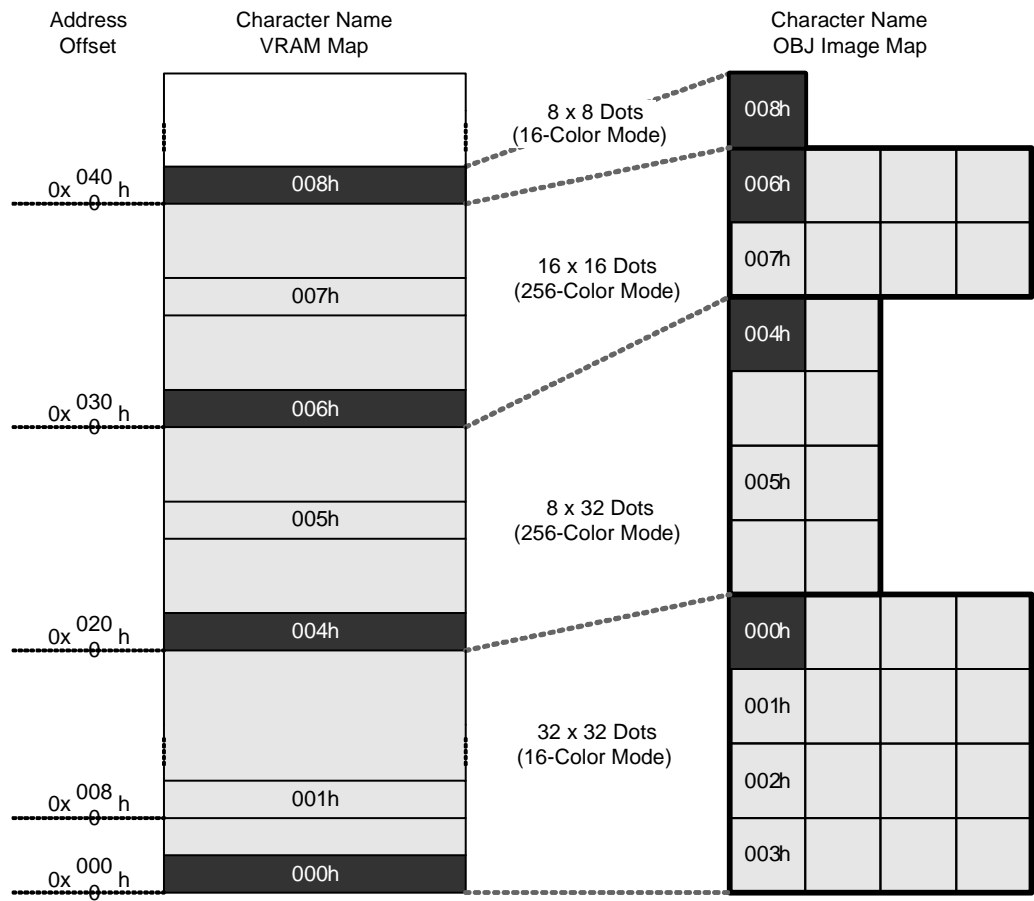
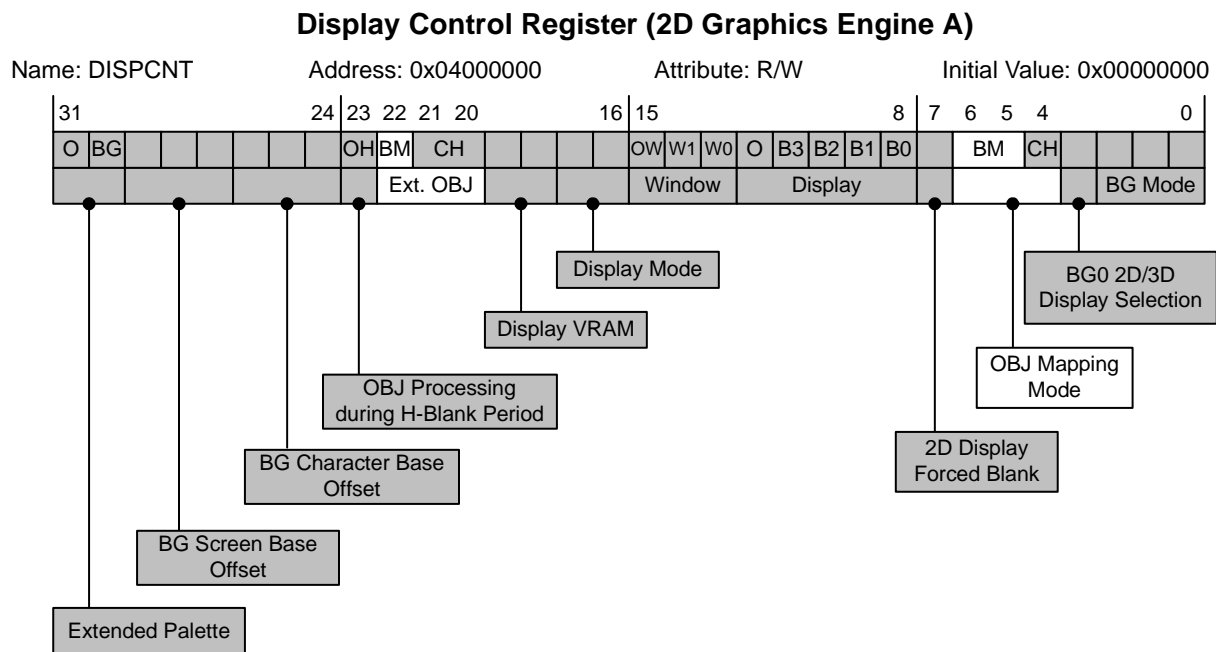


Figure 5-28 : 1D Mapping when Character Name Boundary is 128 Bytes



### 5.3.4 Bitmap OBJ

The VRAM Extended Flag and Mapping Mode for Bitmap OBJ are set with the DISPCNT [DB\_DISPCNT] Register.



- [d22–d20] : OBJ-VRAM Region Extended Flag
  - BM [d22] : VRAM Extended Flag for Bitmap OBJ

These bits specify OBJ-VRAM capacity when 1D mapping is selected for OBJ bitmap data.

0	128 KB (starting character name boundary of 128 bytes)
1	256 KB (starting character name boundary of 256 bytes)

- [d06–d04] : OBJ Data Mapping Mode
  - BM [d06–d05] : Bitmap OBJ Data Mapping Mode

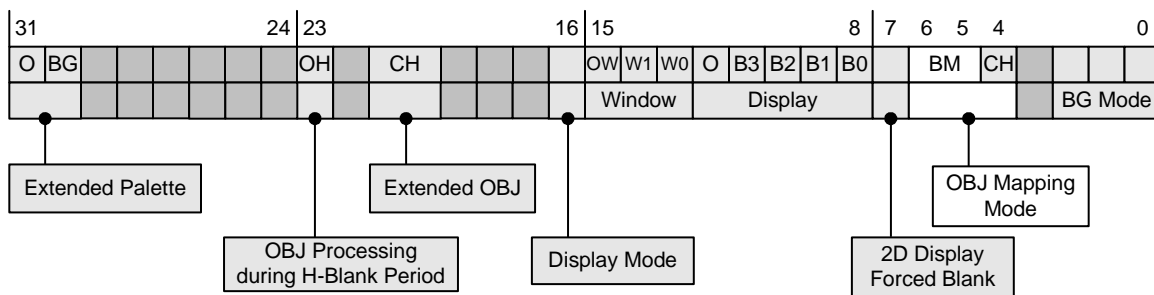
00	2D mapping with 128 horizontal dots
01	2D mapping with 256 horizontal dots
10	1D mapping
11	Prohibited Setting

The OBJ-VRAM Region Extended Flag is ignored with 2D mapping. In this case, OBJ-VRAM is referenced in a range of addresses that can be specified by the 10-bit character name set by OBJ Attribute 2 in OAM.

Capacity is set with the OBJ-VRAM Region Extended Flag with 1D mapping.

**DB\_DISPCNT: Display Control Register 1 (2D Graphics Engine B)**

Name: DB\_DISPCNT      Address: 0x04001000      Attribute: R/W      Initial Value: 0x00000000



- [d06–d04] : OBJ Data Mapping Mode
  - BM [d06–d05] : Bitmap OBJ Data Mapping Mode

<b>00</b>	2D mapping with 128 horizontal dots
<b>01</b>	2D mapping with 256 horizontal dots
<b>10</b>	1D mapping
<b>11</b>	Prohibited Setting

In 2D mapping mode, only up to 32 KB of OBJ-VRAM can be referenced.

In 1D mapping mode, OBJ-VRAM capacity is set to 128 KB.

**Note:** With 2D Graphics Engine B, a maximum of 128 KB can be allocated to VRAM. Accordingly, although it can be specified for 2D Graphics Engine A, the OBJ-VRAM capacity for 1D mapping mode is fixed to 128 KB.

### 5.3.4.1 Bitmap OBJ Data

**Bitmap OBJ data**

15	14	10	9	8	7	5	4	0
A	BLUE			GREEN			RED	
$\alpha_{BMP}$	P0							

- A [d15] :  $\alpha_{BMP}$

The  $\alpha_{BMP}$  value is an element of the OBJ's transparency  $\alpha$ , where  $\alpha = \alpha_{BMP} \times (\alpha_{OAM} + 1)$ . Set  $\alpha_{OAM}$  using OBJ Attribute 2 (see "[5.3.2.2 OAM Data Format](#)" on page 113).

**OBJ Display**

P0							

### 5.3.4.2 Blending with BG

As with translucent OBJ, Bitmap OBJ can be blended with the BG of the second target screen for display. When  $\alpha_{OAM} = 0$ , the entire region of the OBJ becomes transparent and is not rendered. When  $\alpha_{OAM}$  is non-zero, the OBJ is blended for display according to the following formula:

$$C = \frac{C_{OBJ} \times \alpha + C_{BG} \times (16 - \alpha)}{16}$$

$$\alpha = \alpha_{BMP} \times (\alpha_{OAM} + 1)$$

$\alpha_{BMP}$  is set with Bitmap OBJ data, and  $\alpha_{OAM}$  is a value specified with OBJ Attribute 2 of OAM.

C is the color of the blending result (calculation results are rounded to the nearest integer).

$C_{OBJ}$  is the Bitmap OBJ color of the first target screen.

$C_{BG}$  is the BG color of the second target screen.



### 5.3.4.3 Mapping Modes for Bitmap OBJ Data

#### 5.3.4.3.1 2D Mapping with 128 Horizontal Dots

Figure 5-29 shows the 2D map of Bitmap OBJ data with 128 horizontal dots in VRAM.

**Figure 5-29 : 2D Map of Bitmap OBJ Data VRAM (128 Horizontal Dots)**

	Dot 0	1	2	3		125	126	127
Line 0	0h	2h	4h	6h		FAh	FCh	FEh
1	100h	102h	104h	106h		1FAh	1FCh	1FEh
2	200h	202h					2FCh	2FEh
3	300h	302h					3FCh	3FEh
4	400h							4FEh

Character names are set in units of 8x8 dots (128 bytes) of bitmap data. Figure 5-30 shows the 2D image map of character names in VRAM.

**Figure 5-30 : 2D Image Map of Character Name VRAM**

Line \ Dot	0-7	8-15	16-23	24-31		104-111	112-119	120-127
0-7	0h	1h	2h	3h		Dh	Eh	Fh
8-15	10h	11h	12h	13h		1Dh	1Eh	1Fh
16-23	20h	21h	22h	23h		2Dh	2Eh	2Fh
24-31	30h	31h	32h	33h		3Dh	3Eh	3Fh
32-39	40h	41h	42h	43h		4Dh	4Eh	4Fh

### 5.3.4.3.2 2D Mapping with 256 Horizontal Dots

Figure 5-31 shows the 2D map of Bitmap OBJ data with 256 horizontal dots in VRAM.

**Figure 5-31 : 2D Map of Bitmap OBJ Data VRAM (256 Horizontal Dots)**

	Dot 0	1	2	3		253	254	255
Line 0	0h	2h	4h	6h		1FAh	1FCh	1FEh
1	200h	202h	204h	206h		3FAh	3FCh	3FEh
2	400h	402h					5FCh	5FEh
3	600h	602h					7FCh	7FEh
4	800h							9FEh

Character names are set in units of 8x8 dots (128 bytes) of bitmap data. Figure 5-32 shows the 2D image map of character names in VRAM.

**Figure 5-32 : 2D Image Map of Character Name VRAM**

Line \ Dot	0-7	8-15	16-23	24-31		232-239	240-247	248-255
0-7	0h	1h	2h	3h		1Dh	1Eh	1Fh
8-15	20h	21h	22h	23h		3Dh	3Eh	3Fh
16-23	40h	41h	42h	43h		5Dh	5Eh	5Fh
24-31	60h	61h	62h	63h		7Dh	7Eh	7Fh
32-39	80h	81h	82h	83h		9Dh	9Eh	9Fh

### 5.3.4.3.3 1D Mapping

#### Character Names

In 2D Graphics Engine A, the Bitmap OBJ VRAM extension flag in the display control register DISPCNT changes both the OBJ-VRAM range that can be specified by a character name, and the character name boundary (see Table 5-14).

In 2D Graphics Engine B, VRAM allocation is restricted, so the OBJ-VRAM range is fixed to 128 KB, and the character name boundary is fixed to 128.

**Table 5-14 : Character Name Boundaries**

Bitmap OBJ VRAM Extended Flag	OBJ-VRAM Specifiable Range	Character Name Boundary
0	128 KB	128 bytes
1	256 KB	256 bytes

For example, if the VRAM Extended Flag for Bitmap OBJ is set to 0, and the OBJ Attribute 2 setting for Starting Character Name is set to 4Ch, the Bitmap OBJ data defined from address 2600h (= 4Ch x 128 bytes) is referenced.

#### 1D VRAM Mapping of Bitmap OBJ Data

Map bitmap data from the starting address of the character name boundary for the size of the character. This size is not in units of 8x8 dots. Figure 5-33 and Figure 5-34 show the 1D map for 8x8-dot characters and 16x16-dot characters. In these figures, C+xxh denotes the offset from the starting address of the character name boundary.

**Figure 5-33 : 1D Map of VRAM with 8x8-Dot Characters**

	0	1	2	3	4	5	6	7
0	C+0h	C+2h	C+4h	C+6h	C+8h	C+Ah	C+Ch	C+Eh
1	C+10h	C+12h	C+14h	C+16h	C+18h	C+1Ah	C+1Ch	C+1Eh
2	C+20h							C+2Eh
3	C+30h							C+3Eh
4	C+40h							C+4Eh
5	C+50h							C+5Eh
6	C+60h	C+62h	C+64h	C+66h	C+68h	C+6Ah	C+6Ch	C+6Eh
7	C+70h	C+72h	C+74h	C+76h	C+78h	C+7Ah	C+7Ch	C+7Eh

**Figure 5-34 : 1D Map of VRAM with 16x16-Dot Characters**

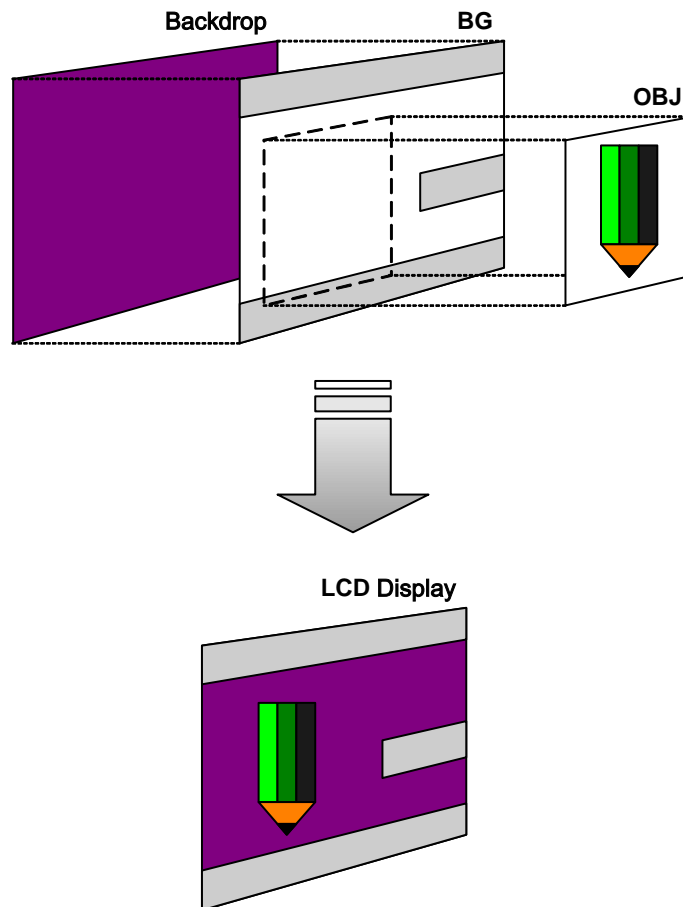
	0	1	2		13	14	15
0	C+0h	C+2h	C+4h		C+1Ah	C+1Ch	C+1Eh
1	C+20h	C+22h	C+24h		C+3Ah	C+3Ch	C+3Eh
2	C+40h						C+5Eh
13	C+1A0h						C+1BEh
14	C+1C0h	C+1C2h	C+1C4h		C+1DAh	C+1DCh	C+1DEh
15	C+1E0h	C+1E2h	C+1E4h		C+1FAh	C+1FCh	C+1FEh

## 5.4 Backdrop

The 2D graphics displayed on the LCD are composed of OBJ, BG, and the Backdrop. An OBJ is a relatively small image, but several of them can be displayed. They are mainly used to display characters that move around the screen. A BG has features equivalent to an OBJ, but only a few BG screens can be displayed because a BG is large and consumes a lot of memory. A BG is used to display large images such as objects that are continuously on-screen or in the background.

On NITRO, regions of the LCD screen where no OBJ and BG are displayed are filled with a single color. This region is called the *Backdrop* and can be visualized as a single-color surface that is always displayed furthest in the back, as depicted in Figure 5-35. The Backdrop is a surface filled only with a single color and does not have the features of OBJ and BG. The Backdrop color can be changed with the palette (see "[5.5 Color Palettes](#)" on page 136).

**Figure 5-35 : Backdrop Schematic**



5.5 Color Palettes

As a standard feature, NITRO has RAM allocated specifically for BG and OBJ palettes (Palette RAM). Data stored in Palette RAM are called standard palettes.

A BG or OBJ can be displayed using just a standard palette, but extended palettes allow the use of 256 colors x 16 palettes and enable richer visuals. To use extended palettes, allocate VRAM using the RAM Control Register and enable the Extended Palette Flag with the DISPCNT [DB\_DISPCNT] Register.

5.5.1 Standard Palettes

Standard palette RAM is allocated separately for OBJ and for BG in both 2D Graphics Engine A and 2D Graphics Engine B. Color 0 in each palette is the transparent color, regardless of the settings. The Backdrop screen uses the color set at the beginning of the BG palette (Color 0 of Palette 0). Because standard palette RAM resides inside the 2D Graphic Engines, the 2D Graphic Engine must be enabled in the Power Control Register (POWCNT) before data can be written to its RAM.

Figure 5-36 shows the standard palette RAM addresses. Figure 5-37 shows the color specifications for 16 Colors x 16 Palettes. Figure 5-38 shows the color specifications for 256 Colors x 1 Palette.

Figure 5-36 : Standard Palette RAM Addresses (Add 0x400h for 2D Graphics Engine B)

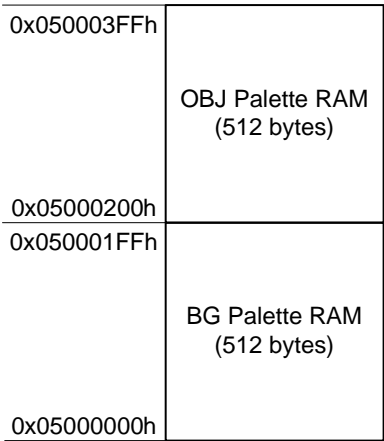


Figure 5-37 : 16 Colors x 16 Palettes

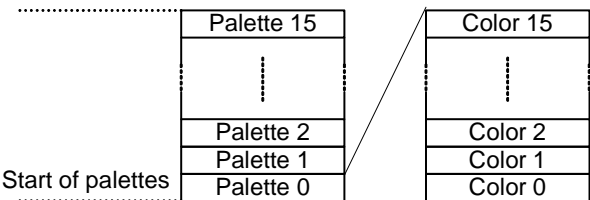
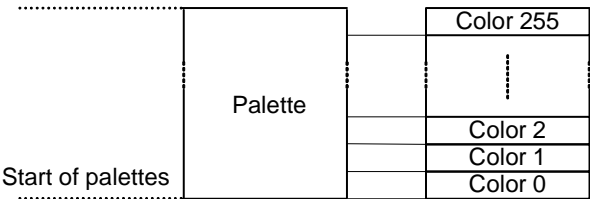


Figure 5-38 : 256 Colors x 1 Palette





- [d31,d30] : Extended Palette Enable Flag
  - BG [d30] : BG Extended Palette

This flag is valid for BG screens that can be displayed with 256 Colors x 16 Palettes.

<b>0</b>	Disable (256 colors x 1 palette)
<b>1</b>	Enable (256 colors x 16 palettes)

The standard palette is always used for BG screens that do not support 256 Colors x 16 Palettes, even if BG Extended Palettes are enabled. In addition, the Backdrop screen always uses Color 0 of the standard palette.

To use BG extended palettes, VRAM must be allocated to the BG Extended Palette Slots. See the RAM Bank Control Register for allocating VRAM to the BG Extended Palette Slots.

### BG Extended Palette Slots

BG Extended Palettes can have up to 32 KB allocated to Slots 0-3. Whether BG0 uses Slot 0 or Slot 2 is selected with the BG0 Control Register, and whether BG1 uses Slot 1 or Slot 3 is selected with the BG1 Control Register. BG2 can use only Slot 2, and BG3 can use only Slot 3. Therefore, if Slot 0 is set to BG0 and Slot 1 to BG1, each BG screen can use its own extended palette. On the other hand, by setting Slots 2 and 3 to be shared by all BG screens, the BG Extended Palettes can conserve 16 KB.

Color 0 in both palettes is the transparent color, regardless of the settings. The Backdrop screen uses the color set at the beginning of the BG standard palette (Color 0 of Palette 0).

Figure 5-39 shows the memory map for BG extended palette slots.

**Figure 5-39 : BG Extended Palette Memory Map**

0x00008000	
0x00006000	Slot 3
0x00004000	Slot 2
0x00002000	Slot 1
0x00000000	Slot 0



Table 5-15 lists the palettes that can be used by each type of BG.

**Table 5-15 : Palettes and BG Types**

Category	BG Type		Colors/ Palette	BG Screen	Usable Palette Region				
					Standard	Extended Palette Slot			
						0	1	2	3
Character BG	Text BG		16/16	BG0-3	X				
			256/16	BG0	X	X		X	
				BG1	X		X		X
				BG2	X			X	
				BG3	X				X
	Affine BG		256/1	BG2-3	X				
	Ext. BG	256-Color x 16- Palette BG	256/16	BG2	X			X	
BG3				X				X	
Bitmap BG	Ext. BG	256-Color	256/1	BG2-3	X				
		Direct Color	32,768	BG2-3					
	Large-Screen 256-Color		256/1	BG2	X				

As shown in "[Table 5-15 : Palettes and BG Types](#)" on page 139, the Extended Palette Slot number can be selected for BG0 and for BG1. With 2D Graphics Engine B, Large-Screen 256-Color Bitmap BG cannot be selected for the BG type. See the section on the BG Control Register in "[5.2.2 BG Control](#)" on page 81 to learn how to select slots.



**OBJ Extended Palette Slots**

Although 16 KB of VRAM is allocated to the OBJ extended palettes, only 8 KB of this can be used as an extended palette. As Figure 5-40 illustrates, only Slot 0 can be used as an extended palette; Slot 1 is invalid.

Color 0 for each palette is handled as the transparent color, regardless of the settings. The Backdrop screen uses the color set at the beginning of the standard BG palette (Color 0 of Palette 0).

**Figure 5-40 : OBJ Extended Palette Slot Memory Map**

0x00004000	
0x00002000	Slot 1 (Invalid Region)
0x00000000	Slot 0

## 5.6 Windows

Window features can restrict the regions where BG and OBJ screens are displayed as well as the region where color special effects are applied. NITRO uses three kinds of windows: Window 0, Window 1, and the OBJ Window (see OBJ Attribute 0 in "[5.3 OBJ](#)" on page 109 to read about the OBJ Window settings).

### Window Interior Control Register

Name	Address	Attribute	Initial Value
(2D_A) WININ	0x04000048	R/W	0x0000
(2D_B) DB_WININ	0x04001048	R/W	0x0000

15	13	12	8				7	5	4	0					
		EFCT	OBJ	BG3	BG2	BG1	BG0			EFCT	OBJ	BG3	BG2	BG1	BG0
Inside Window 1								Inside Window 0							

### Window Exterior and OBJ Window Interior Control Register

Name	Address	Attribute	Initial Value
(2D_A) WINOUT	0x0400004A	R/W	0x0000
(2D_B) DB_WINOUT	0x0400104A	R/W	0x0000

15	13		12		8			7	5		4		0		
		EFCT	OBJ	BG3	BG2	BG1	BG0			EFCT	OBJ	BG3	BG2	BG1	BG0
Inside OBJ Window								Outside Window (0,1, and OBJ Windows)							

- [d13], [d05] : EFCT: Color Special Effects Enable Flag

0	Disable
1	Enable

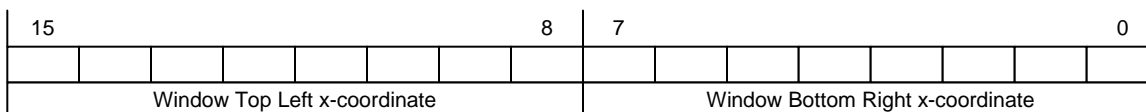
- OBJ, BG3-0 [d12–d08], [d04–d00] : Display Enable Flag

0	Hide
1	Show

Display control over the window's exterior is valid whenever Window 0, Window 1, or the OBJ Window is displayed.

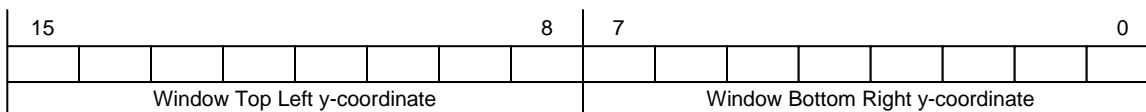
## Window Position Setting Register

Name	Address	Attribute	Initial Value
(2D_A) WINxH(x=0, 1)	0x04000040, 0x04000042	W	0x0000
(2D_B) DB_WINxH(x=0, 1)	0x04001040, 0x04001042	W	0x0000



- [d15–d08] : Window Top-Left x-coordinate  
Set this in the range between 0 and 255.
- [d07–d00] : Window Bottom-Right x-coordinate  
Set this in the range between 0 and 255.

Name	Address	Attribute	Initial Value
(2D_A) WINxV (x=0, 1)	0x04000044, 0x04000046	W	0x0000
(2D_B) DB_WINxV (x=0, 1)	0x04001044, 0x04001046	W	0x0000



- [d15–d08] : Window Top-Left y-coordinate  
Set this in the range between 0 and 191.
- [d07–d00] : Window Bottom-Right y-coordinate  
Set this in the range between 0 and 192.

### Window Range

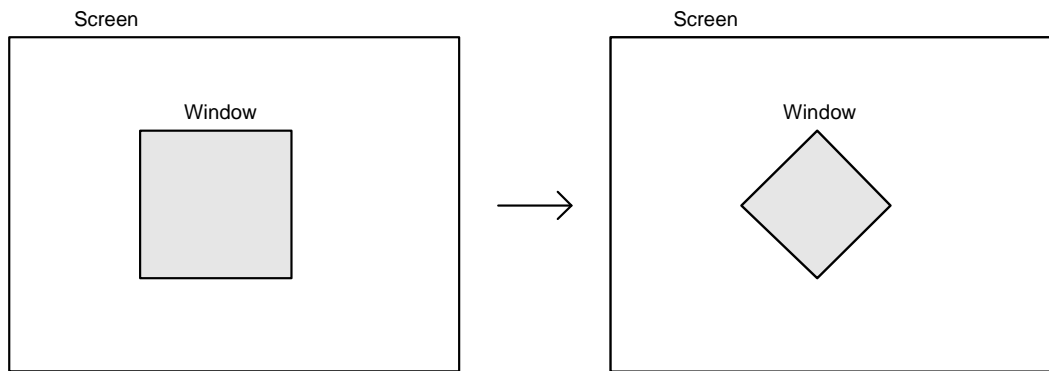
If the window's top-left coordinates are (lx, ly) and the bottom-right coordinates are (rx, ry), the window range for LCD coordinates (0, 0) – (255, 191) is (lx, ly) – (rx-1, ry-1).

To locate a window along the right side of the LCD screen, set its bottom-right x-coordinate to 0. To locate a window along the left side of the LCD screen, set the top-left x-coordinate to 0. However, if both x-coordinates are set to 0, the window is not displayed. Consequently, the window cannot span the entire LCD screen width. Use another window (or the OBJ Window) to span the entire width of the LCD screen with windows.

### Window Shape

Window 0 and Window 1 can be set only as rectangular shapes. However, the shape's appearance can be altered by overwriting the Window Position Setting Register during an H-Blank period (see Figure 5-41).

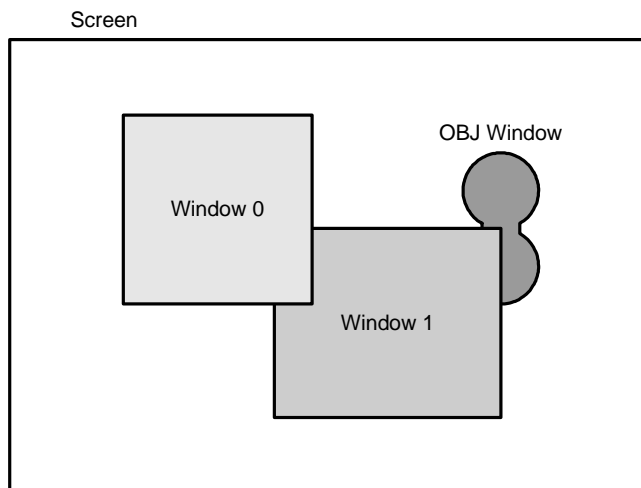
**Figure 5-41 : Altering a Window Shape**



### 5.6.1 Precedence of Windows

As shown in Figure 5-42, Window 0 always has display priority (precedence) over Window 1, and the OBJ Window has the lowest precedence. The precedence cannot be changed.

**Figure 5-42 : Display Priority of Window 0, Window 1, and the OBJ Window**



**Note 1 Regarding Windows:** When the top-left y-coordinate of the window is between 0 and 6, the top-left y-coordinate is forcibly displayed as though it were 0. Use one of the following two methods to work around this problem.

**Method 1: Perform the following steps:**

1. Set the window's y-coordinate to a value of 7 or higher before the V-Count reaches 256 during the V-Blank process.
2. Restore the window's y-coordinate to its original value after confirming that V-Count has reached 262 by checking V-Count using a V-Count Match Interrupt or an H-Blank Interrupt.

**Method 2: Perform the following steps:**

1. Set the Window Display Enable Flag in the Display Control Register, using the values shown below after confirming that V-Count has reached 262 by checking V-Count using a V-Count Match Interrupt or an H-Blank Interrupt.
 

When the window y-coordinate is 0:	1 (Show window)
When the window y-coordinate is non-zero:	0 (Hide window)
2. Read the V-Count value using an H-Blank Interrupt, and compare that value to (window y-coordinate – 1). If the two values are equal, set the Window Display Enable Flag in the Display Control Register to 1 (Show window) during the H-Blank.

**Note 2 Regarding Windows:** Immediately after drawing a line in which the bottom-right x coordinate is set to 0, drawing of the current window will not be complete at H-Blank. If the coordinates of the next window are set in this state, the window will continue drawing up to the bottom-right x coordinate of the next window.

To avoid this, you must wait until the window has finished drawing before setting the coordinates of the next window. If the bottom-right x coordinate is 0, the window will finish drawing 3 clock cycles of the system clock (33 MHz) after the H-Blank flag changes from 1 to 0 (or the V-Count value increments by 1). Between this time and the time when the H-Count reaches the top-left x coordinate of the next window, it will be outside the window. So set the coordinates for the next window during this period. Note that you cannot use this method when setting window coordinates using H-Blank-initiated DMA.

## 5.7 Color Special Effects

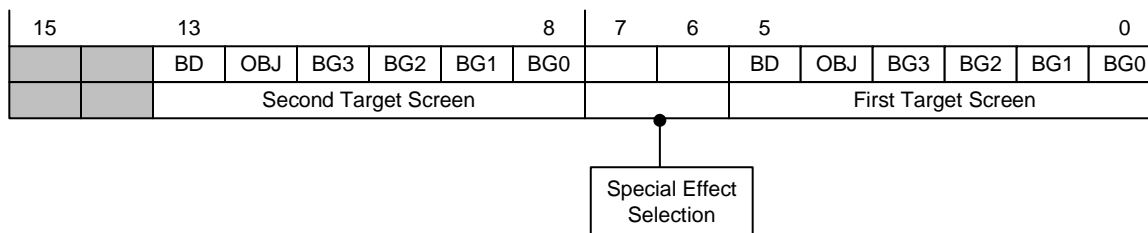
OBJ and BG can use the alpha-blending and fade-in/fade-out color special effects. These effects can be limited to a region by using windows (see "[5.6 Windows](#)" on page 142). Table 5-16 summarizes color special effects.

**Table 5-16 : Color Special Effects**

Color Special Effect	Result
<b>Alpha-Blending</b>	Computations are conducted and a 16-level translucency process is performed on two selected screens. This process is not performed on transparent portions (transparent pixels).
<b>Brightness Up/Down (Fade-in/Fade-out)</b>	Computations are conducted and a 16-level process of changing the brightness is performed on the selected screen. This process is not performed on transparent portions (transparent pixels).

### Color Special Effect Control Register

Name	Address	Attribute	Initial Value
(2D_A) BLDCNT	0x04000050	R/W	0x0000
(2D_B) DB_BLDCNT	0x04001050	R/W	0x0000



Color special effects are set with the BLDCNT [DB\_BLDCNT] Register. For alpha-blending, which processes two screens, the two target screens must have the proper priority. In addition, translucent OBJ are specified separately in OAM and the BLDCNT [DB\_BLDCNT] Register specifies color special effects for the entire OBJ.



- [d07–d06] : Special Effect Selection

**Table 5-17 : Color Special Effects and Processing**

Effect Selection		Type	Description of the Color Special Effect Processing
d07	d06		
0	0	None	Normally, color special effect processing is not performed. However, 16-level translucency processing (alpha-blending) is performed if a second target screen is directly behind a translucent OBJ, Bitmap OBJ, or a 3D surface is rendered to a BG0 screen.
0	1	Alpha-Blending	16-level translucency processing (alpha-blending) is performed if a second target screen is directly behind the first target screen. Set the first target screen's Backdrop screen bit to off ([d05] = 0). If OBJ = 1 for the first target screen, processing is executed on all OBJ, regardless of type. If OBJ = 0, processing is executed only for translucent OBJ, Bitmap OBJ, or a 3D surface rendered to a BG0 screen.
1	0	Brightness Up (see note)	Gradually increases the brightness of the first target screen. If the first target screen specifies OBJ = 1, processing is executed only for normal OBJ. alpha-blending is always performed when a second target screen is directly behind a translucent OBJ, Bitmap OBJ, or 3D surface rendered to a BG0 screen.
1	1	Brightness Down (see note)	Gradually decreases the brightness of the first target screen. If the first target screen specifies OBJ = 1, processing is executed only for normal OBJ. alpha-blending is always performed when a second target screen is directly behind a translucent OBJ, Bitmap OBJ, or 3D surface rendered to a BG0 screen.

**Note:** As stated in Table 5-17, alpha-blending is always performed on translucent OBJ, Bitmap OBJ, and BG0 rendered with a 3D surface and a second target screen, where the second target screen is directly behind, regardless of the Special Effect Selection setting. Therefore, to use Fade-in/Fade-out with these screens, do not specify a second target screen (clear all), or place something other than a second target screen immediately behind these screens.

## 1. Alpha-Blending

### Color Special Effect Alpha-Blending Factor Register

Name	Address	Attribute	Initial Value
(2D_A) BLDALPHA	0x04000052	R/W	0x0000
(2D_B) DB_BLDALPHA	0x04001052	R/W	0x0000



The factors used for alpha-blending are set with EVA and EVB in the BLDALPHA [DB\_BLDALPHA] Register. EVA and EVB are divided by 16 and are used as the pixel color factors in the equations below (when EVA or EVB exceeds 16, it is reset to 16).

Note that when a Bitmap OBJ is blended with the second target screen, the Bitmap OBJ's alpha value is used instead of these values. For further details, see "[5.3.4 Bitmap OBJ](#)" on page 128 and "[5.3.4.2 Blending with BG](#)" on page 130.

- Computations for alpha-blending (16 levels of translucency)

$$\text{Display color (R)} = \text{1st pixel color (R)} \times (\text{EVA} / 16) + \text{2nd pixel color (R)} \times (\text{EVB} / 16)$$

$$\text{Display color (G)} = \text{1st pixel color (G)} \times (\text{EVA} / 16) + \text{2nd pixel color (G)} \times (\text{EVB} / 16)$$

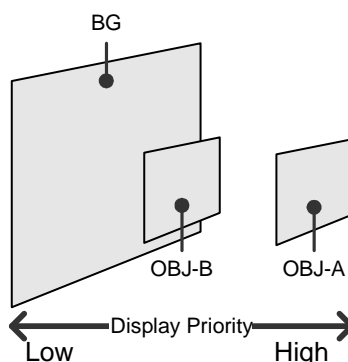
$$\text{Display color (B)} = \text{1st pixel color (B)} \times (\text{EVA} / 16) + \text{2nd pixel color (B)} \times (\text{EVB} / 16)$$

The computation results for alpha-blending are rounded to the nearest integer.

**Note:** An OBJ cannot be alpha-blended with another OBJ.

Figure 5-43 shows a case where an OBJ is specified as the first target screen, and a BG and an OBJ are specified as the second target screen. In this situation, OBJ-B is ignored as the target pixels for alpha-blending, and alpha-blending is carried out with OBJ-A and BG, just as if the BG were located directly behind OBJ-A.

**Figure 5-43 : Alpha-Blending Display Priority**



## 2. Brightness Up/Down

## Color Special Effect Brightness Change Factor Register

Name	Address	Attribute	Initial Value
(2D_A) BLDY	0x04000054	W	0x0000
(2D_B) DB_BLDY	0x04001054	W	0x0000



The factor used for changing brightness is set with EVY in the BLDY [DB\_BLDY] Register. EVY is divided by 16 and is used as the pixel color factor in the equations below (when EVY exceeds 16, it is reset to 16.)

- Computations to increase brightness

$$\text{Display color (R)} = \text{1st pixel (R)} + (31 - \text{1st pixel (R)}) \times (\text{EVY} / 16)$$

$$\text{Display color (G)} = 1\text{st pixel (G)} + (31 - 1\text{st pixel (G)}) \times (\text{EVY} / 16)$$

$$\text{Display color (B)} = 1\text{st pixel (B)} + (31 - 1\text{st pixel (B)}) \times (\text{EVY} / 16)$$

- Computations to decrease brightness

$$\text{Display color (R)} = \text{1st pixel (R)} - \text{1st pixel (R)} \times (\text{EVY} / 16)$$

$$\text{Display color (G)} = \text{1st pixel (G)} - \text{1st pixel (G)} \times (\text{EVY} / 16)$$

$$\text{Display color (B)} = 1\text{st pixel (B)} - 1\text{st pixel (B)} \times (\text{EVY} / 16)$$

The computation results for brightness are rounded to the nearest integer.

## 5.8 Mosaic

The Mosaic size is set with the MOSAIC [DB\_MOSAIC] Register. Mosaic is turned on/off for each BG with the Mosaic Flag on the BG Control Register.

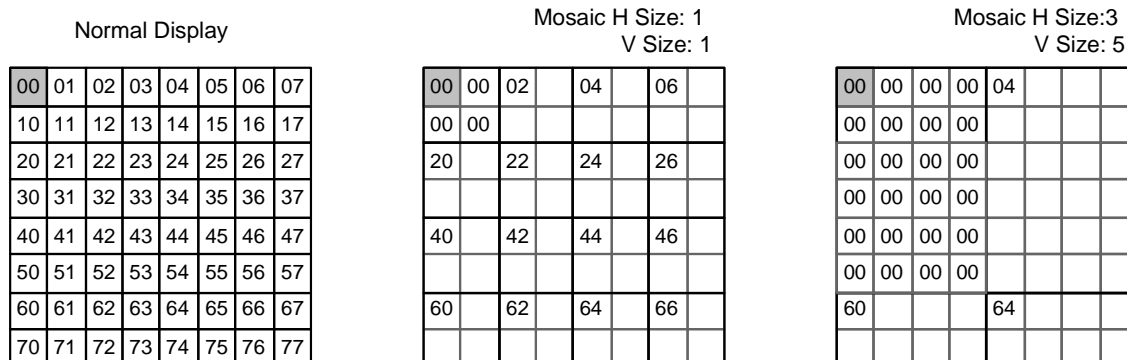
### Mosaic Register

Name	Address	Attribute	Initial Value
(2D_A) MOSAIC	0x0400004C	W	0x0000
(2D_B) DB_MOSAIC	0x0400104C	W	0x0000

15	8	7	0
V Size	H Size	V Size	H Size
OBJ Mosaic Size		BG Mosaic Size	

The Mosaic Size value specifies how many dots of a normal display should comprise each large dot displayed. The Mosaic display starts with the top-left dot on the screen and uses the dots spaced a distance of the Mosaic size from the top-left dot. All other dots are overwritten with the mosaic (see Figure 5-44). In other words, if the Mosaic size is set to 0, images display normally, even if Mosaic is on.

**Figure 5-44 : Display Changes According to Mosaic Size**



## 5.9 Display Priority

- BG Display Priority

Four levels of display priority can be set for BGs with the BG Control Register. When BGs have the same priority, the one with the lower BG number has higher priority. The Backdrop screen always has the lowest priority.

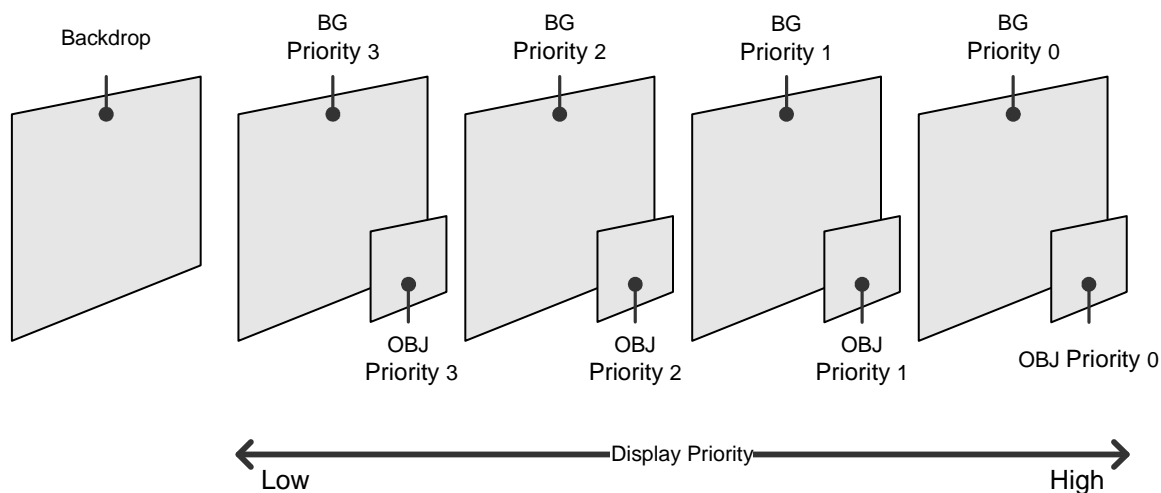
- OBJ Display Priority

Four levels of display priority can be set for OBJs with the OBJ Attribute Data stored in OAM. When OBJs have the same priority, the one with the lower OBJ number has higher priority.

- BG - OBJ Display Priority

If an OBJ and a BG have the same priority, the OBJ has higher priority than the BG (see Figure 5-45).

**Figure 5-45 : Display Priority**



### Workaround for the AGB OBJ Display Priority Problem

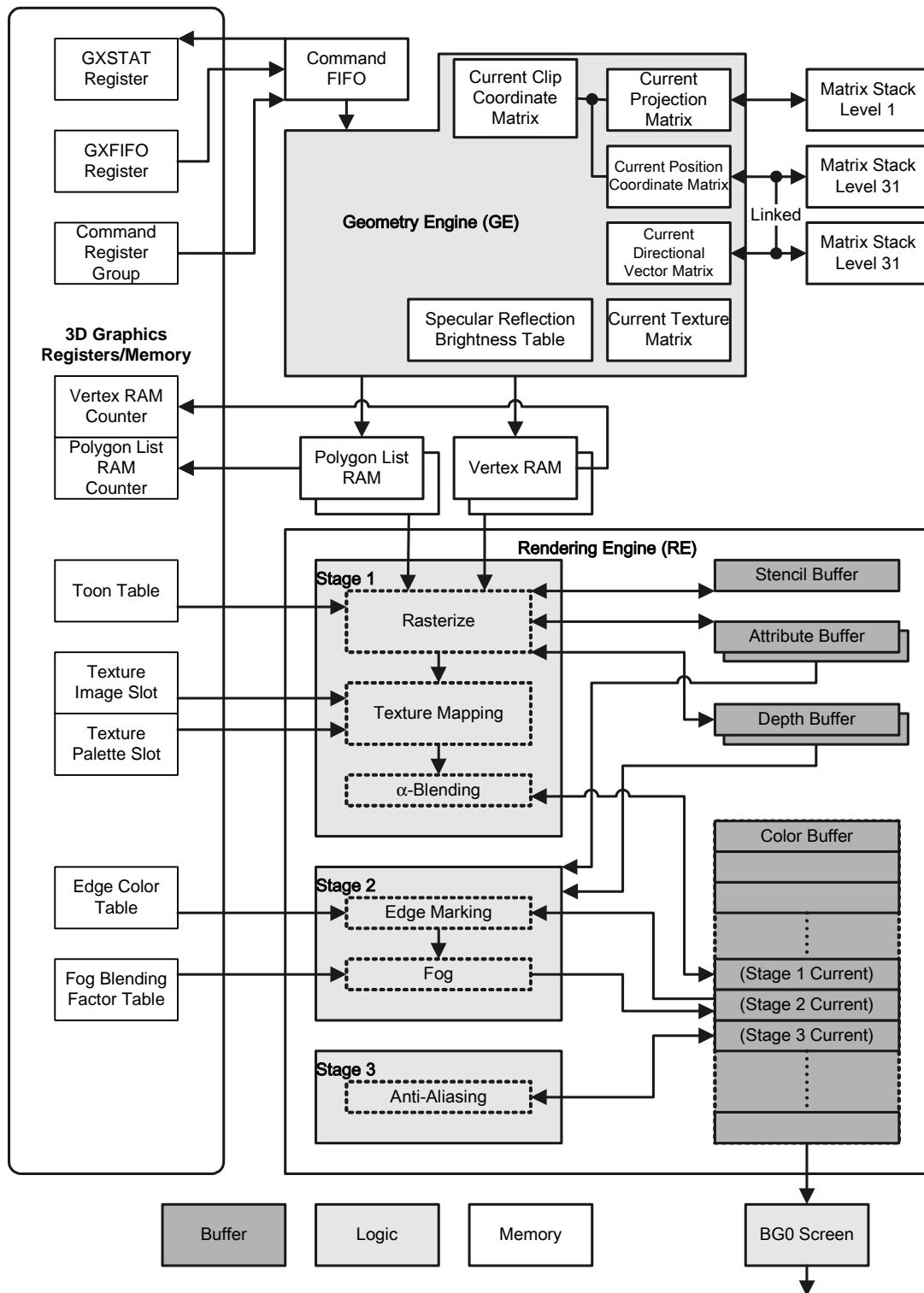
A problem with AGB caused an improper display when the OBJ numbers (the order OBJs are registered in OAM), and the OBJ priority levels were reversed. This problem is corrected in NITRO (unless NITRO is set to AGB-compatibility mode).



## 6 3D Graphics

Figure 6-1 is a schematic of the hardware block involved in the rendering of 3D graphics.

**Figure 6-1 : 3D Graphics Hardware Block Diagram**



- Polygon List RAM, Vertex RAM

The data that is passed from the Geometry Engine to the Rendering Engine is stored in Polygon List RAM and Vertex RAM.

Table 6-1 shows the capacity for this Polygon list RAM and Vertex RAM.

**Table 6-1 : Capacity of Polygon List RAM and Vertex RAM**

RAM	Capacity
<b>Polygon List RAM</b>	2048 polygons
<b>Vertex RAM</b>	6144 vertices

The capacity of polygon list RAM is just enough for 2048 triangular polygons. Because quadrilateral polygons have four vertices, fewer of these can be stored in this RAM. A maximum of 1706 can be stored for the quadrilateral strip because neighboring quadrilateral polygons share vertices.

- Buffers inside the Rendering Engine

The Stencil buffer, Attribute buffer, Depth buffer and Color buffer are memory regions that store information for each pixel. One buffer block in Figure 6-1 depicts a line buffer for an LCD horizontal line of 256 pixels. For more details, see the ["6.3 Rendering Engine"](#) on page 226.



## 6.1 3D Display Control

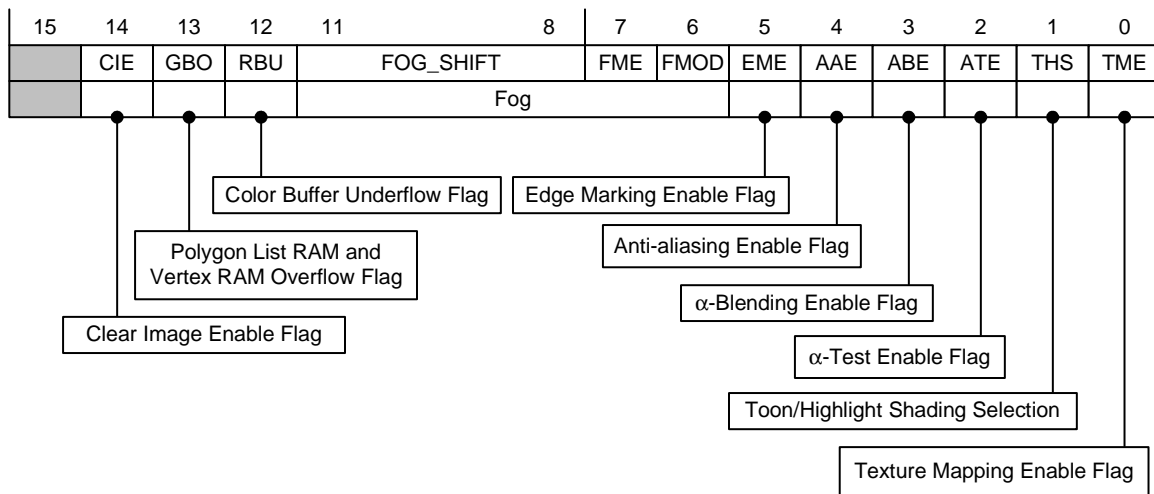
### DISP3DCNT: 3D Display Control Register

Name: DISP3DCNT

Address: 0x04000060

Attribute: R/W

Initial value: 0x0000



- CIE[d14] : Clear Image enable flag

Two VRAMs of 128 kilobytes each are used to set the values for Clear Color, Clear  $\alpha$ , Clear Depth, and Clear Fog.

The Clear  $\alpha$  value is specified with 1 bit, so you can select only transparent or opaque.

VRAM2 banks must be assigned to the Clear Image buffer with the RAM Control register 0.

Even when this feature is used, the register value is used for the Clear Polygon ID.

For further information, see ["6.3.3.2 Initializing with Clear Images"](#) on page 232.

- GBO[d13] : Polygon List RAM and Vertex RAM overflow flag

The flag is set to 1 when the Geometry Engine processes too many polygons and vertices and Polygon List RAM and Vertex RAM overflow.

Once an overflow has occurred, the bit remains 1 even if it is no longer overflowing. You can reset the flag by writing 1.

<b>0</b>	No overflow
<b>1</b>	Overflow

When the polygon list RAM or vertex RAM overflow, the polygon that caused the overflow and all the polygons after it are not registered.

- RBU[d12] : Color buffer underflow flag

This flag is set to 1 when rendering is not done in time to display and a Color buffer underflow occurs (a *line overflow* in the Rendering Engine).

Once the Color buffer has underflowed, this bit remains 1 even upon deletion. You can reset by writing 1. See ["6.3.2 Rendering Methods"](#) on page 228.

<b>0</b>	No underflow
<b>1</b>	Underflow

- [d11–d06] : Fog

This feature applies a fog effect. For details, see the ["6.3.1 Overview"](#) on page 226.

- [d11–d08] : Fog Shift

The depth value used by fog uses the upper 15 bits (called the fog depth value) of the 24 bits.

The Fog table is referenced using 5 bits of the depth value as an index.

When the Fog Shift is 0, the bits d14 - d10 of the depth value are referenced as the index. For every 1-step increase in the Fog Shift, the reference bits are shifted 1-bit to the right.

**Note:** Setting values of 11–15 is prohibited.

- FME[d07] : Fog master enable flag

0	Disable
1	Enable

- FMODE[d06] : Fog mode

Selecting 1 makes 3D objects appear to dissolve into a 2D background.

0	Fog blending using pixel's color value and $\alpha$ value.
1	Fog blending using only the pixel's $\alpha$ value.

- EME[d05] : Edge-marking enable flag

This feature draws an outline in the specified color around the edges of polygons with different polygon IDs.

For details, see the ["6.3.1 Overview"](#) on page 226.

0	Disable
1	Enable

- AAE[d04] : Anti-aliasing enable flag

This feature blends the edges of a polygon with the color value for the polygon behind it.

For details, see the ["6.3.1 Overview"](#) on page 226.

If you plan to capture the rendering result used as Bitmap OBJ, etc., disabling this flag gives you more natural images.

0	Disable
1	Enable

- ABE[d03] : Alpha-Blending enable flag

This feature blends the Color buffer's color with the fragment color in accordance with the fragment's alpha value.

For details, see the ["6.3.1 Overview"](#) on page 226.

0	Disable
1	Enable

- ATE[d02] :  $\alpha$ -Test enable flag

This feature enables you to skip the drawing of pixels that have an  $\alpha$ -value lower than the specified value.

For details, see the ["6.3.1 Overview"](#) on page 226.

<b>0</b>	Disable
<b>1</b>	Enable

- THS[d01] : Toon/Highlight Shading Selection

This bit selects the shading mode for the polygon specified for Toon shading/Highlight shading with the PolygonAttr command.

- TMOD : Toon/Highlight polygon mode

<b>0</b>	Toon shading
<b>1</b>	Highlight shading

- TME[d00] : Texture Mapping master enable flag

This selects whether to perform texture mapping.

<b>0</b>	Disable
<b>1</b>	Enable

## 6.2 Geometry Engine

### 6.2.1 Overview

Table 6-2 lists geometry engine specifications.

**Table 6-2 : Geometry Engine Specifications**

<b>Operating Frequency</b>	33.514MHz
<b>Coordinate Transformation</b>	Maximum 4 million vertices/sec
<b>Matrix Computation</b>	4×4 matrix computation and matrix stack
<b>Clipping</b>	6-plane clipping
<b>Lighting</b>	Light: Parallel light source × 4 Material: Reflected light (diffuse reflection, specular reflection, ambient reflection), Emission light
<b>Other features</b>	Backface culling; Specify display of 1-dot polygons (see note); Box culling test; Texture coordinate transformation; Adjust specular reflection shininess distribution

**Note:** A *1-dot polygon* is a polygon whose constituent coordinates (x, y) have been condensed to the same coordinate.

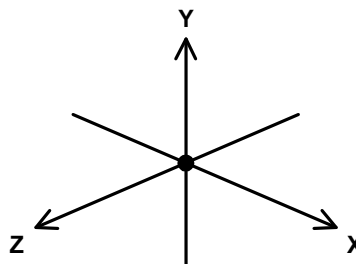
### 6.2.2 Coordinate System

In three-dimensional space, the coordinate system can be defined in two ways: as a right-handed coordinate system or as a left-handed coordinate system, depending on the direction of the z-axis relative to the x and y axes.

As a rule, NITRO adopts the right-handed coordinate system. However, because Z values are inverted with the projection matrix, the coordinates after clipping are in the left-handed coordinate system.

Figure 6-2 shows the relation of the x, y, and z axes in a right-handed coordinate system.

**Figure 6-2 : Right-Handed Coordinate System**



### 6.2.3 Coordinate Transformations

In OpenGL, when a model is located in view coordinates, the model is first multiplied by the model view matrix, which includes the view transformation. This is then multiplied by the projection matrix, and the apparent size of the model is determined based on the view volume.

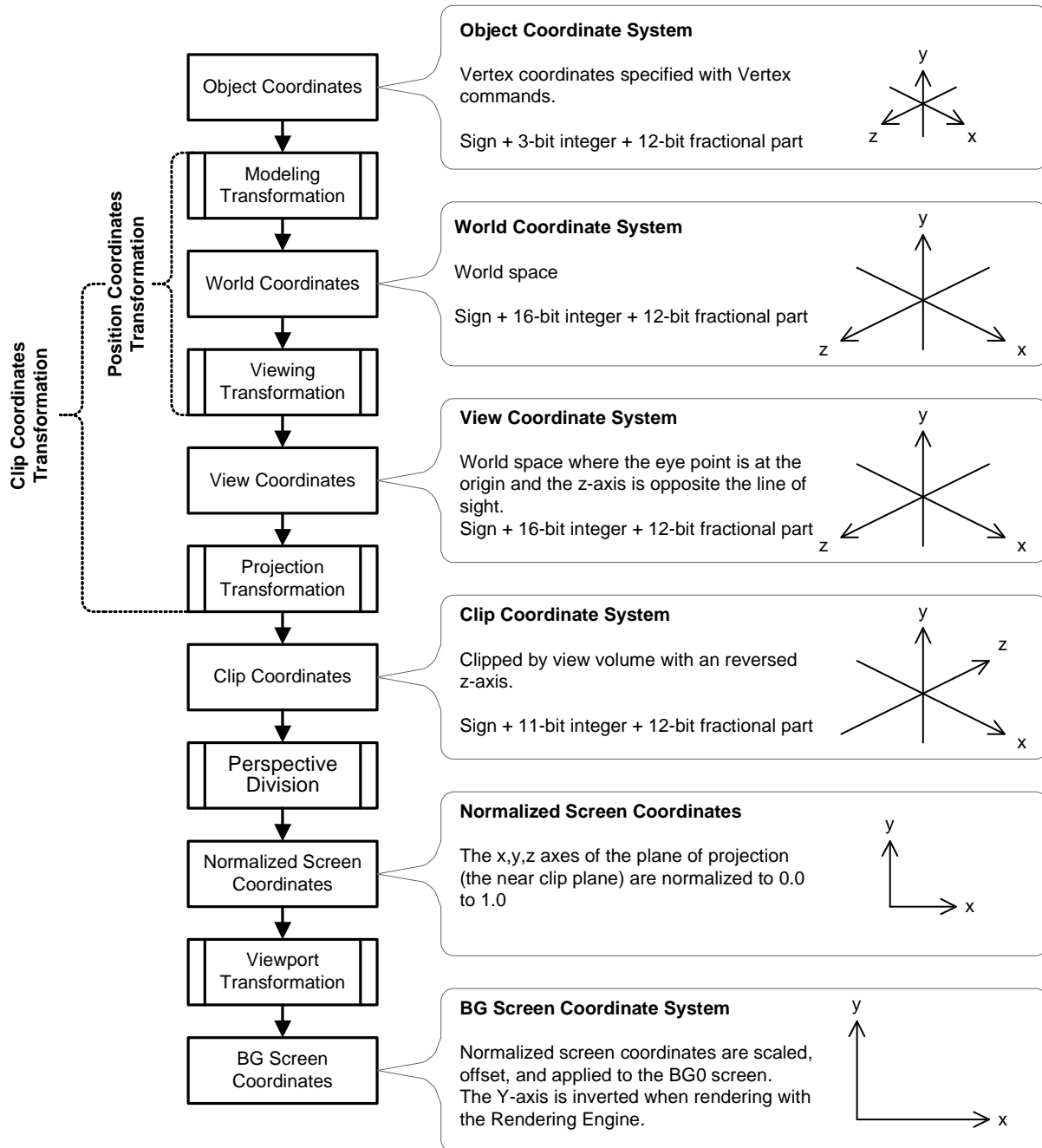
To reduce the load on the hardware, NITRO uses a clip coordinate matrix that is a concatenation of the projection matrix and the position coordinates matrix, so that clip coordinate conversion is done with only one coordinate transformation. Then the clip coordinate values (x, y, z, w) are divided by 2w (perspective division) after only the w coordinate is translated to get the normalized screen coordinates, and a scaling transformation is done on the BG screen coordinates by a viewport transformation.

For the vertex's normal vector and light vector, OpenGL performs a transformation with the transposed matrix of the modelview matrix. In contrast, NITRO assumes the vector is normalized, and uses only the rotational component's matrix (the orthogonal matrix) for the transformation. This kind of transformation is called a *directional vector transformation*.

In OpenGL, vertex position coordinates and directional vectors are transformed into view coordinates just by setting the modelview matrix. But in NITRO, the vertex position coordinates and directional vectors are transformed by separate matrices, and these are separately defined as the *position coordinates matrix* and *directional vector matrix*.

Figure 6-3 illustrates the coordinate transformation flow on the NITRO.

**Figure 6-3 : Coordinate Transformation Flow Chart**



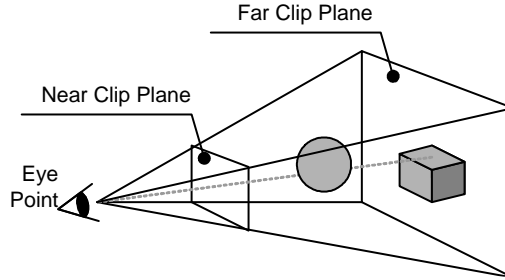
## 6.2.4 Projection Transformations

The perspective of the viewed polygon object from the eye point is defined by the view volume.

The view volumes and projection matrices for perspective projections (Figure 6-4) and orthogonal projections (["Figure 6-5 : Orthogonal Projections"](#) on page 162) are described below.

### 1. Perspective Projections

**Figure 6-4 : Perspective Projections**



#### a. Left-Right Asymmetrical Perspective Projection

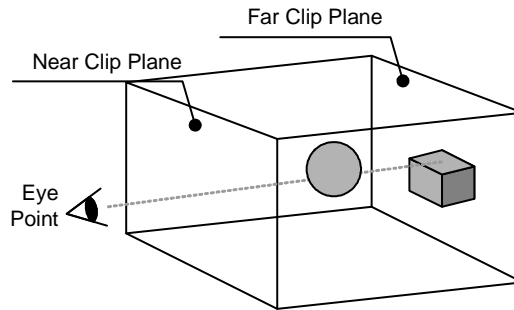
$$Frustum = \begin{bmatrix} \frac{2n}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2n}{t-b} & 0 & 0 \\ \frac{r+l}{r-l} & \frac{t+b}{t-b} & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix} \times scaleW$$

#### b. Left-Right Symmetrical Perspective Projection

$$Perspective = \begin{bmatrix} \frac{\cos\theta}{asp \cdot \sin\theta} & 0 & 0 & 0 \\ 0 & \frac{\cos\theta}{\sin\theta} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -1 \\ 0 & 0 & -\frac{2fn}{f-n} & 0 \end{bmatrix} \times scaleW$$

## 2. Orthogonal Projections

Figure 6-5 : Orthogonal Projections



$$Ortho = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & 0 \\ 0 & \frac{2}{t-b} & 0 & 0 \\ 0 & 0 & -\frac{2}{f-n} & 0 \\ -\frac{r+l}{r-l} & -\frac{t+b}{t-b} & -\frac{f+n}{f-n} & 1 \end{bmatrix}$$

t : Top edge y-coordinate of near clip plane

b : Bottom edge y-coordinate of near clip plane

r : Right edge x-coordinate of near clip plane

l : Left edge x-coordinate of near clip plane

n : Distance from eye point to near clip plane

f : Distance from eye point to far clip plane

$\theta$  : Angle of field-of-view (screen angle) in vertical (y) direction  $\div 2$

asp : Aspect ratio of width to height of field-of-view (height:width ratio = width of field-of-view  $\div$  height of field-of-view)

scaleW : Parameter for precision-adjusting view volume

(Use to change the scaling of clip coordinate space and increase the precision of the orthogonal screen coordinates after perspective division.)



## 6.2.5 Depth Buffering

### 1. For Perspective Projections

The perspective projection matrix parameters are set as shown below. (For details about elements p0-p5, see "[6.2.4 Projection Transformations](#)" on page 161).

$$M = \begin{bmatrix} p0 & 0 & 0 & 0 \\ 0 & p2 & 0 & 0 \\ p1 & p3 & p4 & -1 \\ 0 & 0 & p5 & 0 \end{bmatrix} \times scaleW$$

When the View coordinates are (x, y, z, 1), the Clip coordinates (x', y', z', w') are as follows:

$$x' = (p0 \times x + p1 \times z) \times scaleW$$

$$y' = (p2 \times y + p3 \times z) \times scaleW$$

$$z' = (p4 \times z + p5 \times 1) \times scaleW$$

$$w' = -z \times scaleW$$

When each clip coordinate component is translated by w' and then divided by 2w' (perspective division), the normalized screen coordinates (x'', y'', z'', 1) are obtained. The z'' component is calculated as follows:

$$z'' = \frac{z' + w'}{2w'} = \frac{(p4 \times z + p5 - z) \times scaleW}{-2z \times scaleW} = \frac{1}{2} - \frac{p4}{2} - \frac{p5}{2z}$$

p4 and p5 are elements of the perspective projection matrix, so using the matrix shown in the equation "[a. Left-Right Asymmetrical Perspective Projection](#)" on page 161 yields:

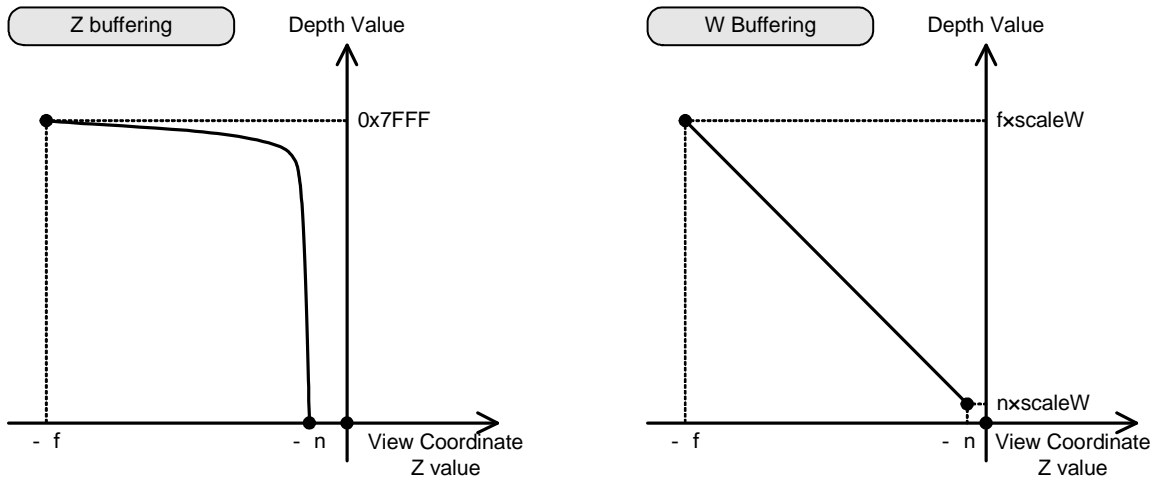
$$z'' = \frac{f}{(f - n)} \left( 1 + \frac{n}{z} \right) \quad (-far \leq z \leq -near)$$

During Z buffering, this z'' value is multiplied by 0x7FFF to get the depth value z'''. The z''' value is proportional to the inverse of the View coordinate z value, so in View coordinate space the position coordinates are more precise the closer they are to the eye point, and less precise the farther they are away from the eye point. As a result, when you represent a large space, drawings that are farther away tend to be more imprecise. To resolve this problem, NITRO supports *W buffering*, which uses clip coordination as the depth value.

Because the depth value is taken as a multiple of the view coordinate z from the equation...

$$w' = -z \times scaleW \quad (-far \leq z \leq -near)$$

... the precision of rendering in the distance improves. The trade-off is that this diminishes the precision of nearby images compared to Z-buffering. But by enlarging the view volume space and adjusting scaleW in order to maintain a certain level of resolution, this should not present a problem for nearby images. (See Figure 6-6.)

**Figure 6-6 : Z-Buffering and W-Buffering (Perspective Projection)**

f : Distance from eye point to far clip plane

n : Distance from eye point to near clip plane

## 2. For Orthogonal Projections

The orthogonal projection matrix parameters are set as shown below. (For details about elements p0-p5, see ["6.2.4 Projection Transformations"](#) on page 161.

$$M = \begin{bmatrix} p0 & 0 & 0 & 0 \\ 0 & p2 & 0 & 0 \\ 0 & 0 & p4 & 0 \\ p1 & p3 & p5 & 1 \end{bmatrix} \times scaleW$$

When the View coordinates are (x, y, z, 1), the Clip coordinates (x', y', z', w') are as follows:

$$x' = (p0 \times x + p1 \times 1) \times scaleW$$

$$y' = (p2 \times y + p3 \times 1) \times scaleW$$

$$z' = (p4 \times z + p5 \times 1) \times scaleW$$

$$w' = 1 \times scaleW$$

When each component of the clip coordinate is translated by w' and then divided by 2w'

When each clip coordinate component is translated by w' and then divided by 2w' (perspective division), the normalized screen coordinates (x'', y'', z'', 1) are obtained. When scaleW = 1, coordinates are translated by 1 and divided by 2, so the clip coordinate system values -1.0 to 1.0 are transformed into the normalized screen coordinate system values 0.0 to 1.0.

The normalized screen coordinate z'' component is calculated as follows:

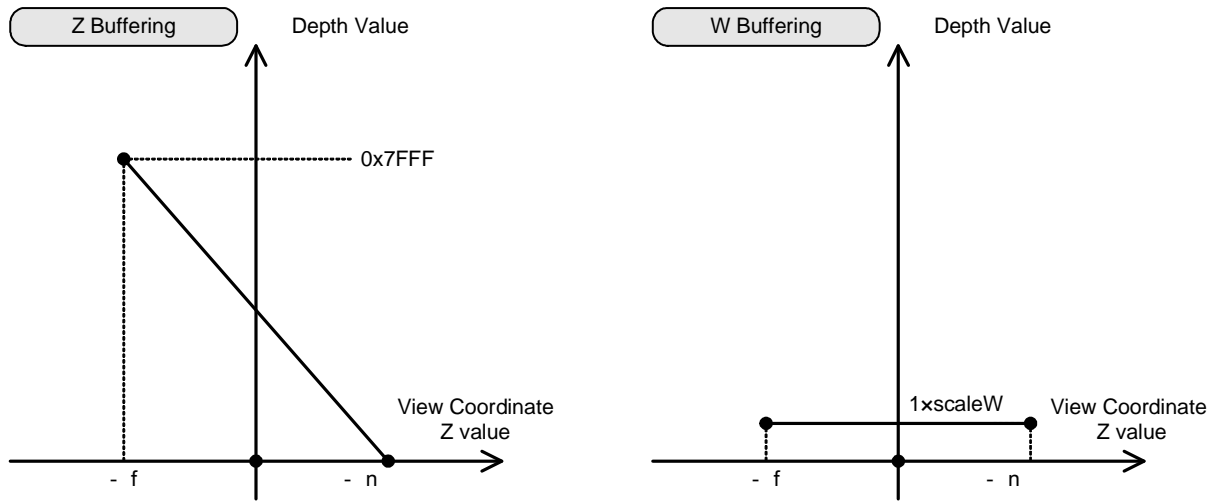
$$z'' = \frac{z' + w'}{2w'} = \frac{(p4 \times z + p5 + 1) \times scaleW}{2 \times scaleW} = \frac{p4 \times z + p5 + 1}{2}$$

p4 and p5 are elements of the orthogonal projection matrix, so using the matrix shown in the equation ["2. Orthogonal Projections"](#) on page 162 yields:

$$z'' = \frac{-1}{(f - n)}(z + n) \quad (-far \leq z \leq -near)$$

During Z buffering, this z'' value is multiplied by 0x7FFF to get the depth value z'''. This z''' value is proportional to the View coordinate z value, so no problems arise with the precision of distant images.

When w buffering is used, the w' clip coordinate which serves as the depth value is always fixed to the value 1 x scaleW. For this reason, w buffering is not used with orthogonal projections. (See Figure 6-7.)

**Figure 6-7 : Z-Buffering and W-Buffering (Orthogonal Projection)**

$f$  : Distance from eye point to far clip plane

$n$  : Distance from eye point to near clip plane

### **Depth value format**

The NITRO Depth buffer uses 24 bits for each pixel, so the depth value must fit inside that range.

For Z buffering, the depth value is the value that expresses the distance from the near clip plane to the far clip plane with 24-bit precision.

For W buffering, the distance from the eye point in the View coordinate system must fit within the 24-bit precision (sign + 11-bit integer + 12-bit fractional part) range of the Clip coordinate system, and the depth value is the result of translating the W value of the clip coordinates a distance of W and then dividing by two (12-bit integer + 12-bit fractional part).

**Note:** Make sure that the Clip coordinate values do not exceed the 24-bit range.

For fog, the depth value is the upper 15 of the 24 bits. See "[6.3.3.1 Initializing with the Clear Registers](#)" on page 231 for details.

## 6.2.6 Geometry Commands

To transfer the data of matrices and polygons, etc. to the Geometry Engine requires the writing of command strings to Command FIFO. There are two ways to write to Command FIFO:

Method 1: Write parameters to the group of command registers mapped in the register space of the main processor.

Write the parameters into Command registers, and the command code and parameters are automatically written into Command FIFO. This method works when the CPU will process one command at a time.

Method 2: Write command code and parameters to the Command FIFO register.

This method is appropriate for transferring large amounts of data to the Geometry Engine, such as for DMA transfers of command strings stored in main memory.

**Note:** The Geometry Command register group and Command FIFO are specialized for 32-bit access. Whether you are writing using the CPU or DMA, make sure the access width is 32 bits.

When an attempt is made to write to Command FIFO when it is full, the process enters a wait state until 32 bits open up in Command FIFO. During this wait state the bus cannot be used even by another bus master. You can avoid this situation by doing the following:

### Transfer commands using the DMA Geometry Command FIFO startup mode

In this mode, DMA is started when Command FIFO becomes less than half full, sending in units of 112 words (see note) until the specified transfer volume is reached. See [Chapter 7](#).

**Note:** If commands have been packed, the word count equals the number of words before unpacking.

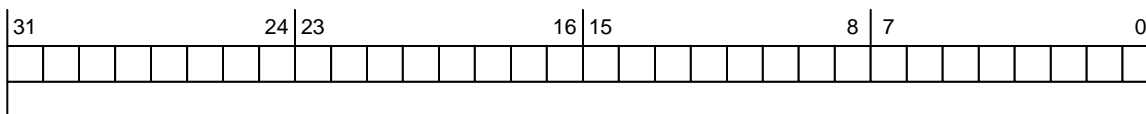
You can determine the status of Command FIFO by checking the Command FIFO status flag in the Geometry Engine status register (GXSTAT).

## GXFIFO: Command FIFO Register

Address: 0x04000400

(Image: 0x04000404 - 0x0400043F)

Attribute: W



Data written to the Command FIFO register is sent to Command FIFO.

Command FIFO has a depth of 32 bits x 256 levels.

Be careful not to transfer undefined command codes.

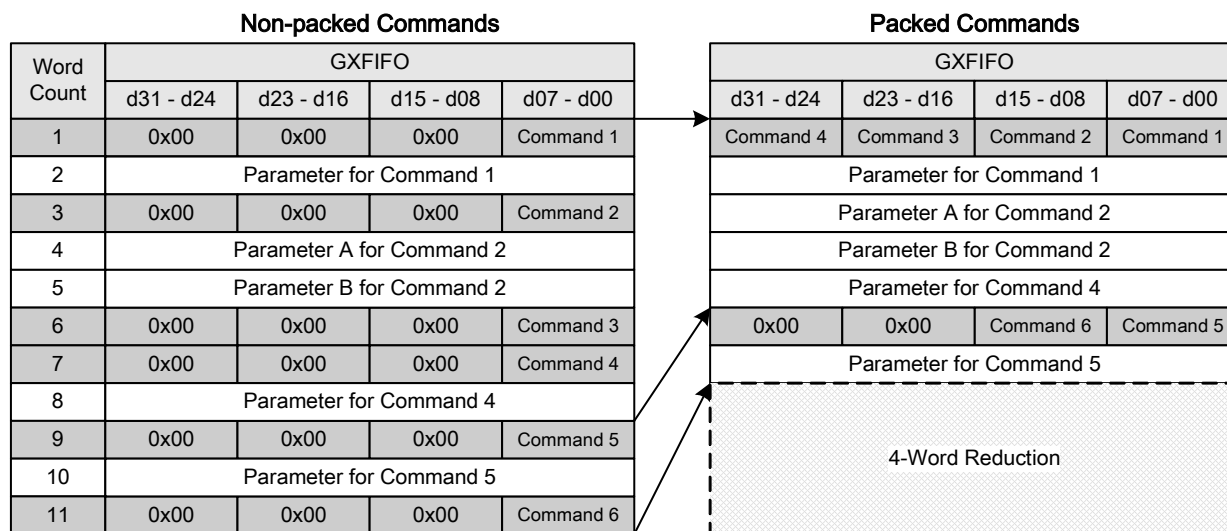
## Command Packing

When transferring command strings to the Command FIFO register, the command strings can be compressed by packing up to four command codes in one word.

Packed command strings are first decompressed and then stored in command FIFO. The packed command strings are stored in command FIFO in order starting from the low-order byte, so you need to pack the command codes starting from the low-order address and fill the empty higher-order bytes with 0.

Figure 6-8 shows the different command transfers for commands that are packed and that are not packed.

### Figure 6-8 : Transferring Packed and Non-Packed Commands



In the example shown above, Commands 1, 4 and 5 have one parameter each, Command 2 has two parameters, and Commands 3 and 6 have no parameters.

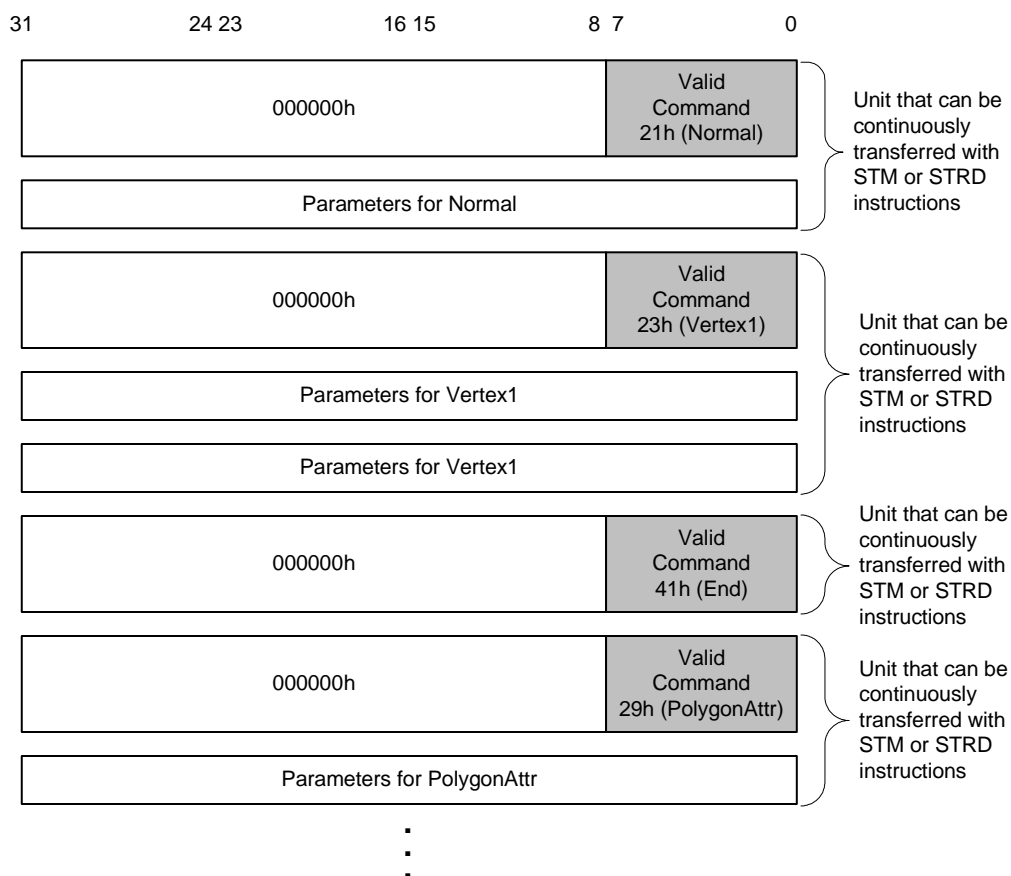
In Figure 6-8, a data volume of 11 words is sent to the Command FIFO register when the commands are not packed, but when commands are packed only 7 words are sent, for a savings of 4 words.

### **Precautions regarding the CPU continuously writing to the Geometry FIFO**

Continuous writing to the Geometry FIFO using STM or STRD instructions can occur properly only when the two following conditions are met. See Figure 6-9.

- Command Pack is not used
- Write only one command-parameter pair at a time

**Figure 6-9 : Continuous Writing to the Geometry FIFO using STM or STRD Instructions**



The “Unit that can be continuously transferred with STM or STRD instructions” mentioned above can be written at a single time. However, do not perform a write with STM or STRD instructions that exceed this unit. Leave a blank interval of one system cycle between each “Unit that can be continuously transferred with STM or STRD instructions.”

### **Cautions regarding Data Arrays for Command Packs**

Writing to the Geometry FIFO when command packs are used can occur properly only when one of the following conditions are met whether using the CPU or DMA.

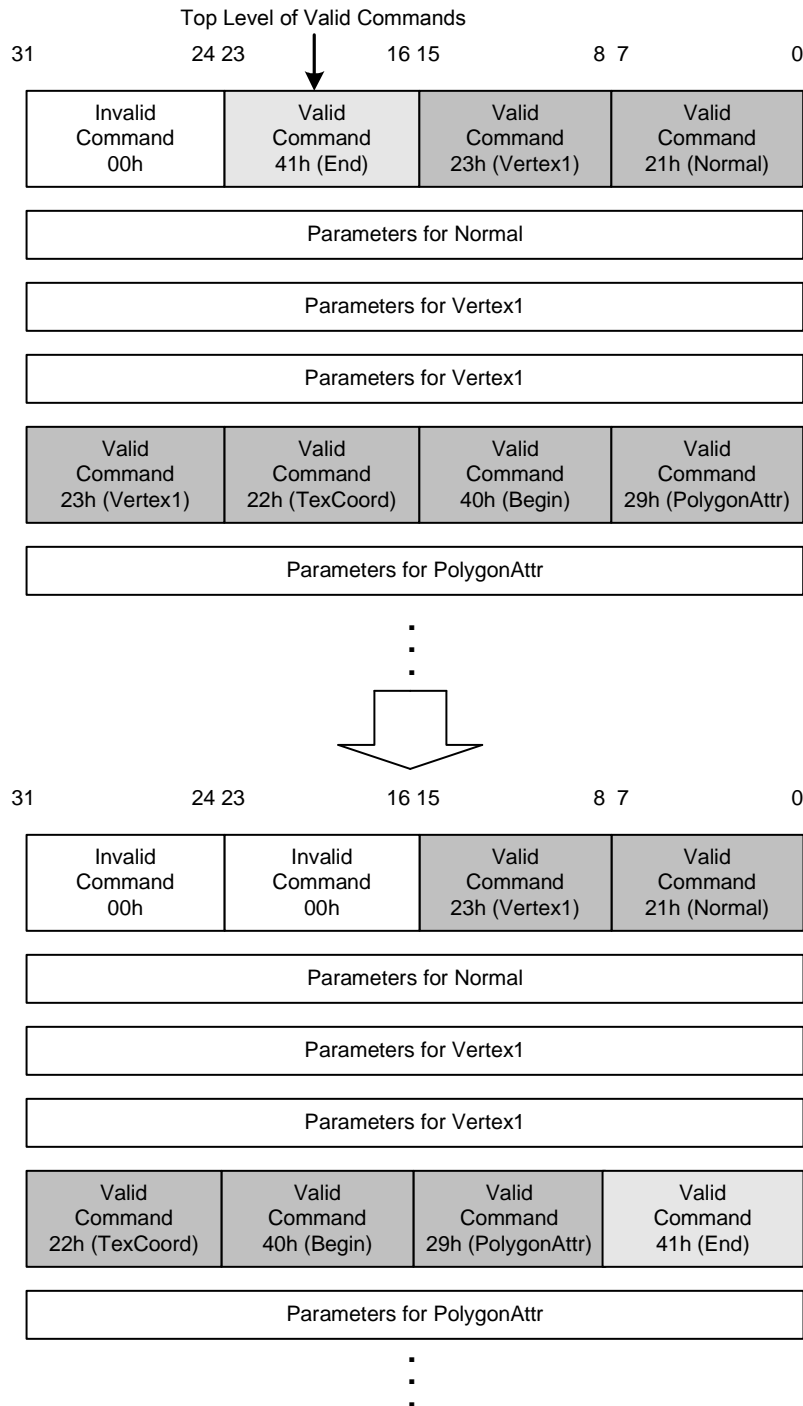
- Do not have a command without parameters (see Note 2 below) come in the top level of valid commands in the command pack (see Note 1 below). See Figures for details.

**Note 1:** “Valid command” indicates a command defined within the region between 0x10 and 0xFF. “Invalid Commands” are in the region between 0x00 and 0x0F.

**Note 2:** A “command without parameters” is one of the four following commands.

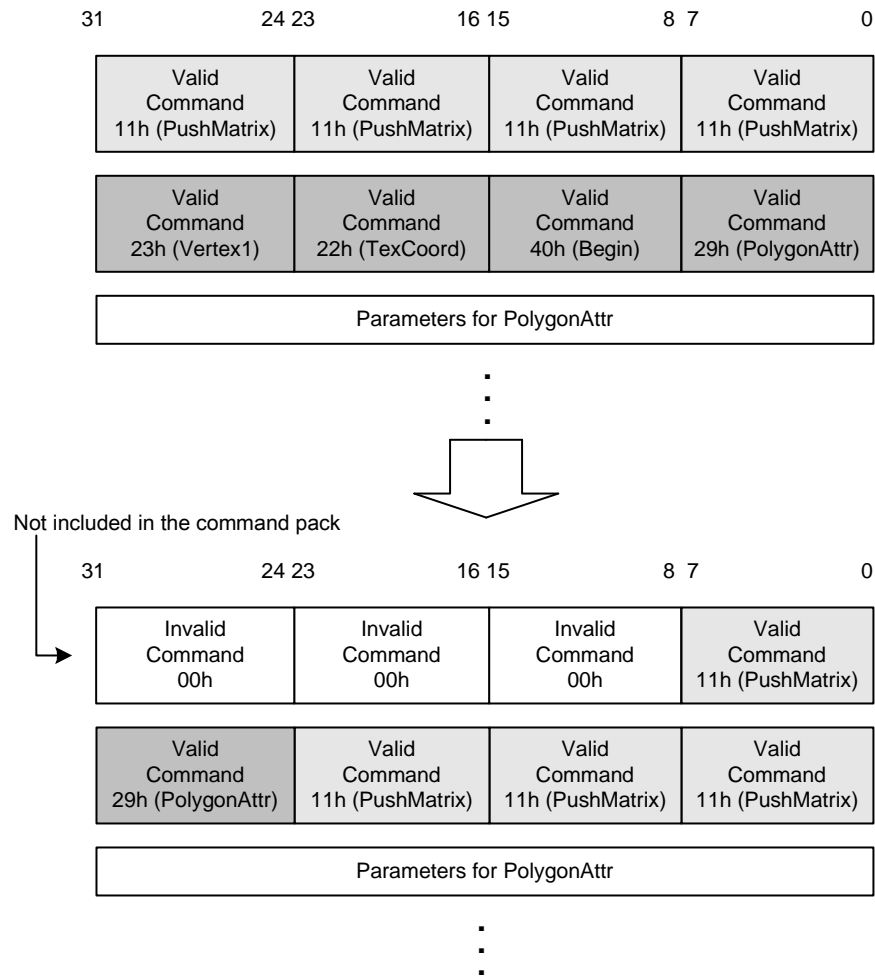
- PushMatrix
- LoadIdentity
- End
- Commands undefined within the region between 0x10 and 0xFF

**Figure 6-10 : Case 1: Preventing a Command without Parameters from Being the First Valid Command**





**Figure 6-11 : Case 2: Preventing a Command without Parameters from Being the First Valid Command**



- When a command without parameters comes in the top level of valid commands in the command pack, insert zeros at the end of the parameter array corresponding to that command pack.

**Figure 6-12 : When the First Valid Command has no Parameters**

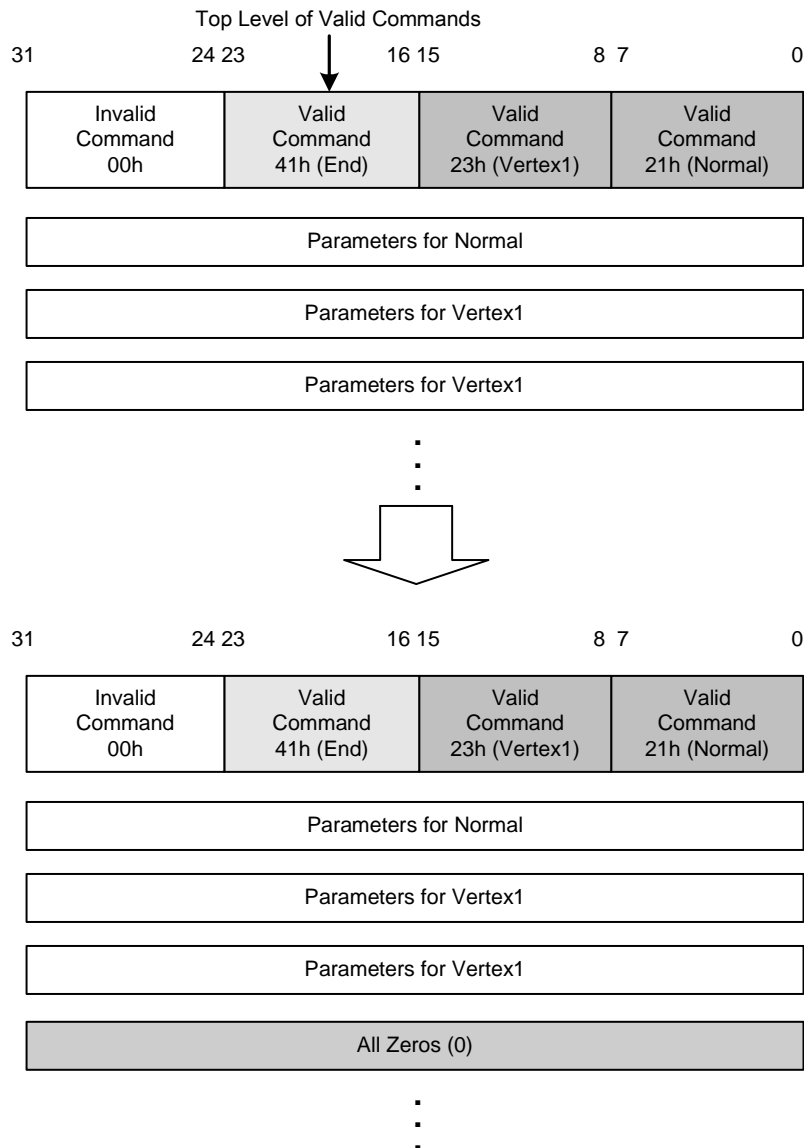


Table 6-3 and Table 6-4 show lists of Geometry Commands.

**Table 6-3 : Geometry Commands (in Command Code Order)**

Category	Feature	Command name	Command register address (see note)	Command code	No. of words in parameter	Page
—	No operation	Nop	—	0x00	0	None
<b>Matrix mode</b>	Sets the matrix mode	MatrixMode	0x440	0x10	1	<a href="#">181</a>
<b>Operations on the current matrix</b>	Pushes to stack	PushMatrix	0x444	0x11	0	<a href="#">186</a>
	Pops from stack	PopMatrix	0x448	0x12	1	<a href="#">186</a>
	Writes to specified location in stack	StoreMatrix	0x44C	0x13	1	<a href="#">187</a>
	Reads from specified location in stack	RestoreMatrix	0x450	0x14	1	<a href="#">187</a>
	Initializes a unit matrix	Identity	0x454	0x15	0	<a href="#">182</a>
	Sets a 4x4 matrix	LoadMatrix44	0x458	0x16	16	<a href="#">182</a>
	Sets a 4x3 matrix	LoadMatrix43	0x45C	0x17	12	<a href="#">182</a>
	Multiplies a 4x4 matrix	MultMatrix44	0x460	0x18	16	<a href="#">183</a>
	Multiplies a 4x3 matrix	MultMatrix43	0x464	0x19	12	<a href="#">183</a>
	Multiplies a 3x3 matrix	MultMatrix33	0x468	0x1A	9	<a href="#">184</a>
	Multiplies a scale matrix	Scale	0x46C	0x1B	3	<a href="#">185</a>
	Multiplies a translation matrix	Translate	0x470	0x1C	3	<a href="#">184</a>
<b>Vertex information</b>	Directly sets vertex color	Color	0x480	0x20	1	<a href="#">201</a>
	Sets normal vector	Normal	0x484	0x21	1	<a href="#">202</a>
	Sets texture coordinates	TexCoord	0x488	0x22	1	<a href="#">206</a>
<b>Vertex coordinates</b>	Sets the vertex coordinates	Vertex	0x48C	0x23	2	<a href="#">202</a>
	Same as above	Vertex10	0x490	0x24	1	<a href="#">203</a>
	Sets the XY coordinates of the vertex	VertexXY	0x494	0x25	1	<a href="#">203</a>
	Sets the XZ coordinates of the vertex	VertexXZ	0x498	0x26	1	<a href="#">203</a>
	Sets the YZ coordinates of the vertex	VertexYZ	0x49C	0x27	1	<a href="#">203</a>
	Sets vertex using the differential value of the last-set coordinate	VertexDiff	0x4A0	0x28	1	<a href="#">204</a>
<b>Polygon attribute</b>	Sets the polygon attribute	PolygonAttr	0x4A4	0x29	1	<a href="#">196</a>
<b>Texture information</b>	Sets the texture parameters	TexImageParam	0x4A8	0x2A	1	<a href="#">207</a>
	Sets the base address of the texture palette	TexPlttBase	0x4AC	0x2B	1	<a href="#">212</a>
<b>Material</b>	Sets the colors for ambient reflection and diffuse reflection	MaterialColor0	0x4C0	0x30	1	<a href="#">192</a>
	Sets the colors for emission light and specular reflection	MaterialColor1	0x4C4	0x31	1	<a href="#">192</a>
	Sets the specular reflection shininess table	Shininess	0x4D0	0x34	32	<a href="#">193</a>

Category	Feature	Command name	Command register address (see note)	Command code	No. of words in parameter	Page
<b>Light</b>	Sets the directional vector for light	LightVector	0x4C8	0x32	1	<a href="#">189</a>
	Sets the light color	LightColor	0x4CC	0x33	1	<a href="#">189</a>
<b>Vertex list begin/end</b>	Declares the start of the vertex list	Begin	0x500	0x40	1	<a href="#">200</a>
	Declares the end of the vertex list	End	0x504	0x41	0	<a href="#">201</a>
<b>Swap Rendering Engine reference data</b>	Swaps the data group referenced by the Rendering Engine	SwapBuffers	0x540	0x50	1	<a href="#">178</a>
<b>Viewport</b>	Sets the viewport	ViewPort	0x580	0x60	1	<a href="#">180</a>
<b>Test</b>	Tests whether the box is inside the view volume	BoxTest	0x5C0	0x70	3	<a href="#">216</a>
	Sets position coordinates for test	PositionTest	0x5C4	0x71	2	<a href="#">218</a>
	Sets directional vector for test	VectorTest	0x5C8	0x72	1	<a href="#">218</a>

**Note:** The Command register address values shown here are offset from address 0x04000000.

Be careful not to issue undefined command codes to the Geometry Engine's command FIFO.

**Table 6-4 : Number of Geometry Command Run Cycles & Timing Related to Command Issue (in Command Code Order)**

Command Name	Run cycle number	Issue timing	When settings take effect	When settings are destroyed
Nop		No Restriction		

Command Name	Run cycle number	Issue timing	When settings take effect	When settings are destroyed
MatrixMode	1	No Restriction	When command is executed	When next MatrixMode command is executed
PushMatrix	17			When next Push/StoreMatrix command is executed
PopMatrix	36			When next Matrix change command is executed
StoreMatrix	17			When next Push/StoreMatrix command is executed
RestoreMatrix	36			When next Matrix change command is executed
Identity	19			
LoadMatrix44	34			
LoadMatrix43	30			
MultMatrix44	35*			
MultMatrix43	31*			
MultMatrix33	28*			
Scale	22			
Translate	22*			
Color	1			When next Color, Normal command is executed
Normal	9 – 12 (+2)*2, 5			
TexCoord	1(+1) <sup>*5</sup>	When next TexCoord command is executed		
Vertex	9(+2) <sup>*5</sup>	Only between Begin–End	When next Vertex related command is executed	
VertexShort	8(+2) <sup>*5</sup>			
VertexXY	8(+2) <sup>*5</sup>			
VertexXZ	8(+2) <sup>*5</sup>			
VertexYZ	8(+2) <sup>*5</sup>			
VertexDiff	8(+2) <sup>*5</sup>			
PolygonAttr	1	Only outside Begin – End	When Begin command is executed (settings are valid between Begin–End units) <sup>*3</sup>	When next PolygonAttr command is executed
TexImageParam	1	Per Polygon <sup>*4</sup>	When command is executed (settings are valid in polygon units)	When next TexImageParam command is executed
TexPlttBase	1			When next TexPlttBase command is executed
MaterialColor0	4	No Restriction	When Normal command is executed	When next MaterialColor0 command is executed
MaterialColor1	4			When next MaterialColor1 command is executed
Shininess	32			When next Shininess command is executed
LightVector	6			When next LightVector command is executed
LightColor	1			When next LightColor command is executed
Begin	1		When command is executed	When next Begin command is executed
End	1			No set value

Command Name	Run cycle number	Issue timing	When settings take effect	When settings are destroyed
SwapBuffers	392	Only outside Begin – End	When enter V-Blank period	When next SwapBuffers command is executed
ViewPort	1		When command is executed	When next ViewPort command is executed
BoxTest	103			When next BoxTest command is executed
PositionTest	9	Per Polygon *4		When next PositionTest is executed
VectorTest	5	No Restriction		When next VectorTest is executed

The number of run cycles is a system clock (33.514Mhz) converted value.

\* When in Position and Vector Simultaneous Setting mode, this takes another 30 cycles.

\*2 Is increased according to the number of lights that are enabled (ON)

\*3 The PolygonAttr command is enabled with the Begin command. However, to reflect the light enable flag on vertex color, issue a Normal command again to recalculate the lighting.

\*4 Concerning the Polygon unit:

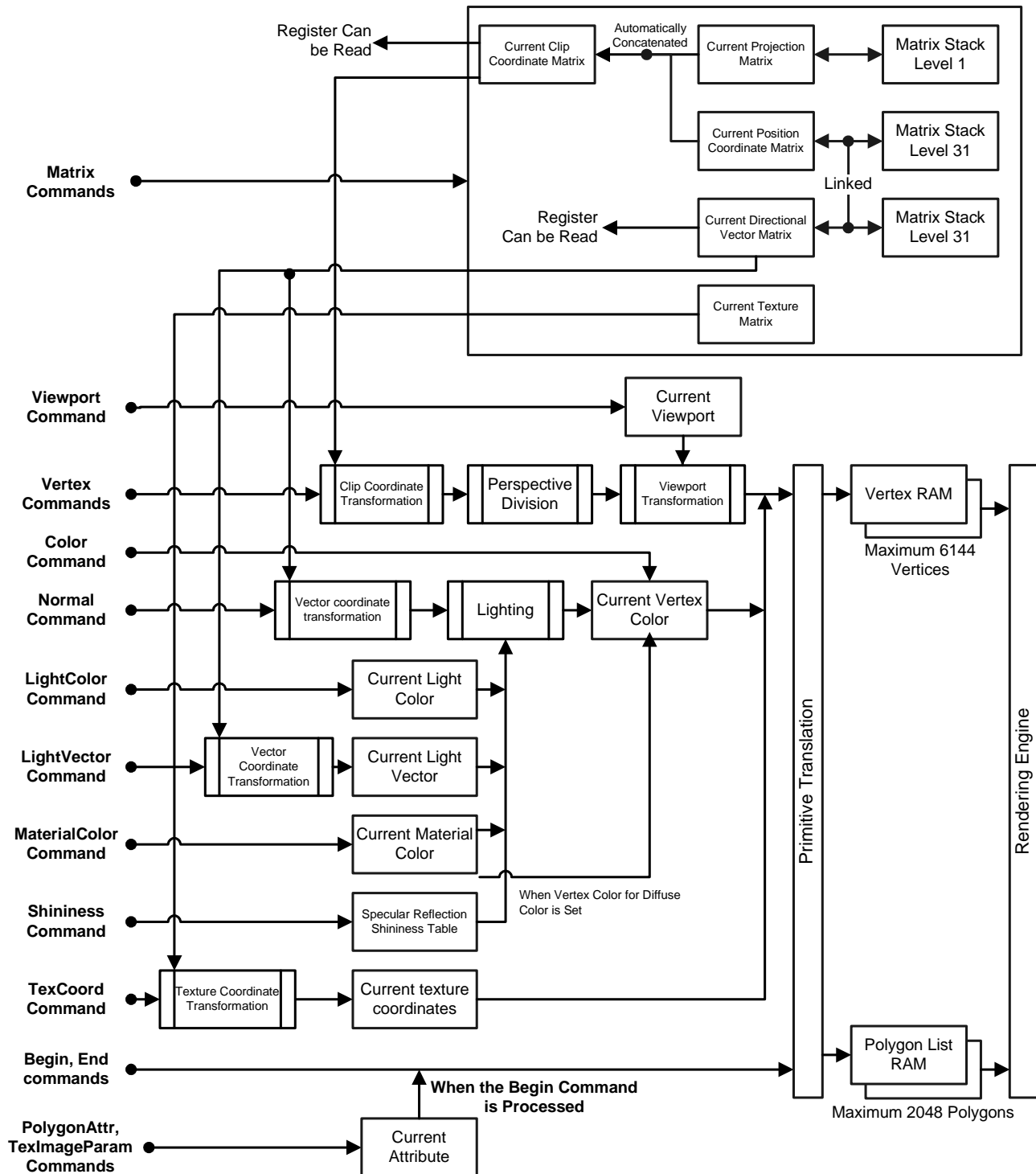
Commands that can be issued in polygon units can be issued in the Vertex-related command string at polygon breakpoints (breakpoints appear as • in the following table.)

Triangular Polygons	Triangular Polygon Strip	Quadrilateral Polygons	Quadrilateral Polygon Strip
<pre> • Begin   •   Vertex //Polygon 1   Vertex   Vertex   •   Vertex //Polygon 2   Vertex   Vertex   •   Vertex //Polygon 3   Vertex   Vertex   • End • </pre>	<pre> • Begin   •   Vertex //Polygon 1   Vertex   Vertex   Vertex   Vertex //Polygon 2   Vertex //Polygon 3   • End • </pre>	<pre> • Begin   •   Vertex //Polygon 1   Vertex   Vertex   Vertex   •   Vertex //Polygon 2   Vertex   Vertex   Vertex   •   Vertex //Polygon 3   Vertex   Vertex   Vertex   • End • </pre>	<pre> • Begin   •   Vertex //Polygon 1   Vertex   Vertex   Vertex   Vertex //Polygon 2   Vertex   Vertex //Polygon 3   Vertex   • End • </pre>

\* 5 The number of run cycles for commands corresponding to each source increases by the amount shown in parentheses when performing texture coordinate conversion in texture coordinate conversion mode.

Figure 6-13 shows a schematic of the main Geometry Command processes.

**Figure 6-13 : Schematic of the Main Geometry Command Processes**



**Note:** The flow shown here for the TexCoord command is for when the texture coordinate transformation mode is set to TexCoord source.

## 6.2.7 Swapping the Rendering Engine's Reference Data

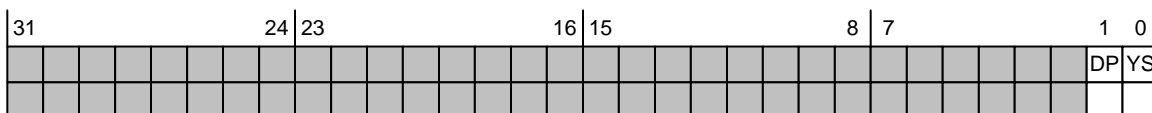
### SwapBuffers: Swaps the Data Group referenced by the Rendering Engine

Name: SWAP\_BUFFERS

Address: 0x04000540

Attribute: W

Command Code: 0x50



- DP[d01] : Depth buffering selection flag

Selects the value used for the depth test. To read how the depth value differs depending on the depth-buffering method, see ["6.2.5 Depth Buffering"](#) on page 163.

0	Select buffering with the Z value
1	Select buffering with the W value (Does not function properly for orthogonal projections)

- YS[d00] : Translucent polygon Y-sorting selection flag

Translucent polygons are polygons with ( $1 \leq \alpha \leq 30$ ) or mapped with a translucent texture.

Select manual sort mode to specify the order of rendering, such as when using shadow volume.

0	Auto-sort mode
1	Manual sort mode

The Geometry Engine writes the data passed to the Rendering Engine to Polygon List RAM.

For translucent polygons, you can choose whether to sort the data and then write, or to simply write the data in the order they are processed, without sorting. In Auto-sort mode, polygons are sorted from the polygon with smallest maximum Y value on the LCD (see note) to the polygon with the largest maximum Y value. Polygons that share the same maximum Y value are sorted in order from the polygon with smallest minimum Y value.

In Manual sort mode, polygons are sorted according to the order in which they are sent to the Geometry Engine.

In Auto sort mode, the Rendering Engine does not reference polygons with minimum Y values larger than the scan line (a line that is being rendered) or polygons with maximum Y values smaller than the scan line. However, all polygons are referenced in manual mode, which increases the load on the geometry engine (and reduces rendering efficiency). Because of this, be careful when using Manual sort mode when there are a large number of translucent polygons.

Opaque polygons are always Auto sorted.

Regardless of the sort mode, opaque polygons are always rendered before translucent polygons.

**Note:** Maximum Polygon Y value on the LCD

The Y coordinates are the inverse of Y coordinates in the BG Screen Coordinate Group in the Coordinate Transformation Flow Chart (Figure 6-3). Therefore, the maximum Y value for a polygon on the LCD is the minimum value in the BG Screen Coordinate Group.



- Reflecting the Depth Buffering Select Flag and Translucent Polygon Y Sorting Select Flag

These flags are reflected in the geometry engine from the next frame. However, because polygon list RAM and vertex RAM are double buffered, the rendering engine renders the data that the geometry engine stored in the previous frame. Therefore, data that the geometry engine outputs is rendered with an additional one-frame delay.

- Processing the SwapBuffers command

The SwapBuffers command is processed at the next V-Blank, regardless of when it was input. (The geometry engine is in wait status until the V-Blank period arrives.) The Polygon List RAM, Vertex RAM, rendering-related registers, and other data referenced by the Rendering Engine is swapped at the start of the next V-Blank period after the issuance of the SwapBuffers command. Because of this timing, rendering reflects the written graphics data in the next frame after the SwapBuffers command is issued.

## 6.2.8 Viewport

### ViewPort: Sets the Viewport

Name: VIEWPORT

Address: 0x04000580

Attribute: W

Command Code: 0x60

31	24	23	16	15	8	7	0
INTEGER_Y2		INTEGER_X2		INTEGER_Y1		INTEGER_X1	
Y2		X2		Y1		X1	

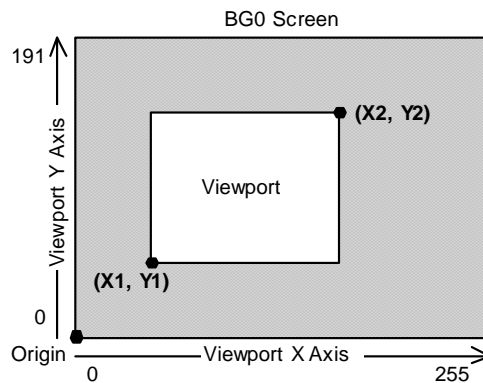
- Y2, X2 [d31–d24, d23–d16] : Top right coordinates  
Set Y2 to a value larger than Y1. Can be set in the range 0–191.  
Set X2 to a value larger than X1. Can be set in the range 0–255.
- Y1, X1[d15–d08, d07–d00] : Bottom left coordinates (the viewport origin)  
Set Y1 to a value smaller than Y2. Can be set in the range 0–191.  
Set X1 to a value smaller than X2. Can be set in the range 0–255.

This sets the position and the size of the viewport which draws 3D graphics on the BG0 screen.

The BG0's H offset is added to get the display position on the LCD.

Notice that the origin point is different than for the 2D coordinate group. (See Figure 6-14)

**Figure 6-14 : Size and Position of the Viewport**



**Note:** Rendering may result in one dot protruding from the right or bottom edge of the viewport.

## 6.2.9 Matrices

### 6.2.9.1 Manipulating the Current Matrix

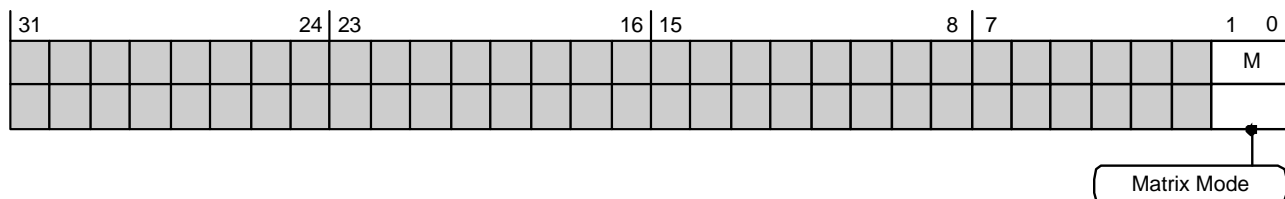
#### MatrixMode: Sets the Matrix Mode

Name: MTX\_MODE

Address: 0x04000440

Attribute: W

Command Code: 0x10



- M[d01–d00] : Matrix mode

<b>00</b>	Projection mode	... (For manipulating projection matrices)
<b>01</b>	Position mode	... (For manipulating position coordinate matrices)
<b>10</b>	Position & Vector Simultaneous Set mode	... (For manipulating position coordinate matrices and directional vector matrices)
<b>11</b>	Texture mode	... (For manipulating texture matrices)

This specifies the current matrix on which Matrix commands operate (this classification is called the Matrix mode).

#### Position mode and Position and Vector Simultaneous Set mode

NITRO does not use the hardware to make a unit normal vector. Therefore, to obtain correct lighting effects, you must set a unit vector as the normal in advance, and the directional vector matrix must be an *orthogonal matrix* (a matrix that does not change the length of the directional vector). When a model is transformed, the transformation matrix is usually used for both the position coordinates matrix and the directional vector matrix. But with NITRO, the directional vector matrix must be maintained as an orthogonal matrix, so depending on the type of transformation, sometimes the transformation matrix is used only for the position coordinates matrix.

Let the situation dictate whether to use Position mode or Position and Vector Simultaneous Set mode to set the Position and Vector simultaneously.

#### Examples

When rotating a model, usually the mode is set to the Position and Vector Simultaneous Setting and the MultMatrix command is executed to apply the rotation component of the matrix for both the position coordinates matrix and the directional vector matrix.

When the Scale command is used for model scaling, the directional vector matrix can be maintained as an orthogonal matrix, so the correct lighting effect can be obtained. (See the Scale command on page [185](#).)

When the scale matrix is applied with the MultMatrix command, the directional vector matrix is not maintained as an orthogonal matrix, and the lighting effect is brighter or darker than the original. Thus, it would be safer not to use this procedure.

When Position mode is selected, Matrix commands are applied only to the position coordinates matrix, so the correct lighting effect is obtained even when the scale matrix is applied with the MultMatrix command. You can also use this mode when you want to apply the rotation matrix only to the position coordinates matrix, but unnatural effects can arise. For example, sometimes the lighting effect does not change upon rotation, or a part that is not being illuminated ends up being the brightest.

### MTX IDENTITY: Initialize Current Matrix to Unit Matrix

Name: MTX\_IDENTITY      Address: 0x04000454      Attribute: W      Command Code: 0x15

[illegible]

### LoadMatrix44: Set 4x4 Matrix to Current Matrix

Name: MTX\_LOAD\_4x4      Address: 0x04000458      Attribute: W      Command Code: 0x16

31	30	24	23	16	15	12	11	8	7	0
S		INTEGER_M44						DECIMAL_M44		
Sign		Integer part						Decimal part		

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- M44 : 4x4 matrix elements  $m[x]$  ( $x = 0 - 15$ )

The matrix M is set as follows with elements m[0] to m[15]:

$$M = \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ m[12] & m[13] & m[14] & m[15] \end{bmatrix}$$

### LoadMatrix43: Set 4x3 Matrix to Current Matrix

Name: MTX\_LOAD\_4x3 Address: 0x0400045C Attribute: W Command Code: 0x17

31	30	24	23	16	15	12	11	8	7	0
S	INTEGER_M43							DECIMAL_M43		
Sign	Integer part							Decimal part		

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- M43 : 4x3 matrix elements  $m[x]$  ( $x = 0 - 11$ )

The matrix M is set as follows with elements m[0] to m[11]:

$$M = \begin{bmatrix} m[0] & m[1] & m[2] & 0 \\ m[3] & m[4] & m[5] & 0 \\ m[6] & m[7] & m[8] & 0 \\ m[9] & m[10] & m[11] & 1 \end{bmatrix}$$

**MultMatrix44: Multiply 4x4 Matrix by Current Matrix**

Name: MTX\_MULT\_4x4

Address: 0x04000460

Attribute: W

Command Code: 0x18

31	30	24	23	16	15	12	11	8	7	0
S	INTEGER_M44							DECIMAL_M44		
Sign	Integer part							Decimal part		

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- M44 : 4x4 matrix elements  $m[x]$  ( $x = 0 - 15$ )

The matrix M is set as follows with elements  $m[0]$  to  $m[15]$ :

$$M = \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ m[12] & m[13] & m[14] & m[15] \end{bmatrix}$$

If the current matrix is C, then the new current matrix  $C' = MC$ **MultMatrix43: Multiply 4x3 Matrix by Current Matrix**

Name: MTX\_MULT\_4x3

Address: 0x04000464

Attribute: W

Command Code: 0x19

31	30	24	23	16	15	12	11	8	7	0
S	INTEGER_M43							DECIMAL_M43		
Sign	Integer part							Decimal part		

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- M43 : 4x3 matrix elements  $m[x]$  ( $x = 0 - 11$ )

The matrix M is set as follows with elements  $m[0]$  to  $m[11]$ :

$$M = \begin{bmatrix} m[0] & m[1] & m[2] & 0 \\ m[3] & m[4] & m[5] & 0 \\ m[6] & m[7] & m[8] & 0 \\ m[9] & m[10] & m[11] & 1 \end{bmatrix}$$

If the current matrix is C, then the new current matrix  $C' = MC$

**MultMatrix33: Multiply 3x3 Matrix by Current Matrix**

Name: MTX\_MULT\_3x3

Address: 0x04000468

Attribute: W

Command Code: 0x1a

31	30	24	23	16	15	12	11	8	7	0
S	INTEGER_M33							DECIMAL_M33		
Sign	Integer part							Decimal part		

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- M33 : 3x3 matrix elements  $m[x]$  (  $x = 0 - 8$  )

The matrix M is set as follows with elements  $m[0]$  to  $m[8]$ :

$$M = \begin{bmatrix} m[0] & m[1] & m[2] & 0 \\ m[3] & m[4] & m[5] & 0 \\ m[6] & m[7] & m[8] & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If the current matrix is C, then the new current matrix  $C' = MC$ **Translate: Multiply Translation Matrix by Current Matrix**

Name: MTX\_TRANS

Address: 0x04000470

Attribute: W

Command Code: 0x1c

31	30	24	23	16	15	12	11	8	7	0
S	INTEGER_TRANSLATE							DECIMAL_TRANSLATE		
Sign	Integer part							Decimal part		

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- TRANSLATE : Translation matrix elements  $m[x]$  (  $x = 0 - 2$  )

The matrix M is set as follows with elements  $m[0]$  to  $m[2]$ :

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ m[0] & m[1] & m[2] & 1 \end{bmatrix}$$

If the current matrix is C, then the new current matrix  $C' = MC$

**Scale: Multiply the Scale Matrix by Current Matrix**

Name: MTX\_SCALE      Address: 0x0400046C      Attribute: W      Command Code: 0x1b

31	30	24	23	16	15	12	11	8	7	0
S	INTEGER_SCALE							DECIMAL_SCALE		
Sign	Integer part							Decimal part		

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- SCALE : Scale matrix elements  $m[x]$  (  $x = 0 - 2$  )

The matrix M is set as follows with elements  $m[0]$  to  $m[2]$ :

$$M = \begin{bmatrix} m[0] & 0 & 0 & 0 \\ 0 & m[1] & 0 & 0 \\ 0 & 0 & m[2] & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If the current matrix is C, then the new current matrix  $C' = MC$

The Scale command performs multiplication only on the position coordinates matrix, even when the matrix mode has been set to Position & Vector Simultaneous Setting mode. (If it were performed on the directional vector matrix, the direction and length of vectors would change and abnormal lighting effects would arise.)

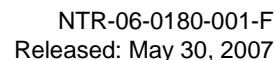




## 187

Address: 0x0400044C

Command Code: 0x13



- INDEX[d04–d00] : Storage position (Can set a value of 0 to 30)

When the Matrix mode is set to projection mode, the stack only has 1 level so the value of INDEX is treated as 0 no matter what value has been set.

### RestoreMatrix: Reads Matrix from Specified Position in Stack

Address: 0x04000450

Command Code: 0x14



- The value in the position specified by INDEX in the matrix stack is set as the matrix specified by the Matrix mode.

When the Matrix mode is set to projection mode, the stack only has 1 level so the value of INDEX is treated as 0 no matter what value has been set.

The matrix stack pointer moved by the PushMatrix and PopMatrix commands does not move after this command is issued.

This command is normally issued outside of the command string that runs from Begin to End, but it can also be issued between Vertex commands within the Begin to End command string.

<b>Command String Example 1</b>	StoreMatrix (i)→Translate→Begin→Vertex→Vertex→Vertex→End→ RestoreMatrix (i)
<b>Command String Example 2</b>	StoreMatrix (i)→Translate→StoreMatrix (i+1)→Begin→Vertex→RestoreMatrix (i)→Vertex→RestoreMatrix (i+1)→Vertex→End→RestoreMatrix (i)

In Command string Example 2, the `RestoreMatrix` command is issued between `Vertex` commands to realize stitching and sprite polygon deformations. Stitching is a type of skinning. Sprite polygons are polygons displayed in 2D.

### 6.2.9.3 Reading the Current Matrix

#### ClipMatrix\_Result: Read the Current Clip Coordinates Matrix

Name	Address	Attribute	Initial Value
CLIPMTX_RESULT_x (x=0-15)	0x04000640, 0x04000644, 0x04000648, 0x0400064C, 0x04000650, 0x04000654, 0x04000658, 0x0400065C, 0x04000660, 0x04000664, 0x04000668, 0x0400066C, 0x04000670, 0x04000674, 0x04000678, 0x0400067C	R	0x00000000

31	30	24	23	16	15	12	11	8	7	0
S	INTEGER_m[x]							DECIMAL_m[x]		
Sign	Integer part							Decimal part		

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- m[x] (x = 0–15) : The elements of the current clip coordinates matrix

$$CurrentClipCoordinatesMatrix = \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ m[12] & m[13] & m[14] & m[15] \end{bmatrix}$$

The current clip coordinates matrix (position coordinate matrix and projection matrix) can be read.

If you want to read the current projection matrix, make the current position coordinates matrix a unit matrix and read this register.

If you want to read the current position coordinates matrix, make the current projection matrix a unit matrix and read this register.

To safely read these matrices, confirmation that the Geometry Engine is stopped must occur before reading.

#### VectorMatrix: Read the Current Directional Vector Matrix

Name	Address	Attribute	Initial Value
VECMTX_RESULT_x (x = 0-8)	0x04000680, 0x04000684, 0x04000688, 0x0400068C, 0x04000690, 0x04000694, 0x04000698, 0x0400069C, 0x040006A0	R	0x00000000

31	30	24	23	16	15	12	11	8	7	0
S	INTEGER_m[x]							DECIMAL_m[x]		
Sign	Integer part							Decimal part		

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

- m[x] (x = 0 - 8) : The elements of the current directional vector matrix

$$CurrentDirectionalVectorMatrix = \begin{bmatrix} m[0] & m[1] & m[2] \\ m[3] & m[4] & m[5] \\ m[6] & m[7] & m[8] \end{bmatrix}$$

**Note:** To safely read this matrix, first confirm that the Geometry Engine is stopped.

## 6.2.10 Light

NITRO supports only parallel light sources.

### LightVector: Set the Light's Directional Vector

Name: LIGHT\_VECTOR

Address: 0x040004C8

Attribute: W

Command Code: 0x32

31	30	29	28	24	23	20	19	18	16	15	10	9	8	7	0	
LNUM	SZ	DECIMAL_Z					SY	DECIMAL_Y					SX	DECIMAL_X		
Light	Directional vector's Z component					Directional vector's Y component					Directional vector's X component					

Signed fixed-point number (sign + 9-bit fractional part)

- LNUM[d31–d30] : Light number  
0–3
- X, Y, Z[d29–d20], [d19–d10], [d09–d00] : Directional vector

Coordinate transformation with the directional vector matrix is performed after the settings are made.

The hardware does not perform vector normalization, so set the unit vector.

### LightColor: Set the Light Color

Name: LIGHT\_COLOR

Address: 0x040004CC

Attribute: W

Command Code: 0x33

31	30	24	23	16	15	14	10	9	8	7	5	4	0
LNUM													
Light													

- LNUM[d31–d30] : Light number  
0–3
- [d14–d00] : Light color

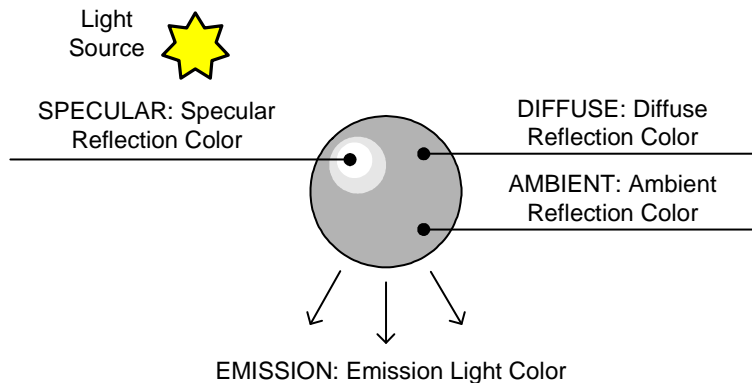
Although OpenGL has light color parameters for diffuse, specular, and ambient light, for NITRO this has been simplified to a single parameter.

## 6.2.11 Material

For objects, the appearance of the texture differs depending on the material on the surface of the object and the environment in which the object sits.

As shown in Figure 6-15, lighting (the illumination process) uses four material colors (specular reflection color, diffuse reflection color, ambient reflection color and emission light color) to express the texture of a model.

**Figure 6-15 : Material Color Schematic**



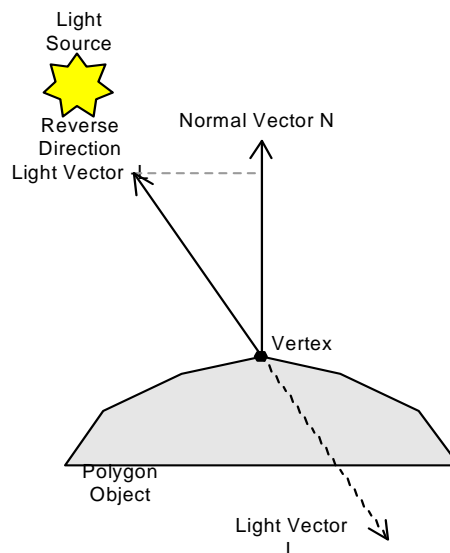
### Diffuse reflection color

This is the color of the object when it is illuminated by the light. Consider this the basic color of the object.

The diffuse reflection color is defined to reflect evenly in all directions, so it is not influenced by the position of the eye point. As shown in Figure 6-16, it is, however, influenced by the color and the direction of the light and by the normal of the polygon.

The only influence it has is on the color of the parts of objects that are directly illuminated by the light.

**Figure 6-16 : Directional Vector Relational Diagram (Diffuse Reflection Color)**



**Normal vector N:** The unit normal vector of the vertex. (Set with the Normal command)

**Light vector L:** Normalized vector of parallel light source. (Set with the LightVector command)

### Ambient reflection color

This is the color of the object when it is illuminated by ambient light.

Objects are illuminated not only by direct light, but also by light reflecting off of other objects.

This reflected light is called ambient light when it is defined to exist uniformly in the entire scene. Since ambient light exists uniformly in the entire scene, it influences the color of the entire object.

Diffuse reflection color has strong influence of parts of the object that are illuminated by direct light, but ambient reflection color has the predominant influence on parts that are shaded.

### Specular reflection color

This is the glossy color of the object when it is illuminated by light. This glossiness is called specular highlight.

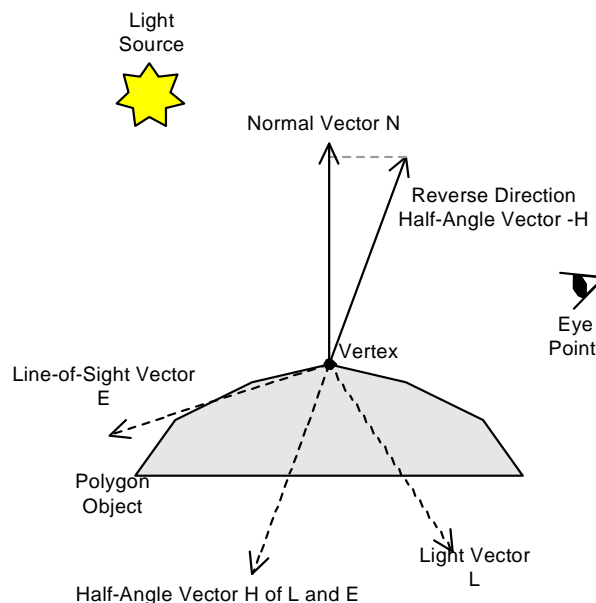
In optical terms, specular highlight is the reflected light of the light source. Accordingly, the part of the object where light strikes and reflects straight back to the eye point is the brightest.

Specular highlight is influenced by the color and direction of light, the normal of the polygon and the position of the eye point. (See Figure 6-17.)

When the eye point shifts the specular highlight moves.

It only has influence on the color of the parts of objects that are directly illuminated by the light, and this influence corresponds to the eye point.

**Figure 6-17 : Directional Vector Relational Diagram (Specular Reflection Color)**



**Normal vector N:** The unit normal vector for the vertex. (Set with the Normal command.)

**Line-of-sight vector E:** Normalized vector from eye point toward vertex. Taken to be the same as the negative direction of the z axis in the View coordinate system.

**Light vector L:** Normalized vector of parallel light source. (Set with the LightVector command.)

**Half-angle vector H of L and E:** Normalized vector of the sum of the line-of-sight vector and the light vector.

When NITRO performs the calculation for specular reflection shininess, the line-of-sight vector is taken to be the same as the negative direction of the z axis in the View coordinate system, and it is assumed that both the light vector and the normal vector will be transformed into the View coordinate system.

For this reason, when the View matrix (the LookAt matrix) is applied to the projection matrix, the coordinate system for the light vector and normal vector differ from the coordinate system for the line-of-sight vector after the transformation, and the specular reflection result is abnormal.

Accordingly, when the specular reflection color is set to any value other than black (0), the Matrix mode must be set to the Position & Vector Simultaneous Set mode, and the rotation component of the view matrix (the LookAt matrix) must be reflected on the directional vector matrix.

When the specular reflection color is set to black, the view matrix can be applied to the projection matrix because diffusion reflection does not depend on the eye point. In short, you can set the model matrix for the position coordinates matrix, and the combination of the view matrix and the projection matrix for the projection matrix.

### **Emission light color**

This is the color of the light that is emitted from the object itself.

Note that this is not treated as light, so it does not illuminate other objects (i.e., it does not influence the color of other objects).

To achieve this result, you need to create a light source that is the same color as the emission light color and place it at the same position as the object that you want emitting light.

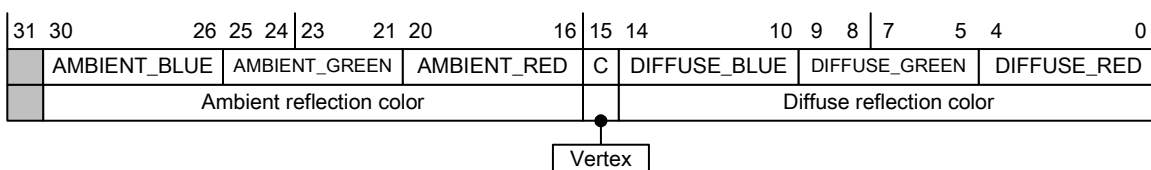
### **MaterialColor0: Set the Material's Diffuse Reflection Color and Ambient Reflection Color**

Name: DIF\_AMB

Address: 0x040004C0

Attribute: W

Command Code: 0x30



- C[d15]: Vertex color set flag

0	Does not set vertex color
1	Sets the diffuse reflection color as the vertex color

If diffuse reflection color has been set for the vertex color, it remains valid until the next time the Color, Normal or MaterialColor0 (vertex color set flag) command is issued and the current vertex color is updated.

Because the vertex color is handled as bits R:G:B = 6:6:6 in the Rendering Engine, the diffuse reflection color is applied to the upper five bits. When diffuse reflection color is 0, the lower 1 bit is 0, and when the diffuse reflection color is nonzero, the lower 1 bit is 1.

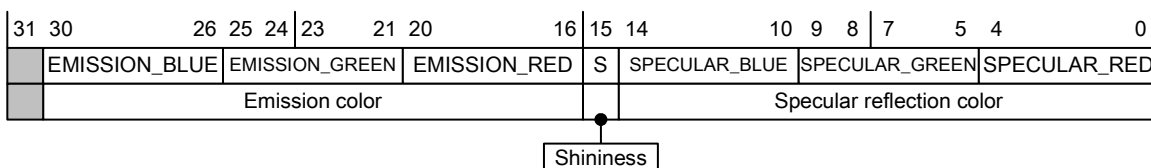
### **MaterialColor1: Set the Material's Specular Reflection Color and Emission Color**

Name: SPE\_EMI

Address: 0x040004C4

Attribute: W

Command Code: 0x31



- S[d15] : Specular reflection shininess table - enable flag

0	Disable
1	Enable

**Shininess: Set the Specular Reflection Shininess Table**

Name: SHININESS

Address: 0x0400004D

Attribute: W

Command Code: 0x34

31	24	23	16	15	8	7	0
SHININESS_n (n=4x+3)		SHININESS_n (n=4x+2)		SHININESS_n (n=4x+1)		SHININESS_n (n=4x+0)	
Shininess when Is=n		Shininess when Is=n		Shininess when Is=n		Shininess when Is=n	

Unsigned fixed point decimal (8-bit fractional part)

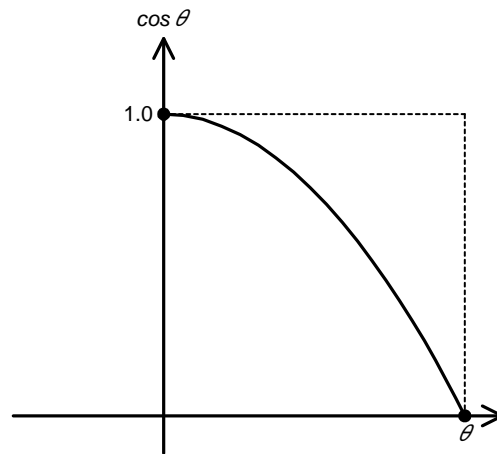
Sets the 8-bit x 128 table for converting the shininess of the specular reflection.

If the specular reflection shininess table-enable flag was set to 1 by the just-issued MaterialColor1 command, the Geometry Engine looks up the table based on the upper 7 bits of  $I_s$ —the result of the specular calculation—and converts the shininess of the specular reflection.

This table can be used to adjust the brightness of the specular reflection. (See the ["6.2.11.1 Lighting \(Illumination Process\)"](#) on page 194 for the computation formula.)

By rewriting the specular reflection shininess table, you can display polygons having a number of different specular reflection effects inside a single scene.

The specular reflection shininess calculation result  $I_s$  is obtained from the inner product of the vectors, so as Figure 6-18 shows, it is less precise near the center of the luster and more precise farther away ( $A \cdot B = |A| |B| \cos \theta$ ). See the ["6.2.11.1 Lighting \(Illumination Process\)"](#) on page 194 for the  $I_s$  calculation.

**Figure 6-18 : Specular Reflection Shininess****Technique**

You can achieve special lighting effects by setting non-consecutive values for the specular reflection shininess table.

### 6.2.11.1 Lighting (Illumination Process)

Lighting (the illumination process) is conducted when the Normal command is issued, and the results of the calculations are used for the vertex color.

The computations shown below are done on each color component (R, G, B).

#### Lighting Formulas for Various Material Colors

Material Color	Lighting Formulas
<b>Diffuse Reflection Color</b>	$I_d = \max[0, -L \cdot N]$ $D = I_d * \text{light} * \text{diffuse\_material}$
<b>Ambient Reflection Color</b>	$A = \text{light} * \text{ambient\_material}$
<b>Specular Reflection Color</b>	$I_s = \max[0, \cos^2 \theta]$  (When the specular reflection shininess table is disabled) $S = I_s * \text{light} * \text{specular\_material}$  (When the specular reflection shininess table is enabled) $S = \text{shininess\_table}[I_s] * \text{light} * \text{specular\_material}$
<b>Emission Color</b>	$E = \text{emission\_material}$

L : The light's directional vector

N : Normal vector

H : The vector that is half the sum of L (the light's directional vector) and the line-of-sight vector (the vector that points in the negative direction of the Z axis.) This is called a *half-angle vector* because it indicates the direction halfway between the L and line-of-sight vectors.

$\theta$  : The angle between the vector (-H) and the vector (N)

$I_d$  : Diffuse reflection shininess

$I_s$  : Specular reflection shininess

light : Light color

diffuse\_material : Material's diffuse reflection color

ambient\_material : Material's ambient reflection color

specular\_material : Material's specular reflection color

emission\_material : Material's emission color



**Vertex color expressions**

The ultimate vertex color is calculated with the following expression using the results of the lighting calculations conducted on each material color.

$$C = \sum_{i=0}^3 [Di + Ai + Si] + E$$

C: Vertex color

Di : Diffuse reflection color for light *i*

Ai : Ambient reflection color for light *i*

Si : Specular reflection color for light *i*

E : The color of self-emitted light

When light *i* is disabled, the corresponding color components (Di, Ai, Si) are not calculated.

The greater the number of lights that are enabled, the greater the load of the vertex color computations (that is, the greater the load of the Normal command). For this reason, be careful not to enable any more lights than are needed.

**Vertex color when lighting is OFF**

Even when lighting is OFF, the vertex color is calculated using the above expressions when the Normal command is issued. The result in this case is that the vertex color is set to the emission color.

## 6.2.12 Polygon Attributes

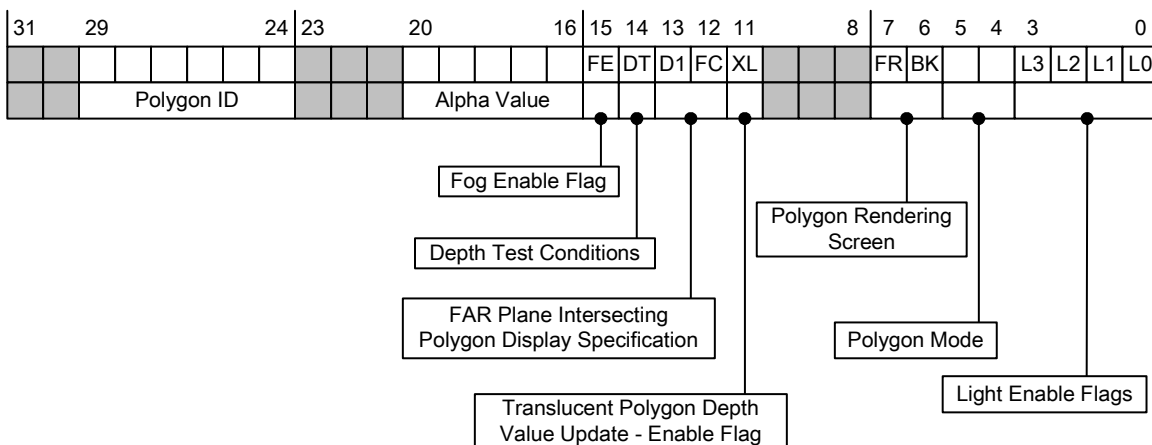
### PolygonAttr: Set the Polygon-Related Attribute Values

Name: POLYGON\_ATTR

Address: 0x040004A4

Attribute: W

Command Code: 0x29



- [d29–d24] : Polygon ID

The polygon ID is stored in separate attribute buffers for opaque polygons and translucent polygons when the polygon is being rendered by the Rendering Engine. The stored polygon ID is used when rendering translucent polygons and shadow polygons and when edge-marking. For details, see ["6.3.4 Rasterizing"](#) on page 235.

- [d20–d16] : alpha value

1–31	Polygon's opaqueness
0	Wire frame display

The polygon is called a *translucent polygon* when  $1 \leq \alpha \leq 30$  and an *opaque polygon* when  $\alpha = 31$ . When  $\alpha = 0$ , the display becomes a wireframe display and the original meaning of  $\alpha$  is lost.

- FE[d15] : Fog-enable flag

When fog is enabled, the Rendering Engine performs fog blending.

To learn about fog blending, see ["6.3.9 Fog Blending"](#) on page 260.

0	Disable
1	Enable

- DT[d14] : Depth test conditions

When set to 1, another polygon can be pasted on a polygon that has already been rendered (decals polygon).

0	Rendering when the fragment's depth value is smaller than the depth buffer's depth value.
1	Rendering when the fragment's depth value is equal to the depth buffer's depth value.

- [d13–d12] : Polygon display specification

- D1[d13] : 1-dot polygon rendering specification

This allows control of whether 1-dot polygons are passed to the rendering engine. A 1-dot polygon is a polygon where the coordinates (x, y) of all the vertices are integrated into a single coordinate as the results of geometry engine calculations.

<b>0</b>	Does not render if becomes a 1-dot polygon
<b>1</b>	Renders even if becomes 1-dot polygon

When set to 1, 1-dot polygons are always written to polygon list RAM and vertex RAM.

When this is set to 0, the depth value of the 1-dot polygon controls whether to write the polygon to polygon list RAM and vertex RAM or to discard it.

Set the display boundary depth value of 1-dot polygons with the Disp1DotDepth register.

- FC[d12] : Far plane intersecting polygon display specification

<b>0</b>	Deletes if intersects the far plane
<b>1</b>	Clips if intersects the far plane

Note that clipping on the far plane increases the load on the Geometry Engine.

- XL[d11] : Translucent polygon depth-value update enable flag

Select whether to update the depth buffer when rendering a polygon with an  $\alpha$  value of 1–30.

When this is set to 1, sometimes you can improve on a situation where too much fog is applied in regions where translucent polygons are being rendered. However, you need to be careful because sometimes the edge-marking of background is not rendered correctly.

<b>0</b>	Does not update the depth buffer when rendering translucent polygons
<b>1</b>	Updates the depth buffer when rendering translucent polygons

- [d07–d06] : Polygon rendering screen specification

The surface is the plane tracing the vertices counterclockwise.

- FR[d07] : Render front surface

<b>0</b>	Disable
<b>1</b>	Enable

- BK[d06] : Render back surface

<b>0</b>	Disable
<b>1</b>	Enable

If the specified screen is in the screen being displayed when the rendering specification is disabled, this polygon will not be included in the List RAM.

For quadrilateral polygons, if any of the first three vertices share the same coordinates, duplication will be detected by the hardware and the polygon is displayed as usual without regard to front or back. Furthermore, when the first three vertices do not overlap but are in a straight line, the straight line is not preserved due to a problem with precision in internal calculations. In this case, the surface may be determined to be the front or back according to the camera state. Use the following procedure to avoid this problem.

- Change the order the vertices are sent
- Have the second coordinate value be the same as the first or third coordinate value (Of the first three vertices, set 1 and 2 or 2 and 3 to the same value)
- Separate into triangles

When rendering a line segment in which polygon vertex coordinates overlap, front/back determination is impossible, and therefore it is always rendered, regardless of this flag's setting.

- PM[d05–d04] : Polygon mode

Modulation mode and decal mode are ways of blending texture color and fragment color.

Toon / Highlight shading is a way to transform with the fragment color table.

Shadow polygon is a feature for applying shadow using the stencil buffer.

For details, see the respective parts in the Rendering Engine ["6.3.1 Overview"](#) on page 226.

<b>00</b>	Modulation mode
<b>01</b>	Decal mode
<b>10</b>	Toon / Highlight shading
<b>11</b>	Shadow polygon

### **About Toon/Highlight Shading**

Toon and Highlight Shading share use of the same table.

Use the DISP3DCN register to choose either Toon or Highlight.

The setting written to the DISP3DCN register becomes valid when the frame switches, so Toon and Highlight Shading cannot be mixed in the same drawing frame.

- L3–L0[d03–d00] : Light enable flags

These are separate flags for setting lights 0–3.

<b>0</b>	Disable (light off)
<b>1</b>	Enable (light on)

**Where to issue the PolygonAttr command**

The values set with the PolygonAttr command become valid when the Begin command is issued. The values are subsequently used as vertex attributes.

Do not issue the PolygonAttr command between the Begin command and the End command.

Simply setting the light enable flag to enabled in the Begin command does not affect the vertex color.

The vertex color is first affected when lighting (lighting process) is performed with the Normal command after the light enable flag setting is enabled in the Begin command.

**Disp1DotDepth: 1-Dot Polygon Display Boundary Depth Value Register**

Name: DISP\_1DOT\_DEPtH

Address: 0x04000610

Attribute: W

Initial value: 0x7FFF

15	14	8	7	0
INTEGER_W				DECIMAL_W
W Coordinate				

Fixed-point number (12-bit integer + 3-bit fractional part)

- W coordinates [d14–d00] : Depth value

When the PolygonAttr command's "1-dot polygon rendering specification flag" is 0, the Geometry Engine references this register for use as described below:

When the X and Y coordinates of all polygon vertices are transformed into BG screen coordinates within a range of 1 dot or less, if the smallest W value (the depth value) is larger than this register's setting value, polygon data is not written to Polygon List RAM or Vertex RAM (and is not displayed as a result).

This W value is referenced even during Z buffering.

## 6.2.13 Polygons

### BEGIN\_VTXS: Declare the Start of the Vertex List

Name: BEGIN\_VTXS

Address: 0x04000500

Attribute: W

Command Code: 0x40

31								24	23								16	15								8	7								1	0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																												

- TYPE[d01–d00] : Primitive type

00	Triangle
01	Quadrilateral
10	Triangle Strips
11	Quadrilateral Strips

Polygon strips share vertices, so they consume less Vertex RAM than independent polygons of the same shape.

The table below shows the relation between drawing and the order in which vertices are issued by the Vertex command for different types of primitives.

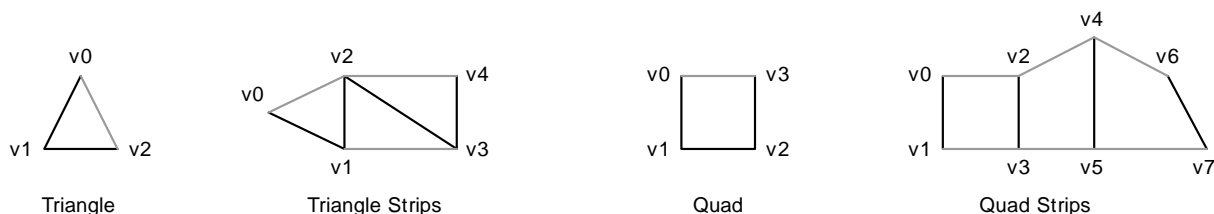
### Relation between Drawing and the Order in which the Vertex Command Issues Vertices

<b>Triangle Polygon</b>	A series of triangles are drawn starting with vertices “v0, v1, and v2” and then vertices “v3, v4, and v5.”
<b>Quadrilateral Polygon</b>	A series of quadrilaterals are drawn starting with vertices “v0, v1, v2, and v3” and then vertices “v4, v5, v6, and v7.”
<b>Triangle Strips</b>	A series of triangles are drawn starting with vertices “v0, v1, and v2,” “v2, v1, and v3” and then “v2, v3, and v4.” This is the order so that the triangles are drawn in the same direction on both sides of the surface. (See Figure 6-15.)
<b>Quadrilateral Strips</b>	A series of quadrilaterals are drawn starting with vertices “v0, v1, v3, and v2,” “v2, v3, v5, and v4” and then “v4, v5, v7, and v6.” This is the order so that the quadrilaterals are drawn in the same direction on both sides of the surface. (See Figure 6-15.)

### Defining the primitive's front surface

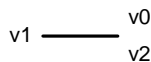
The surface is described counterclockwise (v0, v1, v2,... : in order of issued Vertex-group commands).

Figure 6-19 : Order in which the Vertex commands issues vertices



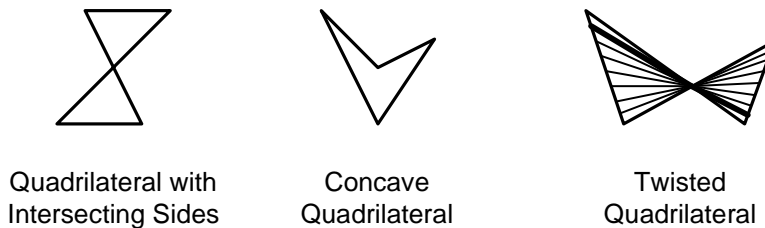
When you want to draw line segments, set the same value for neighboring vertices in the above primitives. However, because it is not possible to determine the front or back of the line segments, they are always rendered, even if the polygon attributes specify to disable display. As Figure 6-20 shows, the anti-aliasing and edge marking features (see the Rendering Engine ["6.3.1 Overview"](#) on page 226) work on line segments as well.

**Figure 6-20: Line segment using sides from a triangle**



When you draw quadrilateral polygons in the shapes shown in Figure 6-21, sometimes the results are not as intended. Be sure to set the vertices so quadrilateral polygons are not drawn in these shapes.

**Figure 6-21: Quadrilateral Polygon shapes that yield unintended shapes**



### End: Declare the End of the Vertex List

Name: END\_VTXS

Address: 0x04000504

Attribute: W

Command Code: 0x41

[illegible]

Make certain to issue Begin and End commands in pairs.

## Color: Directly Set the Vertex Color

Name: COLOR

Address: 0x04000480

Attribute: W

Command Code: 0x20

[illegible]

The vertex color remains valid until the current vertex color is updated by the next Color command, Normal command or MaterialColor0 (vertex color set flag) command.

Thus, multiple vertices can share the same vertex color.

Because the vertex color is handled as bits R:G:B = 6:6:6 in the Rendering Engine, the diffuse reflection color is applied to the upper five bits. When set color value is 0, the lower 1 bit is 0, and when it is nonzero, the lower 1 bit is 1.

The `Color` command is usually issued between the `Begin` and `End` commands, but it can also be issued before the `Begin` command.

<b>Command string Example 1</b>	Begin→Color→Vertex→Vertex→Vertex→End
<b>Command string Example 2</b>	Color→Begin→Vertex→Vertex→Vertex→End→Begin→Vertex→Vertex→Vertex→End

**Normal: Set the Normal Vector**

Name: NORMAL

Address: 0x04000484

Attribute: W

Command Code: 0x21

31	30	29	28	24	23	20	19	18	16	15	10	9	8	7	0
		S	NZ				S	NY				S	NX		
		Normal vector's Z component				Normal vector's Y component				Normal vector's X component					

Signed fixed-point number (sign + 9-bit fractional part)

Lighting (illumination process) is performed only when the Normal command is executed.

Accordingly, you need to reissue the Normal command after you switch lighting On/Of or change the light or material parameters in order for the change to be reflected in the vertex color. (See note.)

Further, the hardware does not normalize vectors, so you need to set the unit vector.

The vertex color obtained with the lighting process remains valid until the current vertex color is updated by the next Color command, Normal command or MaterialColor0 (vertex color set flag) command.

Thus, in actuality multiple vertices can share the same normal vector.

**Note:** To turn light on or off, after setting in the PolygonAttr command, first enable the set value with the Begin command, and then issue the Normal command.**Where to issue the Normal command**

The Normal command is usually issued between the Begin and End commands, but it can also be issued before the Begin command.

<b>Command string Example 1</b>	Begin→Normal→Vertex→Vertex→Vertex→End
<b>Command string Example 2</b>	Normal→Begin→Vertex→Vertex→Vertex→End→Begin→Vertex→Vertex→Vertex→End

**VTX\_16: Set the Vertex Coordinates**

Name: VTX\_16

Address: 0x0400048C

Attribute: W

Command Code: 0x23

31	30	28	27	24	23	16	15	14	12	11	8	7	0
SY	INT_Y		DECIMAL_Y				SX	INT_X		DECIMAL_X			
Y Coordinate							X Coordinate						

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

31	24	23	16	15	14	12	11	8	7	0



## VTX\_10: Set the Vertex Coordinates

Name: VTX\_10

Address: 0x04000490

Attribute: W

Command Code: 0x24

31	29	28	26	25	24	23	20	19	18	16	15	10	9	8	7	5	0		
		SZ	INT_Z		DECIMAL_Z			SY	INT_Y		DECIMAL_Y			SX	INT_X		DECIMAL_X		
		Z Coordinate						Y Coordinate						X Coordinate					

Signed fixed-point number (sign + 3-bit integer + 6-bit fractional part)

### VertexXY: Set the Vertex XY Coordinates (for Z Coordinate, Use the Last-Set Data)

Name: VTX\_XY

Address: 0x04000494

Attribute: W

Command Code: 0x25

31	30	28	27	24	23	16	15	14	12	11	8	7	0
SY	INT_Y		DECIMAL_Y				SX	INT_X		DECIMAL_X			
Y Coordinate							X Coordinate						

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

### VertexXZ: Set the Vertex XZ Coordinates (for Y Coordinate, Use the Last-Set Data)

Name: VTX\_XZ

Address: 0x04000498

Attribute: W

Command Code: 0x26

31	30	28	27	24	23	16	15	14	12	11	8	7	0
SZ		INT_Z		DECIMAL_Z				SX		INT_X		DECIMAL_X	
Z Coordinate						X Coordinate							

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

### VertexYZ: Set the Vertex YZ Coordinates (for X Coordinate, Use the Last-Set Data)

Name: VTX\_YZ

Address: 0x0400049C

Attribute: W

Command Code: 0x27

31	30	28	27	24	23	16	15	14	12	11	8	7	0
SZ		INT_Z		DECIMAL_Z				SY		INT_Y		DECIMAL_Y	
Z Coordinate						Y Coordinate							

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

**VertexDiff: Set the Difference Value of the Last-Set Data for Vertex Coordinates**

Name: VTX\_DIFF

Address: 0x040004A0

Attribute: W

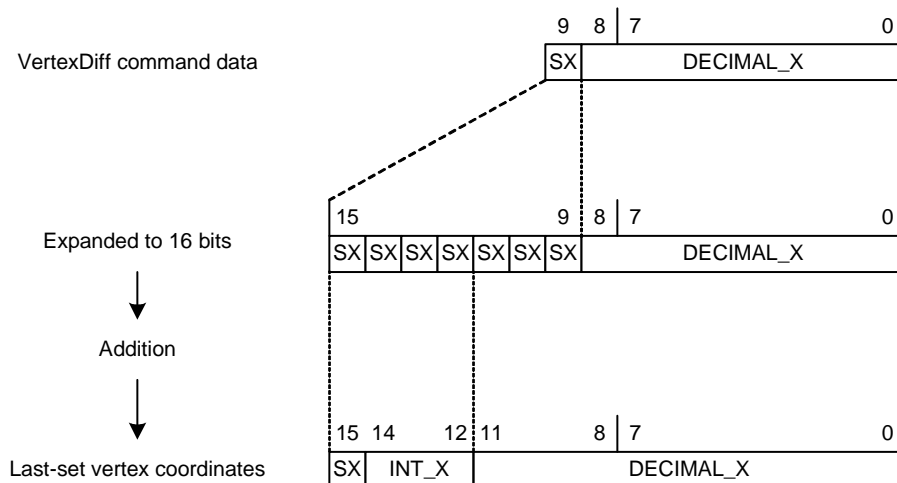
Command Code: 0x28

31	29	28	24	23	20	19	18	16	15	10	9	8	7	0
		SZ	DECIMAL_Z				SY	DECIMAL_Y				SX	DECIMAL_X	
		Z Coordinate				Y Coordinate				X Coordinate				

Signed fixed-point number (sign + 9-bit fractional part)

The value after adding to the last-set vertex value is stored as the 16-bit vertex coordinates.

The VertexDiff command data is sign-extended to 16 bits and added to the prior-set vertex coordinate. The vertex coordinates (the Vertex-group command's data) are 16 bits in size (sign + 3-bit integer + 12-bit fractional part), so the data of the VertexDiff command corresponds to the 4th to 12th places of the fractional part of the vertex coordinates (see Figure 6-22.)

**Note:** Use caution because an overflow can occur during the addition process.**Figure 6-22: The Process for Adding the X Coordinate****Items common to all Vertex commands**

When Vertex commands are issued, the vertex data that have been transformed into BG screen coordinates are stored in Vertex RAM. Further, polygon data is stored to Polygon List RAM when the data for the number of vertices comprising the polygon are processed.

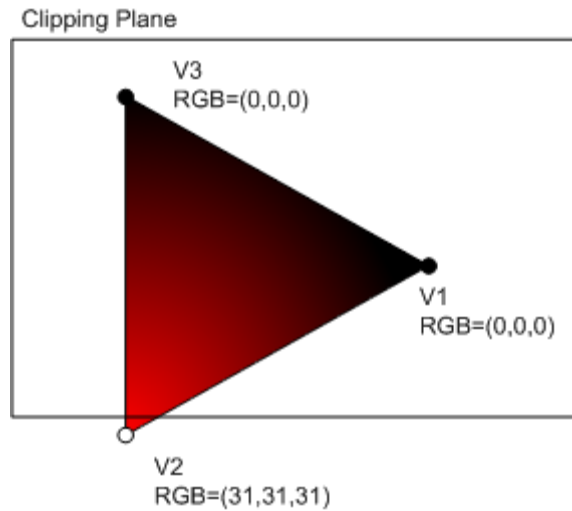
**Note:** Be sure to issue Vertex commands between the Begin command and the End command and that there are not too many or too few vertices in the specified primitives.**Cautions for polygons during clipping**

When a polygon is clipped, the G value, B value, or both on the clipping plane polygon will occasionally change to 0, causing color distortion. Figure 6-23 represents the state in which this has occurred (the G value and the B value have both changed to 0). When calculating the color of the vertex newly created through clipping, the color value may become a value higher than 31. When this happens, the last 5 bits of the value truncated to 32 will become the final color, incorrectly making it 0. This does not occur for the R value because the calculation accuracy is higher than the other two. This results in a reddish display.

This issue may be avoided by using the following methods

- Set the polygon scaling small to assure high geometry calculation accuracy.
- Shorten the vertex interval of the polygons, or pull the vertex away from the clipping plane to reduce the effect of the errors stemming from low calculation accuracy.
- If the vertex color is directly configured through a modeling software, set the vertex color to be (R, G, B) = (31, 30, 30) or smaller.
- If the vertex color is not directly configured, adjust the material or light color so that the calculated vertex will be (R, G, B) = (31, 30, 30) or smaller.

**Figure 6-23: Polygon Clipping Color Distortion**



## 6.2.14 Texture Mapping

### TexCoord: Set Texture Coordinates

Name: TEXCOORD

Address: 0x04000488

Attribute: W

Command Code: 0x22

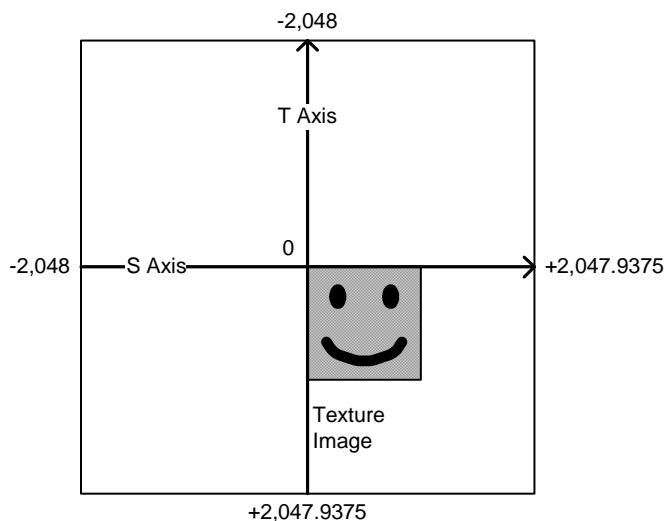
31	30	24	23	16	15	14	8	7	0
ST	INTEGER_T			DECIMAL_T	SS	INTEGER_S			DECIMAL_S
T Coordinate					S Coordinate				

Signed fixed-point number (sign + 11-bit integer + 4-bit fractional part)

- TEX\_T, TEX\_S[d31–d16], [d15–d00]: Texture coordinates

As Figure 6-24 shows, the texture coordinates set the coordinates in texture image space, treating the texel size as 1.0 (4-bit fractional part).

**Figure 6-24: Texture Image Space (for an Image of 1,024x1,024 Texels)**



The texture coordinates remain valid until the next TexCoord command resets the current texture coordinates.

Because of this, the same texture coordinates can be shared by multiple vertices.

### Where to issue the TexCoord command

The TexCoord command is normally issued between the Begin command and the End command, but it can also be issued before the Begin command.

<b>Command String Example 1</b>	Begin→TexCoord→Vertex→Vertex→Vertex→End
<b>Command String Example 2</b>	TexCoord→Begin→Vertex→Vertex→Vertex→End→Begin→Vertex→Vertex→Vertex→End

When texture mapping, the Geometry Engine works faster if you issue commands in the following order: TexCoord→Normal→Vertex.

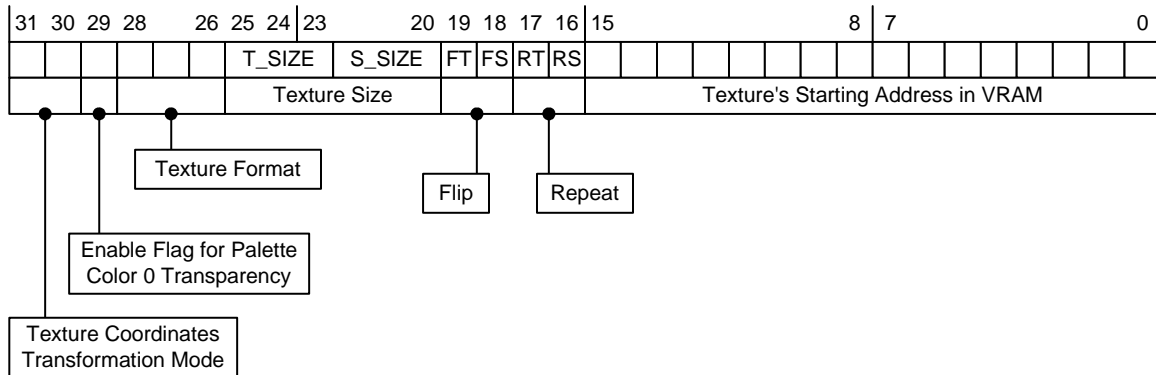
**TexImageParam: Setting the Texture Parameters**

Name: TEXIMAGE\_PARAM

Address: 0x040004A8

Attribute: W

Command Code: 0x2a



- TGEN[d31–d30] : Texture coordinate transformation mode

<b>00</b>	Do not transform texture coordinates
<b>01</b>	TexCoord source
<b>10</b>	Normal source
<b>11</b>	Vertex source

- TR[d29] : Enable flag for the palette's color0 transparency

When using transparent texels with 4-, 16-, and 256-color palette textures, set this bit to 1 so the palette's color0 can be referenced for the transparent color.

<b>0</b>	Enable the palette's color0 setting
<b>1</b>	Make appear transparent, regardless of the palette color0 setting value

- TEXFMT[d28–d26] : Texture format

<b>0</b>	No texture
<b>1</b>	A3I5 translucent texture
<b>2</b>	4-color palette texture
<b>3</b>	16-color palette texture
<b>4</b>	256-color palette texture
<b>5</b>	4x4 texel compressed texture
<b>6</b>	A5I3 translucent texture
<b>7</b>	Direct texture

- [d25–d20] : Texture size

- T\_SIZE, S\_SIZE

Selects texture size of 8 x 8 to 1,024 x 1,024.

<b>0</b>	8 texels
<b>1</b>	16 texels
<b>2</b>	32 texels
<b>3</b>	64 texels
<b>4</b>	128 texels
<b>5</b>	256 texels
<b>6</b>	512 texels
<b>7</b>	1,024 texels

- [d19–d18] : Flip

Specifies whether to flip the texture image up/down and/or left/right for mapping when the texture coordinates pertain to a region beyond the texture size. (See Figure 6-25 and Figure 6-26.) The flip setting is valid only when Repeat has been specified.

- FT[d19] : Flip in direction of T coordinates

<b>0</b>	Do not flip
<b>1</b>	Flip

- FS[d18] : Flip in direction of S coordinates

<b>0</b>	Do not flip
<b>1</b>	Flip

- [d17–d16] : Repeat

Specifies whether to repeatedly map the texture image when the texture coordinates pertain to a region beyond the texture size.

- RT[d17] : Repeat in the direction of the T coordinates

<b>0</b>	Do not repeat
<b>1</b>	Repeat

- RS[d16] : Repeat in the direction of the S coordinates

<b>0</b>	Do not repeat
<b>1</b>	Repeat

- TEX\_ADDR[d15–d00] : Texture's starting address in VRAM

The system references the 3-bit left-shift of the texture's starting address in VRAM.

**Where to issue the TexImageParam command**

The `TexImageParam` command is normally issued before the `Begin` command, but it can also be issued between the `Begin` and `End` commands. By issuing it during the `Begin-End` interval you can set different texture parameters for every polygon inside the `Begin-End` command string.

**Note:** A problem causes the previous polygon texture attributes to be overwritten with the parameters passed by the `TexImageParam` command, according to the process status of the geometry engine.

The following examples show the positions in which to issue the `TexImageParam` command:

- Outside the `Begin-End` interval:

```
TexImageParam;
Begin;
    TexCoord;
    Vertex;
    ...
End;
```

- Within the `Begin-End` interval with quadrilateral polygons

1. When the command structure for the polygon that has changed texture after the second polygon changes from `TexCoord` to `Vertex`:

```
Begin;
    TexImageParam;
    First Polygon;
    TexImageParam;
    Normal;           Send the Normal command as a dummy command
    Second Polygon;
    ...
End;
```

2. When the command structure for the polygon that has changed texture after the second polygon changes from `TexCoord` to `Normal` to `Vertex`:

The problem can be resolved effectively by sending `0xFF` (undefined) as the dummy command, in place of the `Normal` command sent in the first example. However, when the `TexPlttBase` command is sent with the `TexImageParam` command, this problem does not occur, and there is no need to send a dummy command.

- Within the `Begin-End` interval with triangle polygons

1. When the command structure for the polygon that has changed texture after the second polygon changes from `TexCoord` to `Vertex`

Same as example 1 for quadrilateral polygons.

2. When the command structure for the polygon that has changed texture after the second polygon changes from `TexCoord` to `Normal` to `Vertex`

No problems occur.

- Within the Begin-End interval with triangle and quadrilateral polygons

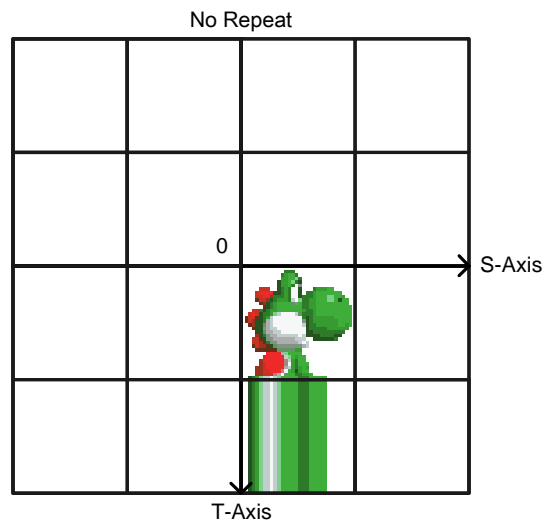
In this case, always use `End` once when changing textures. In other words, limit issuing the `TexImageParam` command to a position before the `Vertex` command is issued, as shown below.

```
Begin;
    TexImageParam;
    Vertex;
    ...
End;
```

- Texture Flip and Repeat Settings (For a Texture Image of 1,024 x 1,024 Texels)

1. When when there is no repeat

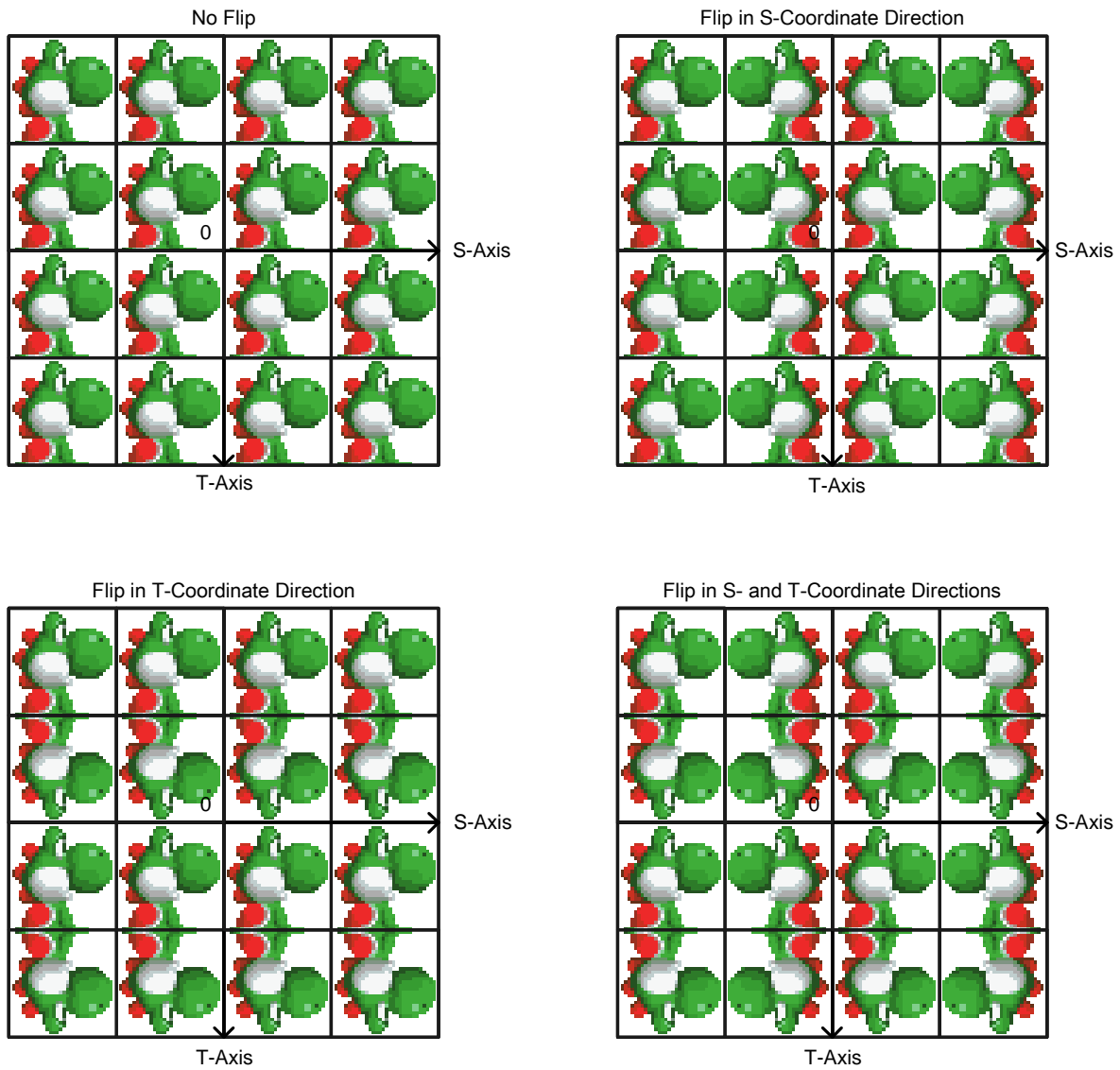
**Figure 6-25: Texture Image Space (No Repeats)**





2. Flip settings when there is a repeat

**Figure 6-26: Texture Image Space (with Repeats)**



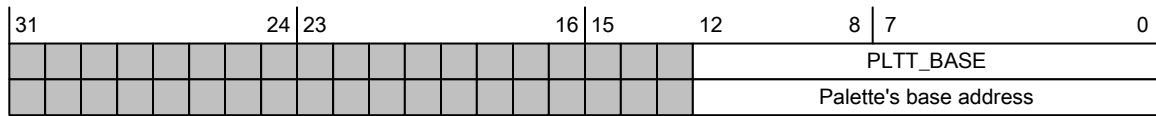
**TexPlttBase: Set Texture Palette's Base Address**

Name: TEXPLTT\_BASE

Address: 0x040004AC

Attribute: W

Command Code: 0x2b



- PLTT\_BASE[d12–d00] : Specifies the palette's base address

The system references a 2 to 4 bit left shift of the palette's base address.

The shift volume varies, depending on the texture format (as shown in "[Table 6-4 : Number of Geometry Command Run Cycles & Timing Related to Command Issue \(in Command Code Order\)](#)" on page 174).

**Table 6-5: PLTT\_BASE Values and Shift Volumes**

PLTT_BASE Value	4-color Palette Texture	16-color Palette Texture	256-color Palette Texture	4x4 texels Compressed Texture	A3I5 Texture	A5I3 Texture
0x0000	0x00000	0x00000	0x00000	0x00000	0x00000	0x00000
0x0001	0x00008	0x00010	0x00010	0x00010	0x00010	0x00010
0x0002	0x00010	0x00020	0x00020	0x00020	0x00020	0x00020
0x17FE	0x0BFF0	0x17FE0	0x17FE0	0x17FE0	0x17FE0	0x17FE0
0x17FF	0x0BFF8	0x17FF0	0x17FF0	0x17FF0	0x17FF0	0x17FF0
0x1800	0x0C000	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited
0x1FFE	0x0FFF0	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited
0x1FFF	0x0FFF8	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited	Setting prohibited

**Where to issue the TexPlttBase command**

The TexPlttBase command is normally issued before the Begin command, but it can also be issued between the Begin and End commands. By issuing it between these two commands you can set a different palette base address for every polygon between the Begin and End commands.

<b>Command String Example 1</b>	TexPlttBase→Begin→TexCoord→Vertex→TexCoord→Vertex→TexCoord→Vertex→End
<b>Command String Example 2</b>	Begin→TexPlttBase→TexCoord→Vertex→TexCoord→Vertex→TexCoord→Vertex→TexPlttBase→TexCoord→Vertex→TexCoord→Vertex→TexCoord→Vertex→End

### 6.2.14.1 Texture Coordinate Transformations

The texture coordinate transformation mode can be switched using the `TexImageParam` command.

In the three modes described in this section, the values given by the pertinent command are used as the input coordinates for the calculations.

The values after the coordinate transformation are not meant to be used, so make sure you set the texture matrix appropriately in advance.

#### TexCoord source

The texture coordinate transformation is performed using the values set by the `TexCoord` command as the input coordinates.

The coordinate transformation is executed when the `TexCoord` command is issued.

You can produce a simple texture scroll by setting a translation matrix or a rotation matrix for the texture matrix.

#### Operation Expressed in Matrix Form

$$\begin{bmatrix} S & T & R & Q \end{bmatrix} = \begin{bmatrix} S & T & \frac{1}{16} & \frac{1}{16} \end{bmatrix} \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ m[12] & m[13] & m[14] & m[15] \end{bmatrix}$$

#### Specific Expression Taking Decimal-Point Position into Account

$$S = \{m[0] \times (S \ll 12) + m[4] \times (T \ll 12) + m[8] \times (1 \ll 12) + m[12] \times (1 \ll 12)\} \gg 24$$

$$T = \{m[1] \times (S \ll 12) + m[5] \times (T \ll 12) + m[9] \times (1 \ll 12) + m[13] \times (1 \ll 12)\} \gg 24$$

#### Normal source

The texture coordinate transformation is performed using the values set by the `Normal` command as the input coordinates.

The coordinate transformation is executed when the `Normal` command is issued.

The S and T values set by the immediately-prior `TexCoord` command are used as the translation components of the texture coordinates.

You can produce spherical reflection mapping by setting in the texture matrix the result of reading the current directional vector matrix and multiplying by the scaling matrix that expands the directional vector space (-1.0 to 1.0) to 1/2 the texture size. When doing this, use the `TexCoord` command to translate the origin of the texture coordinate to the center of the spherical texture.

#### Operation Expressed in Matrix Form

$$\begin{bmatrix} S & T & R & Q \end{bmatrix} = \begin{bmatrix} N_u & N_v & N_w & 1 \end{bmatrix} \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ S & T & m[14] & m[15] \end{bmatrix}$$

### Specific Expression Taking Decimal-Point Position into Account

$$S = \{m[0] \times (Nx \ll 3) + m[4] \times (Ny \ll 3) + m[8] \times (Nz \ll 3) + (S \ll 12) \times (1 \ll 12)\} \gg 24$$

$$T = \{m[1] \times (Nx \ll 3) + m[5] \times (Ny \ll 3) + m[9] \times (Nz \ll 3) + (T \ll 12) \times (1 \ll 12)\} \gg 24$$

#### Vertex source

The texture coordinate transformation is performed using the values set by a Vertex-group command as the input coordinates. The coordinate transformation runs when a Vertex-group command is issued.

The S and T values set by the `TexCoord` command issued immediately prior are used as the translation components of the texture coordinates.

You can produce texture scrolls dependent on the View coordinates by reading the current position coordinate matrix and setting it to the texture matrix.

### Operation Expressed in Matrix Form

$$\begin{bmatrix} S & T & R & Q \end{bmatrix} = \begin{bmatrix} X & Y & Z & 1 \end{bmatrix} \begin{bmatrix} m[0] & m[1] & m[2] & m[3] \\ m[4] & m[5] & m[6] & m[7] \\ m[8] & m[9] & m[10] & m[11] \\ S & T & m[14] & m[15] \end{bmatrix}$$

### Specific Expression Taking Decimal-Point Position into Account

$$S = \{m[0] \times X + m[4] \times Y + m[8] \times Z + (S \ll 12) \times (1 \ll 12)\} \gg 24$$

$$T = \{m[1] \times X + m[5] \times Y + m[9] \times Z + (T \ll 12) \times (1 \ll 12)\} \gg 24$$

- **Decimal point positions between parameters used by texture-coordinate transformation expressions**

The parameter formats used in the expressions for calculating texture coordinate transformations are shown in the following table.

Therefore, to unify texture coordinate transformation calculations to 12-bit fractional parts, the Normal coordinates could be left-shifted by 3 bits, and the texture coordinates could be left-shifted by 8 bits before the calculation is applied.

However, as can be seen in "Specific expression taking decimal-point position into account," texture coordinates are left-shifted by 12 bits before the expression is applied, and the results that are right-shifted by 24 bits are taken as the new texture coordinates.

From this, it is natural to assume that the texture coordinate transformation is (Sign + 15-bit integer + 0-bit fractional part). That is, texture coordinate transformation calculations use units of 1/16 texel, rather than units of 1 texel.

Parameter Name	Parameter	Format
Texture Matrix	m[num] (num = 0-15)	Sign + 19-bit integer + 12-bit fractional part
Normal Coordinate	Nx, Ny, Nz	Sign + 9-bit fractional part
Vertex Coordinate	X, Y, Z	Sign + 3-bit integer + 12-bit fractional part
Texture Coordinate	S, T	Sign + 11-bit integer + 4-bit fractional part

- **Technical tip**

There may be times when you are using polygons to represent 2D graphics and you want the texels to have 1:1 correspondence with the pixels on the LCD. Because texture sampling proceeds from the upper left texel, the texture may be off by 1 texel due to such factors as polygon rotation. To prevent this from happening, use a texture-coordinate transformation to adjust the position from the position at which sampling starts. For details, see ["6.3.5.1.1 Texture Image Sampling"](#) on page 244.

- **Polygon processing cycle count**

When there are two lights or fewer, the execution cycle of the geometry engine will not vary based on the availability of the Normal command. When there are three or more lights, the execution cycle will be faster without the Normal command. Also, regardless of the number of lights, the transfer time (bus usage time) to the command FIFO will be shorter if the Normal command is not executed.

The following table values result from the fact that when the number of lights is small, the coordinate conversion will become the bottleneck even if the calculation cycle of the vertex color is short. Therefore the polygon operation will be fixed to the vertex coordinate conversion time. But when the number of lights increase, the vertex color calculation cycle will exceed this time.

Command Structure	Light Count for Triangular Polygons					Light Count for Quadrilateral Polygons				
	OFF	1ch	2ch	3ch	4ch	OFF	1ch	2ch	3ch	4ch
TexCoord -> Normal -> Vertex	28	28	28	30	33	37	37	37	40	44
Normal -> Vertex	28	28	28	28	30	37	37	37	37	40
TexCoord -> Vertex	28	28	28	28	28	37	37	37	37	37
Vertex	28	28	28	28	28	37	37	37	37	37

## 6.2.15 Tests

When the Geometry Engine executes a test-related command (such as the status flag and the resulting register value), it updates the test command result.

### BoxTest: Test if Cuboid Sits Inside View Volume

Name: BOX\_TEST

Address: 0x040005C0

Attribute: W

Command Code: 0x70

31	30	28	27	24	23	16	15	14	12	11	8	7	0	
SY	INT_Y	DECIMAL_Y						SX	INT_X	DECIMAL_X				
Y Coordinate							X Coordinate							

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

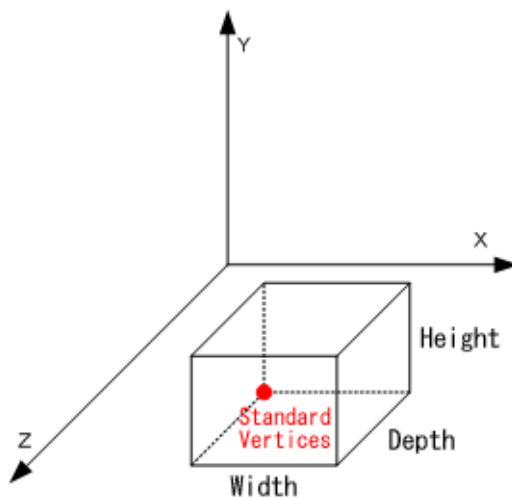
31	30	28	27	24	23	16	15	14	12	11	8	7	0	
SW	INT_W		DECIMAL_W						SZ	INT_Z		DECIMAL_Z		
Width							Z Coordinate							

31	30	28	27	24	23	16	15	14	12	11	8	7	0	
SD	INT_D	DECIMAL_D						SH	INT_H	DECIMAL_H				
Depth							Height							

Specify the box's standard vertices of the box shown in Figure 6-27 for the coordinate values.

The result of the Box test is stored in the Geometry Engine Status register (GXSTAT).

**Figure 6-27: Box to Be Tested**



- Box Test Contents

Determines whether any of the six faces of the box are not completely within the view volume.

For this reason, if the view volume is completely contained within the box, it is considered out of view.

- Conditions Required to Properly run a Box Test

Conduct the box test with both polygon attribute flags set to 1. If either of the flags is set to 0, the test results may not be correct.

Keep in mind that overflow may occur when width/height/depth is added to the reference vertices. (The result of the addition must be greater than or equal to  $-8.0$  and less than  $8.0$ .)

1. Set both of the polygon attribute flags to 1:
  - Far plane-intersecting polygon display specification
  - 1-dot polygon rendering specification
2. `Begin` command
3. `End` command
4. `BoxTest` command

**PositionTest: Set the Position Coordinates for the Tests**

Name: POS\_TEST

Address: 0x040005C4

Attribute: W

Command Code: 0x71

31	30	28	27	24	23	16	15	14	12	11	8	7	0
SY	INT_Y		DECIMAL_Y				SX	INT_X		DECIMAL_X			
Y Coordinate						X Coordinate							

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

31								24	23								16	15	14			12	11			8	7								0																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					

Coordinate transformation is performed on the position coordinates for the test by the current clip coordinate matrix.

The results of the Position test (the clip coordinates) are stored in the PositionResult register.

**VectorTest: Set the Directional Vector for the Tests**

Name: VEC\_TEST

Address: 0x040005C8

Attribute: W

Command Code: 0x72

31	29	28	24	23	20	19	18	16	15	10	9	8	7	0
		SZ	DECIMAL_Z				SY	DECIMAL_Y				SX	DECIMAL_X	
		Z Component				Y Component				X Component				

Signed fixed-point number (sign + 9-bit fractional part)

Coordinate transformation is carried out on the directional vector for the test by the current directional vector matrix.

The result of the Vector test (the directional vector in the View coordinate space) is stored in the VectorResult register.

**PositionResult: Read the PositionTest Computational Results**

Name	Address	Attribute	Initial Value
POS_RESULT_x (x=X,Y,Z,W)	0x04000620, 0x04000624, 0x04000628, 0x0400062C	R	0x00000000

31	30	24	23	16	15	12	11	8	7	0
Sx	INTEGER_x							DECIMAL_x		
Sign	Integer Part							Decimal Part		

Signed fixed-point number (sign + 19-bit integer + 12-bit fractional part)

The clip coordinates values (x, y, z, w) are stored in these registers.



**VectorResult: Read the VectorTest Computational Results**

Name	Address	Attribute	Initial Value
VEC_RESULT_x (x=X,Y,Z)	0x04000630, 0x04000632, 0x04000634	R	0x0000

15	12	11	8	7	0
Sx	INTEGER_x	DECIMAL_x			
Sign	Integer Part	Decimal Part			

Signed fixed-point number (sign + 3-bit integer + 12-bit fractional part)

The directional vector values (x, y, z) of the view coordinate space are stored in these registers.

The read-out computation results are within  $\pm 1$ , so the integer is a sign extension.

## 6.2.16 Status

You can check the status of Polygon List RAM and Vertex RAM using the previously mentioned DISP3DCNT register.

To check the status of command FIFO and matrix stacks, etc., see the GXSTAT register diagram below.

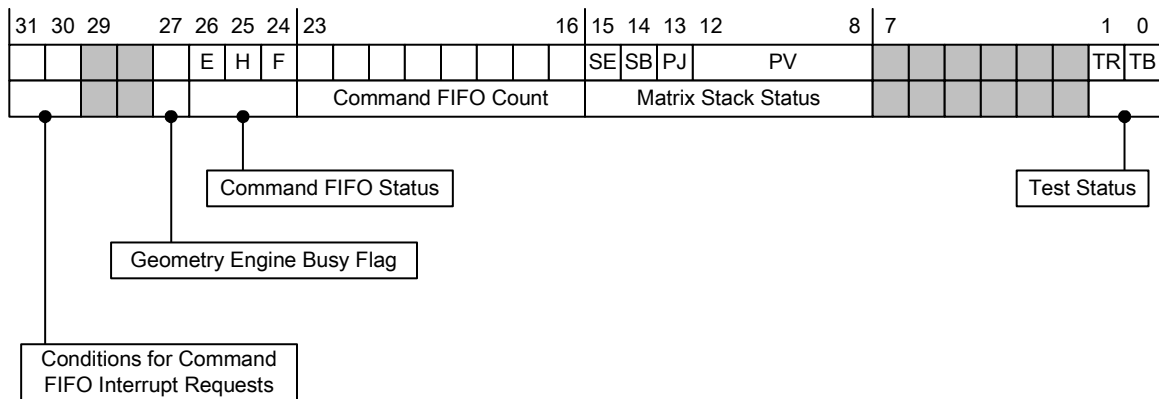
### GXSTAT: Geometry Engine Status Register

Name: GXSTAT

Address: 0x04000600

Attribute: R/W

Initial Value: 0x00000000



- FI[d31–d30] : Conditions for Command FIFO interrupt requests

00	Disable Command FIFO interrupt requests
01	Make interrupt request when Command FIFO is less than half full
10	Make interrupt request when Command FIFO is empty
11	Setting prohibited

- B[d27] : Geometry Engine busy flag

0	Geometry Engine is stopped
1	Geometry Engine is running

If commands or parameters have not been sent, the busy flag goes to the command wait / parameter wait status with the busy flag set to 0.

When commands or parameters resume, geometry processing resumes.

When a `SwapBuffers` command is issued, and no subsequent commands are issued, the geometry engine busy flag is set to 0 at the completion of a `SwapBuffers` process, which begins with a V-Blank. (It is 1 until then.)

This timing occurs 400 cycles (calculated at 33 MHz) after the V-Blank.

If, after the `SwapBuffers` command is issued, a next command is also issued, the next command is executed after the completion of the `SwapBuffers` process, which commences after the V-Blank.

- [d26–d24] : Command FIFO status
  - E[d26] : Command FIFO empty flag

0	FIFO is not empty
1	FIFO is empty

- H[d25] : Command FIFO under-half flag

0	FIFO is at least half full
1	FIFO is less than half full

- F[d24] : Command FIFO full flag

0	FIFO is not full
1	FIFO is full

- [d23–d16] : Command FIFO count value

Can reference the number of commands/amount of data currently stored in Command FIFO.

- [d15–d08] : Matrix stack status

- SE[d15] : Stack error flag

The flag is set to 1 when an overflow or underflow of the matrix stack occurs.

It can be cleared by writing 1.

0	No stack overflow or underflow
1	Stack overflow or underflow

When referencing the status error flag of matrix stack status, confirm that the `PushMatrix` and `PopMatrix` commands that have been issued have completed by first referencing the matrix stack busy flag.

- SB[d14] : Matrix stack busy flag
- When referencing PJ and PV matrix stack levels, check this flag to confirm that execution of the issued `PushMatrix` or `PopMatrix` command has completed.

0	There is no unexecuted <code>PushMatrix</code> or <code>PopMatrix</code> command.
1	The <code>PushMatrix</code> or <code>PopMatrix</code> command has been issued, and the execution is not yet completed.

- PJ[d13] : Projection matrix stack level

Can reference the current stack level (0–1)

- PV[d12–d08] : Position and Vector matrix stack level

Can reference the current stack level (0–31)

- [d01–d00] : Test status flag
  - TR[d01] : Box test result

0	All the faces that constitute the box are outside the view volume.
1	Of the six faces that constitute the box, part of one of the faces, or all are inside the view volume.

- TB[d00] : Test busy flag

Can reference the ready/busy status of each test (BoxTest, PositionTest, VectorTest).

0	Ready
1	Busy

### LISTRAM\_COUNT: Polygon List RAM Count Register

Name: LISTRAM\_COUNT

Address: 0x04000604

Attribute: R

Initial Value: 0x0000

15					11					8		7							0
Polygon-List RAM Counter																			

- [d11–d00] : Polygon List RAM counter (Maximum valid value is 0x800)

You can reference the number of opaque polygons + translucent polygons that are currently stored in Polygon List RAM.

Polygon List RAM has a capacity of 2048 polygons, so the maximum valid value is 0x800.

The polygon list RAM counter is cleared 10 system clock cycles (33.5MHz) after the V-Blank that comes immediately after the *SwapBuffers* command is issued.

### VTXRAM\_COUNT: Vertex RAM Count Register

Name: VTXRAM\_COUNT

Address: 0x04000606

Attribute: R

Initial Value: 0x0000

15					12					8		7							0
Vertex RAM Counter																			

- [d12–d00] : Vertex RAM counter (Maximum valid value is 0x1800)

You can reference the number of vertices that are currently stored in Vertex RAM.

Vertex RAM has a capacity of 6144 vertices, so the maximum valid value is 0x1800.

The vertex RAM counter is cleared 10 system clock cycles (33.5MHz) after the V-Blank that comes immediately after the *SwapBuffers* command is issued.

### 6.2.16.1 Data Storage Capacity of Polygon List RAM and Vertex RAM

#### 1. Polygon List RAM

The 52 KB of Polygon List RAM is divided into an ORDER area of 12 KB, followed by a POLYGON area of 40 KB. The number of polygons that can be stored in these two areas of Polygon List RAM does not change when polygons are connected, but because connected polygons share vertices, the same number of polygons consume less memory if they are connected. This is an effective way to economize on Vertex RAM.

No matter how the polygons are drawn, the ORDER area of Polygon List RAM can store 2048 polygons. However, the storage capacity of the POLYGON area varies, depending on the conditions under which polygons are drawn. In this area, the polygon data comprises a header region of 12 bytes followed by a vertex index region of 8 or more bytes. Normally, each triangular polygon consumes 8 bytes of the vertex index region, and each quadrilateral polygon consumes 12 bytes. Accordingly, the maximum number of polygons that can be stored in the POLYGON area is calculated as follows:

(Connected) triangular polygons:  $40 \text{ KB} / (12 \text{ byte header} + 8 \text{ bytes}) = 2048 \text{ polygons}$

(Connected) quadrilateral polygons:  $40 \text{ KB} / (12 \text{ byte header} + 12 \text{ bytes}) = 1706 \text{ polygons}$

Note that 4 bytes in the vertex index region are consumed each time the number of vertices increases due to clipping. This means that the total number of polygons that can be stored in the POLYGON area decreases by one polygon for every five clippings performed on triangular polygons, and for every six clippings performed on quadrilateral polygons.

#### 2. Vertex RAM

The 72 KB of Vertex RAM is fully available to store vertex data. Each vertex consumes 12 bytes.

Table 6-6 shows the amount of vertex RAM consumed and the maximum number of polygons that can be stored for each primitive type.

**Table 6-6: Vertex RAM Consumed and the Maximum Number of Polygons Stored per Primitive Type**

Primitive Type	Vertex RAM Consumption	Maximum No. of Polygons that Can Be Stored
<b>Triangle Polygon</b>	3 vertices per polygon	2048
<b>Quadrilateral Polygon</b>	4 vertices per polygon	1536
<b>Connected Triangle Polygons</b>	First polygon: 3 vertices Later polygons: 1 vertex	6142 However, with 2050 vertices, the Polygon List RAM maximum of 2048 polygons is reached.
<b>Connected Quadrilateral Polygons</b>	First polygon: 4 vertices Later polygons: 2 vertices	3070 However, with 3414 vertices, the Polygon List RAM maximum of 1706 polygons is reached.

Because clipping also consumes Vertex RAM, connect polygons whenever possible as an effective method to avoid Vertex RAM overflow.

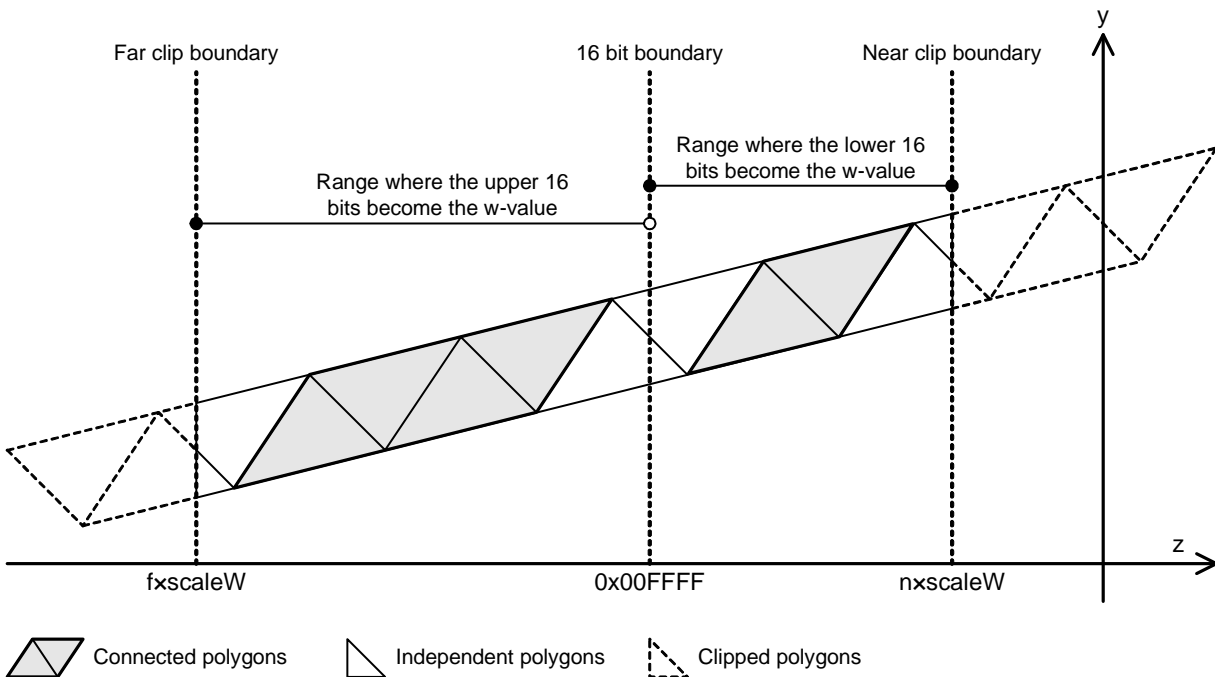
##### a. Why Shared Vertices are Released during both Z-Buffering and W-Buffering

When clipping is performed on connected polygons, the shared vertices between neighboring polygons are released.

### b. Why Shared Vertices are Released only during Z-Buffering

During Z-buffering, shared vertices are released if the W values (twice the clip-coordinate W values) stored in Vertex RAM for neighboring connected polygons are such that the W value for even one vertex of one of the polygons exceeds 16 bits (15 bits in clip coordinates) while not exceeding 16 bits for any of the vertices of the other polygon. (See Figure 6-28.) This happens because the Z value and the W value are both stored in Vertex RAM during Z-buffering, so the 24-bit W value can only be stored with 16-bit precision. What is stored is either the upper 16 bits or the lower 16 bits of the 24-bit W value, with the result that differences arise between polygons. This situation does not arise during W buffering because there is no need to store the Z value to vertex RAM during W buffering. Therefore, the 24-bit W value of the clip coordinate will be stored as it is, and the release of shared vertices does not occur.

**Figure 6-28: Release of Shared Vertices Among Connected Polygons (Clip Coordinate System)**



z axis: 24-bit W value

To read about n, f, scaleW, 24-bit W values see ["6.2.5 Depth Buffering"](#) on page 163.

### c. Reasons for Released Vertices in the Polygon Attribute Settings for Rendering 1-Dot Polygons

If "Do not render 1-dot polygons" is set in the polygon attributes and the polygon's W value exceeds the value in the 1-dot polygon display boundary depth value register, the vertices shared among the connected polygons will be released. In this case, whether or not a shared vertex will be released is decided for each polygon. Therefore, groups of polygons that do not exceed the value in the 1-dot polygon display boundary depth value register will share vertices, unaltered. To avoid this, set "Display 1-dot polygons" in the polygon attributes.

#### 4. Memory capacities and the storable number of polygons

The table shows the relationship between the capacity of each memory region and the number of polygons that can be stored in each region.

Memory Region	Relational Expression for Storable No. of Polygons & Memory Capacity
The POLYGON Area of Polygon List RAM	$(12+8) \times (F3+MF3) + (12+12) \times (F4+MF4) + 4 \times \text{CLIP} \leq 40\text{KB}$
Vertex RAM	$(12 \times 3) \times (F3+MP3+DMF3) + (12 \times 1) \times (MF3-MP3-DMF3) + (12 \times 4) \times (F4+MP4+DMF4) + (12 \times 2) \times (MF4-MP4-DMF4) + 12 \times \text{CLIP} \leq 72\text{KB}$

F3: Number of triangular polygons

MF3: Number of connected triangular polygons

MP3: Number of primitives of connected triangular polygons

F4: Number of quadrilateral polygons

MF4: Number of connected quadrilateral polygons

MP4: Number of primitives of connected quadrilateral polygons

CLI : Number of times clipping occurs

DMF3: Number of times vertices released for connected triangular polygons

DMF4: Number of times vertices released for connected quadrilateral polygons

#### 5. Summary

The actual number of polygons that are rendered is limited by the lesser of the number of polygons that can be stored in Polygon List RAM and in Vertex RAM.

When extensive use is made of polygon strips, Vertex RAM has plenty of space. Therefore, you can estimate, based on the capacity limitations of Polygon List RAM.

If you make extensive use of polygon strips, even in the unusual situation where every polygon is clipped once, you can be sure to obtain these numbers of polygons:

Triangular polygon strips:  $40 \text{ KB} / (12 + 8 + 4) = 1706$  polygons

Quadrilateral polygon strips:  $40 \text{ KB} / (12 + 12 + 4) = 1462$  polygons

#### 6.2.17 Warnings Regarding Calculation Precision

Although you can specify a 32-bit space for the world coordinate system (sign + 19-bit integer + 12-bit fractional part) with NITRO, objects on the edges of this 32-bit space may appear distorted and wrapped because the computational precision of the hardware is also 32 bits. To use the world coordinate system without problems, keep within a range of 29 bits (sign + 16-bit integer + 12-bit fractional part).

NITRO also has a 24-bit view space (sign 1 bit + 11-bit integer + 12-bit fractional part). Objects can appear distorted and wrapped if you specify some other space as the view volume.

## 6.3 Rendering Engine

### 6.3.1 Overview

Table 6-7 lists the rendering engine specifications.

**Table 6-7: Rendering Engine Specification List**

<b>Operating frequency</b>	33.514 MHz
<b>Render data</b>	Triangles and quadrilaterals
<b>Rendering capacity</b>	Maximum 120,000 polygons/sec (60FPS) Maximum 30 million pixels/sec (60FPS)
<b>Shadow surface process</b>	Switch between Z-value buffering and W-value buffering methods
<b>Shading</b>	Gouraud shading
<b>Texture mapping</b>	Perspective correction, modulation/decal Support for 4x4 texel compression Support for translucent textures Flip, Repeat Image sizes of 8x8 texels to 1024x1024 texels
<b>Other capabilities (See Table 6-8)</b>	Alpha blending, alpha test, anti-aliasing, edge marking, fog, toon shading, highlight shading, shadow, wireframe, Clear Image



Table 6-8 gives an overview of rendering engine features.

**Table 6-8: Overview of Rendering Engine Features**

<b>Alpha Blending</b>	Blends the color value stored in the color buffer with the input fragment's color value, based on the alpha value of that fragment (the fragments after texture blending).
<b>Alpha Test</b>	Compares the fragment's alpha value with the reference value set in the register, and draws only if the fragment's alpha value is larger than this reference value. (These fragments are the fragments after texture blending.)
<b>Antialiasing</b>	Blends the color of the polygon's boundaries with color values of the polygon behind it, using the (5-bit width) factor computed based on the shift from the original display position.
<b>Edge Marking</b>	Marks the boundary edges of polygons with different polygon IDs (6-bit) using the polygon edge-specified color (8 colors). When anti-aliasing is enabled, the edge marking is followed by anti-aliasing.
<b>Fog</b>	Using the fog density table, the specified fog's color value is blended with the color buffer's color value. The fog density can be specified in 32 levels, and the value that is applied is the value that results from linear interpolation with the depth value of the target pixels. When 3D is displayed in front of a 2D screen, fog can be applied to the 2D screen as well by using the color buffer.
<b>Toon Shading</b>	Can present cartoon-like pictures by steepening the shininess calculation results.
<b>Highlight Shading</b>	Can present shininess beyond the texture color.
<b>Shadow</b>	Can easily put shadows on even bumpy surfaces by defining the shadow volume.
<b>Wireframe</b>	Can draw only the edges of polygons without drawing the surfaces.
<b>Clear Image</b>	Can apply Clear Images in VRAM as the initial values for the color buffer's, depth buffer's and attribute buffer's fog-enable flags.

**Note:** The rendering-related registers have a double-buffer structure, and the contents of each register are sent to the Rendering Engine at the start of the V-Blank period that begins right after the `SwapBuffers` command is issued. Therefore, data can be written to these registers even during the middle of a frame. The data is not reflected in the image drawn in the frame at the time of the change.

## 6.3.2 Rendering Methods

### 6.3.2.1 Line Buffer Rendering

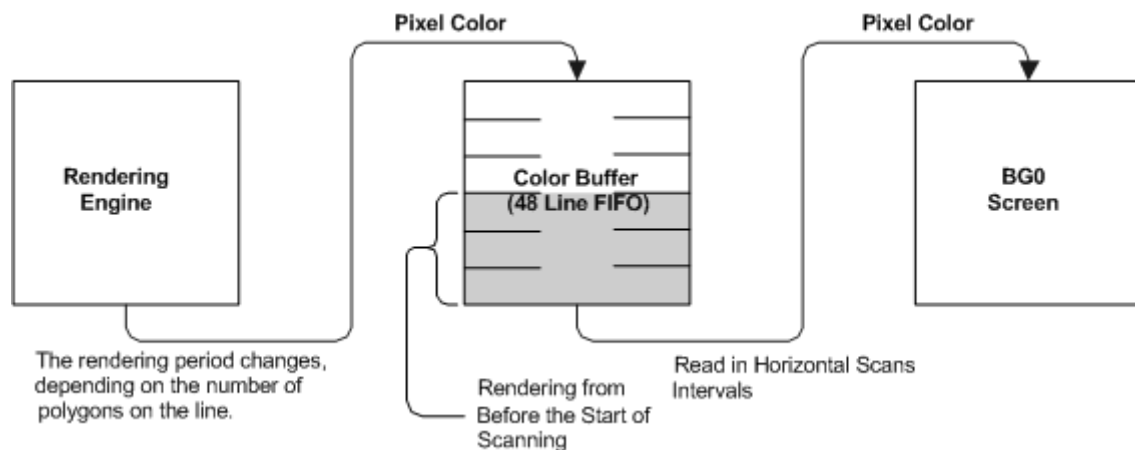
NITRO employs a line-buffer method for rendering, rather than a frame-buffer method. Because of this, line overflow can occur when too many polygons are layered on a single line to be rendered during the horizontal period. To prevent line overflow from occurring easily, the color buffer operates FIFO, holding 48 lines, and the rendering begins from the middle of the V-Blank period, drawing and storing the line data before display (see Figure 6-29.)

By reading the RDLINES\_COUNT register, you can check the minimum number of lines that remain in the color buffer while the frame that has been rendered is displaying. In other words, you cannot confirm whether or not lines have been dropped, but you can determine the risk of this happening.

To read about the RDLINES\_COUNT register see ["6.3.11 Status"](#) on page 267.

This diagram shows only the FIFO operation. The Rendering Engine is actually reading from and writing to the color buffer.

**Figure 6-29: Color Buffer's FIFO Operation**



### 6.3.2.2 Buffers in the Rendering Engine

Table 6-9 shows the buffers in the Rendering Engine that store information about every pixel.

Also see ["Figure 6-1 : 3D Graphics Hardware Block Diagram"](#) on page 153.

**Table 6-9: Buffers in the Rendering Engine**

<b>Stencil buffer</b>	This buffer holds one line at 1 bit/pixel. It is used when rendering shadow polygons.
<b>Attribute buffer</b>	This buffer holds two lines at 23 bits/pixel. This buffer stores the polygon ID and fog enable flag for every pixel. Polygon IDs are managed separately for opaque polygons and translucent polygons.
<b>Depth buffer</b>	This buffer holds two lines at 24 bits/pixel. It is used for depth test and fog blending calculations.
<b>Color buffer</b>	This buffer holds 48 lines at 23 bits/pixel (R:G:B:A = 6:6:6:5).

### 6.3.2.3 Blank Periods

The Rendering Engine begins rendering when scanning starts for the LCD's 214<sup>th</sup> line, and it keeps rendering until the start of display of the 191<sup>st</sup> final line. Thus, the Rendering Engine's blank period is the 23 lines from line 191 to 213 (see Figure 6-30).

To safely rewrite data to the VRAM region (texture image and texture palette) referenced by the Rendering Engine, do so during these 23 lines.

**Figure 6-30: Rendering Engine Blanking Periods**

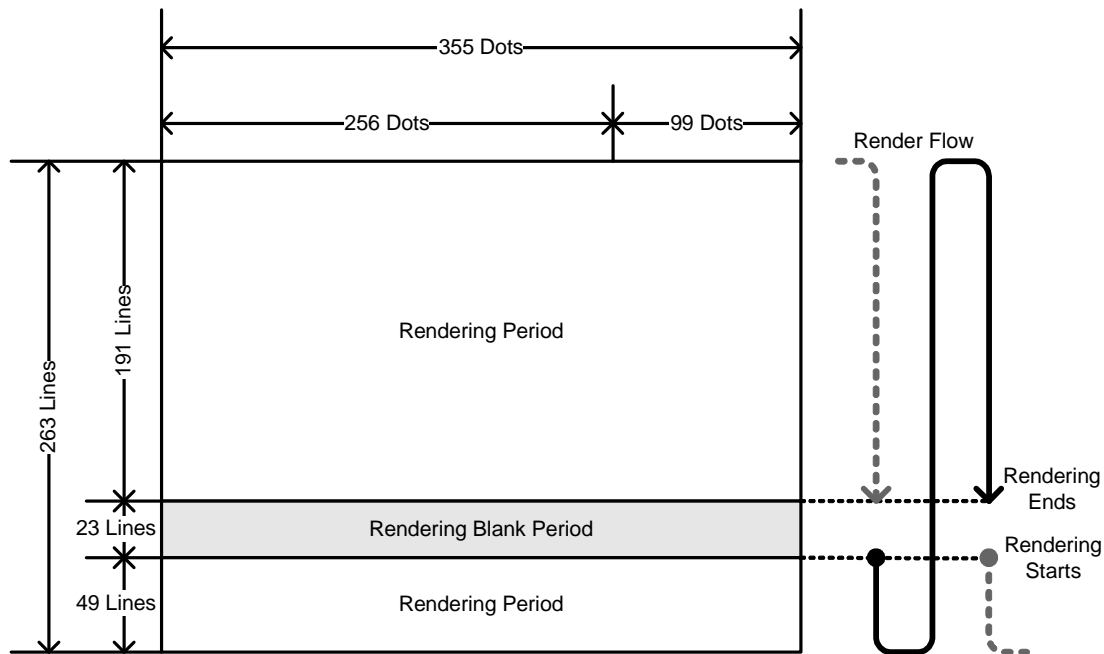


Table 6-10 shows the Rendering Engine timing specifications.

**Table 6-10: Rendering Engine Timing Specifications**

Item		Spec	Period
Rendering Period	Number of Horizontal Dots	256 dots	45.8316 $\mu$ s
	Number of Vertical Lines	49+191 lines	15.2533 ms
Total Period	Number of Horizontal Dots	355 dots	63.5556 $\mu$ s
	Number of Vertical Lines	263 lines	16.7151 ms
Blank Period	Number of blank Lines	23 lines	1.4618 ms
Scan Cycle	V Cycle	59.8261Hz	16.7151 ms

### 6.3.2.4 Number of Polygons that can be Drawn with One Line

The Rendering Engine can draw for a period of 355 dots with one line. (See "[Table 6-10 : Rendering Engine Timing Specifications](#)" on page 229).

Each dot involves 6 cycles, but since there is also an overhead of 4 cycles for each line, the number of cycles that can be used for rendering is  $(355 \times 6) - 4 = 2126$  cycles.

In NITRO, the fill rate for even texture-mapped translucent polygons is 1 pixel per cycle, but there is an 8-cycle overhead every time rendering of a polygon starts.

Given these factors, the minimum guaranteed number of polygons that can be drawn on one line is:

$$(2126 \text{ cycles}) / (8 \text{ cycles} + \text{number of horizontal pixels in polygon}).$$

Excluding the overhead, the per-line fill rate is:

$$(2126 \text{ cycles}) - (8 \text{ cycles} \times \text{guaranteed number of polygons drawn with one line}).$$

**Note:** Using these formulas yields the results shown in Table 6-11.

**Table 6-11: Maximum Polygons Rendered per Line and Fill Rate (Calculated Values)**

Number of Horizontal Pixels in the Polygon	8	16	32	64	128	256
Number of Polygons Guaranteed to Be Drawn by One Line	132	88	53	29	15	8
Per-line Fill Rate	1070	1422	1702	1894	2006	2062

Because the line buffer is FIFO, the actual number of polygons drawn may be higher.

### 6.3.3 Initializing the Rendering Buffers

#### 6.3.3.1 Initializing with the Clear Registers

##### ClearColorAttr: Clear Color Attribute Register

Name: CLEAR\_COLOR      Address: 0x04000350      Attribute: W      Initial value: 0x00000000

31	29	24	23	20	16	15	14	10	9	8	7	5	4	0
						F		BLUE			GREEN			RED
		Clear Polygon ID						$\alpha$ value				Fog		Color

- **Clear Polygon ID [d29–d24] :** Polygon ID initial value  
Sets the initial value of the opaque Polygon ID in the Attribute buffer. Whether edge marking is applied is determined by comparing the polygon ID and this Clear Polygon ID in the case of edges that can be clipped at the edge of the screen.
- **$\alpha$  value[d20–d16] :** Initial value of  $\alpha$   
Sets the initial value of the  $\alpha$  value in the Color buffer.  
Normally, set this to 0 when compositing with 2D.
- **F[d15] :** Fog enable flag  
Sets the initial value of the fog enable flag in the Attribute buffer.  
When compositing with the 2D screen, you can use this to control whether or not to apply fog to the rear plane.  
This is effective when you want to clearly display a 2D background.
- **Color[d14–d00] :** Clear Color RGB values  
Sets the Color buffer's initial RGB values.  
The Color buffer in the Rendering Engine is (R:G:B = 6:6:6) bits, so the lower 1 bit is treated as 0 when the Clear Color value is 0, and as 1 when the value is non-zero.

##### ClearDepth: Clear Depth register

Name: CLEAR\_DEPtH      Address: 0x04000354      Attribute: W      Initial value: 0x7FFF

15	14	8	7	0
		CLEARDEPTH		
		Clear Depth Value		

- **CLEARDEPtH[d14–d00] :** Clear Depth value  
The Depth buffer in the Rendering Engine is 24 bits/pixel, so the Clear Depth value is used after shifting 9 bits to the left. The lower 9 bits are treated as 0, except when the Clear Depth value is 0x7FFF, in which case the lower 9 bits are treated as 1.  
To read how this differs from the depth value in the different depth buffering methods, see ["6.2.5 Depth Buffering"](#) on page 163.

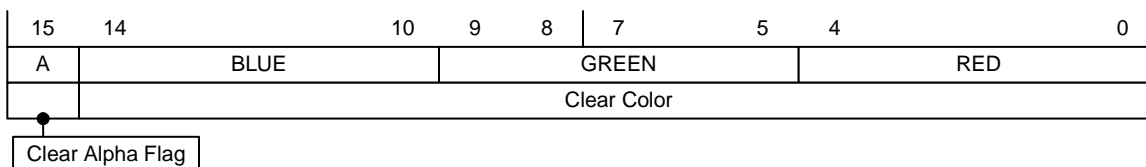
### 6.3.3.2 Initializing with Clear Images

When the Clear Image Enable Flag is set in the 3D Display Control register (DISP3DCNT), the Clear Images stored in VRAM are used as the initial values of the Color buffer and Depth buffer and the Attribute buffer's fog enable flag. Even when this feature is used, the value of the CLEAR\_COLOR register is still used for the Attribute buffer's polygon ID.

Use the RAM Bank Control register to assign the VRAM that stores Clear Images to the Clear Image buffer.

The format for each Clear Image is given below. "[Figure 6-31 : VRAM Mapping of Clear Images \(Texture Image Slots 2 and 3 Shared\)](#)" on page 233 shows Clear Image VRAM mapping.

#### Clear Color Image Format



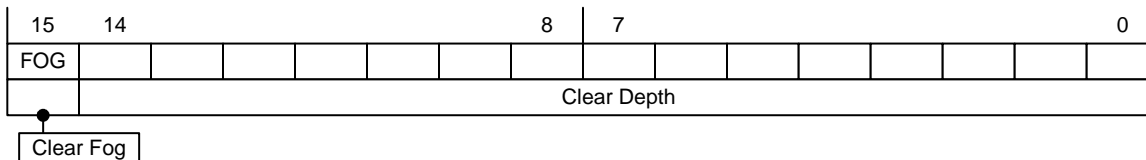
- [d15] : Clear  $\alpha$  flag

The actual clear  $\alpha$  values are as follows:

0	0x00
1	0x1F

- [d14–d00] : Clear Color RGB values
- Sets the Color buffer's RGB initial values.
- The Color buffer in the Rendering Engine is (R:G:B = 6:6:6) bits, so the lower 1 bit is treated as 0 when the Clear Color value is 0, and as 1 when the value is non-zero.

#### Clear Depth Image Format



- FOG[d15]: Clear Fog

Sets the initial value of the fog enable flag in the Attribute buffer.

When compositing with the 2D screen, you can use this to control whether or not to apply fog to the rear plane.

This is effective when you want to clearly display a 2D background.

- [d14–d00] : Clear Depth

To read how this differs from the depth value in the different depth buffering methods, see "[6.2.5 Depth Buffering](#)" on page 163

**Figure 6-31: VRAM Mapping of Clear Images (Texture Image Slots 2 and 3 Shared)**

	Dot 0	1	2	3		253	254	255
Line 0	0h	2h	4h	6h		1FAh	1FCh	1FEh
1	200h	202h	204h	206h		3FAh	3FCh	3FEh
2	400h	402h					5FCh	5FEh
3	600h	602h					7FCh	7FEh
4	800h							9FEh
251	1F600h							1F7FEh
252	1F800h	1F802h					1F9FCh	1F9FEh
253	1FA00h	1FA02h					1FBFCh	1FBFEh
254	1FC00h	1FC02h	1FC04h	1FC06h		1FDFAh	1FDfCh	1FDfEh
255	1FE00h	1FE02h	1FE04h	1FE06h		1FFFAh	1FFFCh	1FFFEh

**ClearImageOffset: Clear Image Offset Settings Register**

Name: CLRIMAGE\_OFFSET

Address: 0x04000356

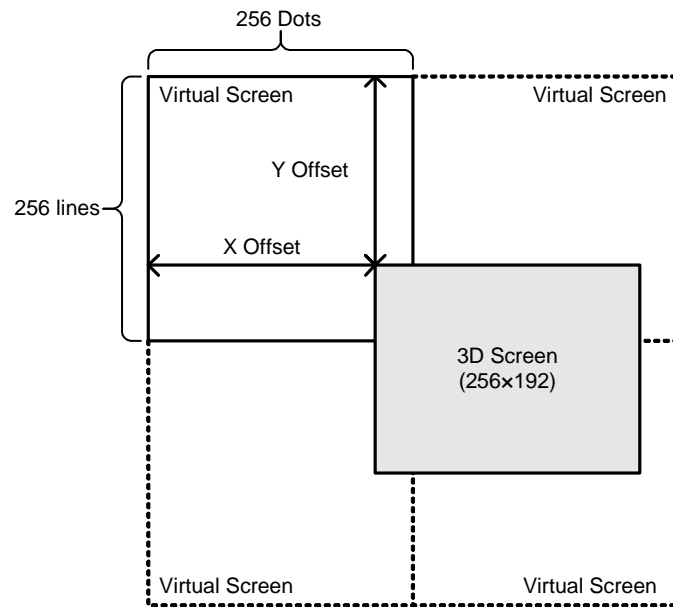
Attribute: W

Initial value: 0x0000

15							8	7							0
Y Offset								X Offset							

Can assign offsets to the Clear Images that are read when the rendering buffers are initialized. The image is wrapped and read if it exceeds the 256x256 data region on screen. Figure 6-32 shows the clear image offset.

**Figure 6-32: Clear Image Offset**





### 6.3.4 Rasterizing

*Rasterization* is the process of dividing the polygon surface into pixels and writing them to a buffer. During rasterization, the Rendering Engine interpolates the pixel colors inside the polygon, based on the vertex colors passed from the Geometry Engine.

The Rendering Engine stores the pixel colors in the Color buffer and stores each pixel's polygon ID and fog enable flag in the Attribute buffer. The rendering engine first finishes rendering the opaque polygon that are in polygon list RAM, and then renders the translucent polygons.

#### 6.3.4.1 Opaque Polygons

An opaque polygon is a polygon with an  $\alpha$  value of 31 ( $\alpha = 31$ ).

- Polygon ID  
This is stored in the region for opaque polygon IDs in the Attribute buffer when the polygon is rendered.
- Fog enable flag  
When an opaque polygon is rendered, the fragment's fog enable flag overwrites the Attribute buffer's fog enable flag.

#### 6.3.4.2 Translucent Polygons

A translucent polygon has either an  $\alpha$  value between 1 and 30 ( $1 \leq \alpha \leq 30$ ) or a translucent texture applied.

Therefore, this term also encompasses shadow polygons.

##### 6.3.4.2.1 Polygons with $1 \leq \alpha \leq 30$

- Polygon ID  
This is stored in the region for translucent polygon IDs in the Attribute buffer when the polygon is rendered.  
When different translucent polygons overlap on the screen, a translucent polygon that uses the same polygon ID as another translucent polygon is not overwritten.
- Fog enable flag  
When a translucent polygon is rendered, the fragment's fog enable flag and the Attribute buffer's fog enable flag are combined with a logical AND operation, and the result is written to the Attribute buffer.  
This feature can be used to apply fog to everything except specific translucent polygons.

#### 6.3.4.2.2 Translucent texture-mapped polygons

Translucent texture-mapped polygons (A3I5 and A5I3 textures) are stored in the translucent polygon region of the polygon list RAM, even if all of the texels are opaque. Therefore, they are rendered after opaque polygons. Notice that processing of polygon IDs and fog enable flags differs, according to whether the pixel is opaque ( $\alpha=31$ ) or translucent ( $1 \leq \alpha \leq 30$ ). Therefore, opaque pixels and translucent pixels may be mixed within a polygon.

- Polygon ID
  - Opaque pixels
 

When rendering polygons, these are stored in the attribute buffer's opaque polygon ID area.

Therefore, they are targeted for edge marking.
  - Translucent pixels
 

When rendering polygons, they are stored in the attribute buffer's translucent polygon ID area.

If different translucent polygons overlap on the screen, translucent polygons that have the same polygon ID are not overwritten.
- Fog enable flag
  - Opaque pixels
 

Fragment fog enable flags are overwritten by the attribute buffer's fog enable flags.
  - Translucent pixels
 

When a translucent polygon is drawn, the fragment's fog enable flag and the Attribute buffer's fog enable flag are combined with a logical AND operation, and the result is written to the Attribute buffer.

This feature can be used to apply fog to everything except specific translucent polygons.

#### 6.3.4.3 Wireframes

A *wireframe* is a polygon with an  $\alpha$  value of 0 ( $\alpha = 0$ ). In this case,  $\alpha$  does not have its original meaning of opacity level. Instead, only the outline of the polygon (wireframe) is rendered. If it is clipped, the *clipping boundary* (a new side created due to clipping) is also rendered as a wireframe. To render a wireframe as translucent, map a translucent texture.

**Note:** The characteristics of the circuit do not permit a wireframe to be drawn semi-transparently.

#### About the Polygon ID

When a wireframe is rendered, it is stored in the opaque polygon ID region of the attribute buffer. (Only the polygon ID of the wire is updated.)

#### 6.3.4.4 Shadow Polygons

*Shadow volume* is defined as the space that is not illuminated by light because the light has been obstructed by an object. A shadow, then, can be thought of as something that is generated in the region where the shadow volume intersects with another object. The polygon used to express a shadow volume is called a *shadow polygon*.

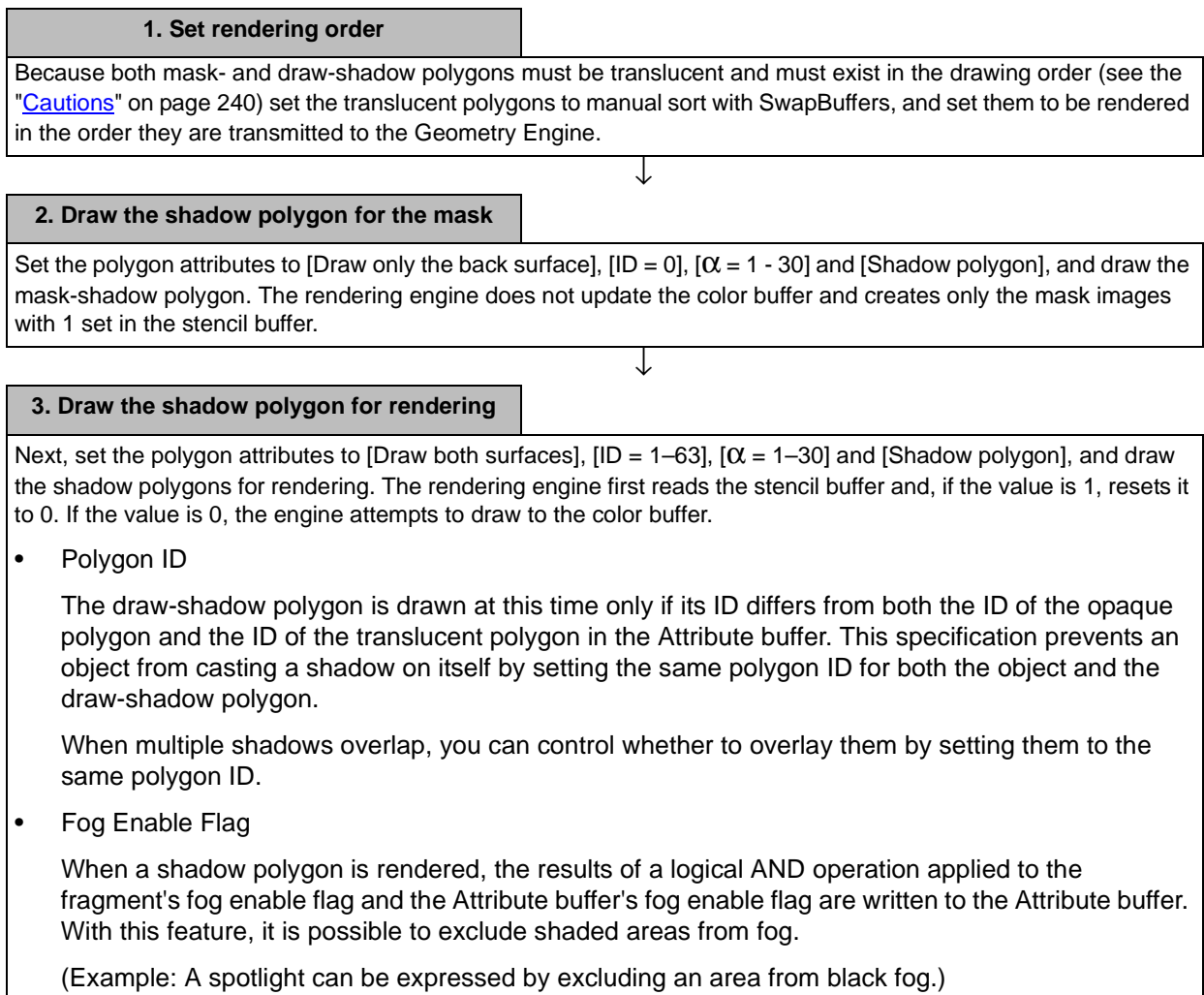
You can create a shadow image that can be seen from the user's perspective by creating only a mask image of the Stencil buffer when rendering the inner side of the shadow volume, and then excluding the mask region when drawing outside the shadow volume.

Shadow polygons used for masks and rendering can be differentiated by their polygon IDs.

Polygon ID	Classification
0	Shadow polygons for masks
1–63	Shadow polygons for drawing

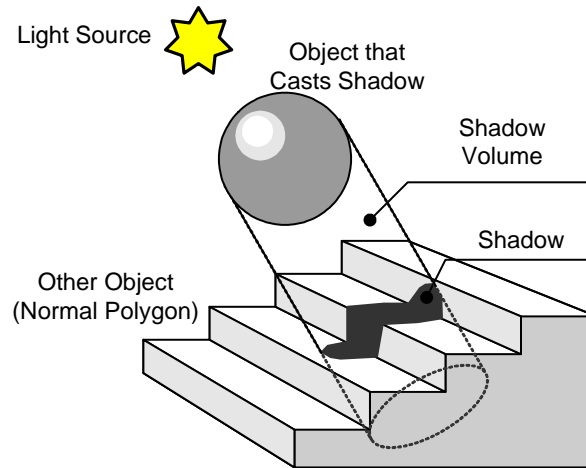
Figure 6-33 illustrates the concept of shadow polygons. "[Figure 6-34 : When Drawing a Shadow Polygon for a Mask](#)" on page 239 describes rendering shadow polygons for masking. "[Figure 6-34 : When Drawing a Shadow Polygon for a Mask](#)" on page 239 describes rendering shadow polygons for drawing.

The procedure for attaching shadows using shadow polygons is as follows:



- Shadow Volume

**Figure 6-33: Shadow Volume**



- **Shadow-volume shape**

In principle, the shadow volume takes the shape of a closed 3D shape. Note the "[Cautions](#)" on page 240 if you plan to use an open 3D shape in order to reduce the number of polygons.

- **Shadow-volume direction**

To create the shadow of a spherical object, a cylinder-shaped shadow volume is created on the straight line defined by the light source and the spherical object.

- **Shadow-volume position**

The cylinder-shaped shadow volume is located where it cannot be seen from the light source (a place not illuminated by the light of the sphere).

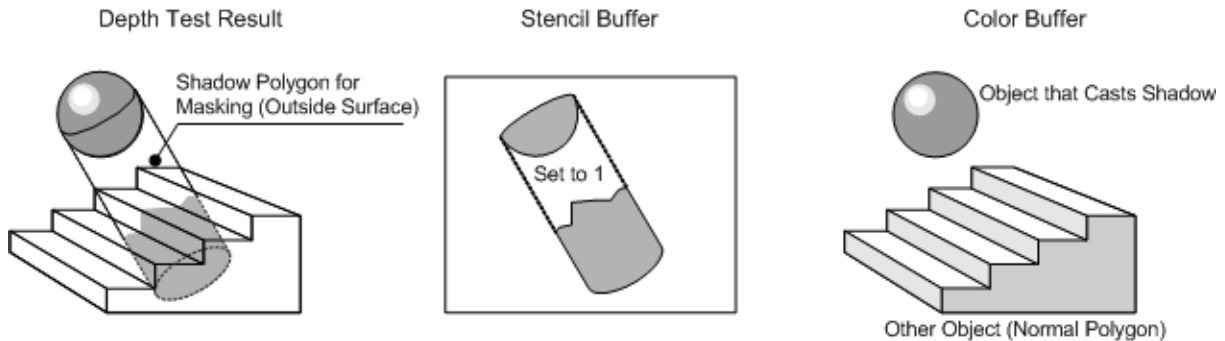
- **Shadow-volume length**

The shadow is drawn on the surface of the object that is located inside the shadow volume. Thus, the length of the shadow volume should be long enough to pass through the surface of the object on which you want the shadow to be drawn.

**Stencil buffer calculations**

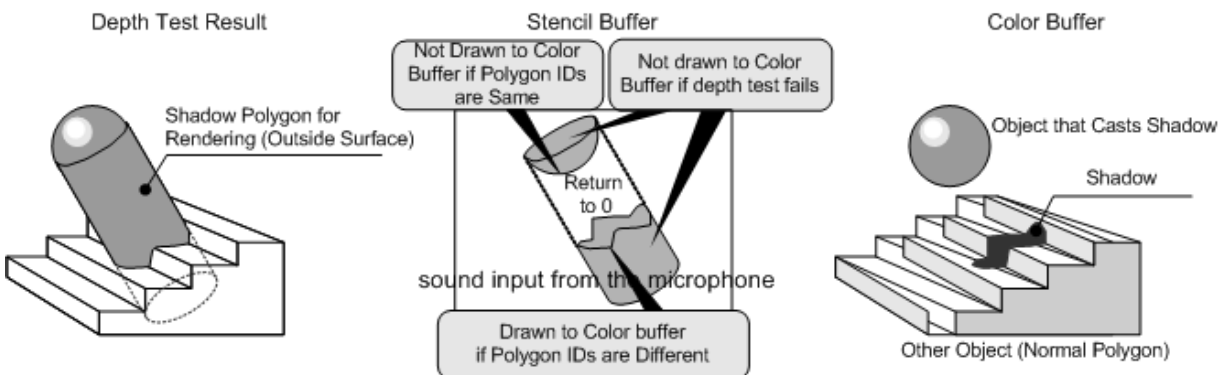
## 1. When drawing a shadow polygon for a mask

The Stencil buffer is set to 1 when the depth test fails without affecting the Color buffer.

**Figure 6-34: When Drawing a Shadow Polygon for a Mask**

## 2. When drawing a shadow polygon for rendering

If the Stencil buffer is already set to 1, it is reset to 0. When the depth test succeeds, if the polygon ID in the attribute does not match the polygon ID of the shadow polygon for rendering, the polygon is drawn to the color buffer.

**Figure 6-35: When Drawing the Shadow Polygon for Rendering**

As the diagrams illustrate, the shadow is drawn to a portion of one side of the shadow polygon for rendering. That is why you cannot achieve the effect of directly pasting textures to shadows when texture mapping.

## **Cautions**

### **1. Order for rendering shadow polygons**

To create a single shadow, draw the necessary shadow polygon for the mask, and then the shadow polygon for rendering. If you draw a collection of shadow polygons for the mask and then draw the corresponding collection of shadow polygons for rendering, shadows may not be created in the intended regions.

Correct rendering string:

Shadow volume for mask 1 → Shadow volume for rendering 1 →

Shadow volume for mask 2 → Shadow volume for rendering 2

### **2. Shadow volumes with open shapes**

When the shadow volume takes the shape of an open 3D shape, a region that has a shadow polygon for rendering but not a shadow polygon for the mask may be generated. This results in the creation of an incorrect shadow.

Note that when a shadow volume is clipped, it takes on an open shape.

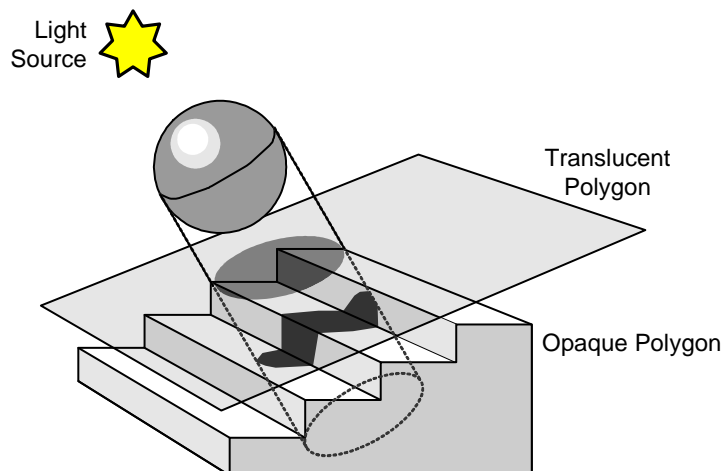
## **Technique**

By following the procedure below, you can create scenes in which shadows are cast on translucent polygons. (However, this technique requires more shadow polygons.)

1. Opaque polygon →
2. Shadow polygon (shadow on opaque polygon) →
3. Translucent polygon on which a shadow is cast (update depth buffer) →
4. Shadow polygon (the shadow on the translucent polygon / polygon ID is the same as in step 2) →
5. Translucent polygon on which the shadow is not cast.

Figure 6-36 shows a schematic representation.

**Figure 6-36: Technique for Rendering a Shadow on a Translucent Polygon**



### 6.3.4.5 Toon Shading/Highlight Shading

#### ToonTable: Toon Table Register

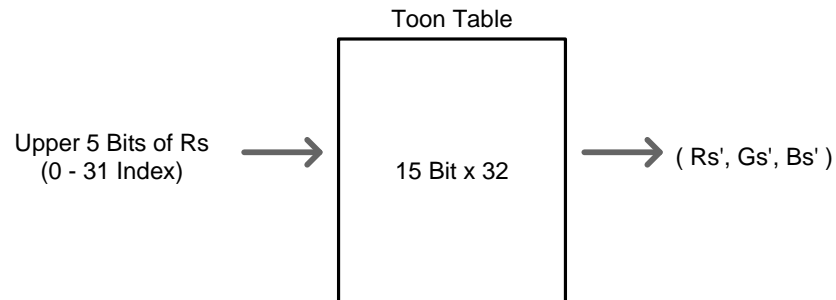
Name	Address	Attribute	Initial Value
TOON_TABLE_x (x=0-15)	0x04000380, 0x04000384, 0x04000388, 0x0400038C, 0x04000390, 0x04000394, 0x04000398, 0x0400039C, 0x040003A0, 0x040003A4, 0x040003A8, 0x040003AC, 0x040003B0, 0x040003B4, 0x040003B8, 0x040003BC	W	0x00000000

31	30	26	25	24	23	21	20	16	15	14	10	9	8	7	5	4	0	
	BLUE n1			GREEN n1			RED n1				BLUE n0			GREEN n0			RED n0	
RGB transformation values when brightness n1=2x+1									RGB transformation values when brightness n0=2x									

This register is used for toon shading and highlight shading.

In toon shading and highlight shading, the fragment color R value is treated as the brightness, and this R value (upper 5 bits) is used as the index to reference RGB values from the toon table.

**Figure 6-37: Transformations Using a Toon Table**



### 6.3.4.5.1 Toon Shading

The fragment color R value is treated as the brightness, and this R value (the upper 5 bits) is used as the index to reference RGB values from the toon table to set the new fragment colors.

The texture color and the post-table-reference fragment color both have ( $R_5:G_5:B_5 = 5:5:5$ ) number of bits. Therefore, before the texture-blending computation is conducted, the following formulas are used to expand the number of bits to ( $R_6:G_6:B_6 = 6:6:6$ ).

$$R_6 = R_5 \ll 1 \quad (\text{When } R_5 \text{ is } 0)$$

$$R_6 = (R_5 \ll 1) + 1 \quad (\text{When } R_5 \text{ is nonzero})$$

Texture mapping involves the same equations used in Modulation mode, except that the toon table-transformed values are used for the fragment colors (see Table 6-12).

**Table 6-12 : Texture Blending Equations (toon table)**

Type of Texture	Translucent Texture	Non-translucent Texture
<b>Texture Blending Equations</b>	$R = \{ (R_t+1) \times (R_s'+1) - 1 \} / 64$ $G = \{ (G_t+1) \times (G_s'+1) - 1 \} / 64$ $B = \{ (B_t+1) \times (B_s'+1) - 1 \} / 64$ $A = \{ (A_t+1) \times (A_s'+1) - 1 \} / 64$	$R = \{ (R_t+1) \times (R_s'+1) - 1 \} / 64$ $G = \{ (G_t+1) \times (G_s'+1) - 1 \} / 64$ $B = \{ (B_t+1) \times (B_s'+1) - 1 \} / 64$ $A = A_t \times A_s$

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(R<sub>t</sub>, G<sub>t</sub>, B<sub>t</sub>, A<sub>t</sub>) : Texture color expanded to (R:G:B = 6:6:6) bits

(R<sub>s</sub>, G<sub>s</sub>, B<sub>s</sub>, A<sub>s</sub>) : Fragment color

(R<sub>s'</sub>, G<sub>s'</sub>, B<sub>s'</sub>) : Fragment color expanded to (R:G:B = 6:6:6) bits after table conversion

**Note:** The toon table is referenced for fragment colors shared by all toon-shaded polygons. If you want every toon shading polygon to have different colors, use textures to color even monochrome polygons.

Using ambient reflection color and emission color to raise the minimum value of the fragment color R value also narrows the effective range of R, and this coarsens the gradation of toon shading. Because material color loses its meaning when toon shading, we recommend that you use only diffusion reflection color so as to retain gradation.



### 6.3.4.5.2 Highlight Shading

The fragment color R value is treated as the brightness, and this R value (the upper 5 bits) is used as the index to reference RGB values (color offset values) from the toon table to add to the fragment color.

The texture color and the post-table-reference fragment color both have ( $R_5:G_5:B_5 = 5:5:5$ ) number of bits. Therefore, before the texture-blending computation is conducted, the following formulas are used to extend the number of bits to ( $R_6:G_6:B_6 = 6:6:6$ ).

$$R_6 = R_5 \ll 1 \quad (\text{When } R_5 \text{ is } 0)$$

$$R_6 = (R_5 \ll 1) + 1 \quad (\text{When } R_5 \text{ is nonzero})$$

When texture mapping, a color offset value is added to texture colors whose RGB values have each been modulated by the fragment color's R value (see Table 6-13). This can produce an effect as if the texture is highlighted (emitting a color brighter than the texture's own color).

**Table 6-13 : Texture Blending Equation (Highlight Shading)**

Texture Type	Translucent Texture	Non-translucent Texture
<b>Texture Blending Equations</b>	$R = \min[63, \{ (R_t+1) \times (R_s+1) - 1 \} / 64 + R_s']$ $G = \min[63, \{ (G_t+1) \times (R_s+1) - 1 \} / 64 + G_s']$ $B = \min[63, \{ (B_t+1) \times (R_s+1) - 1 \} / 64 + B_s']$ $A = \{ (A_t+1) \times (A_s+1) - 1 \} / 32$	$R = \min[31, \{ (R_t+1) \times (R_s+1) - 1 \} / 64 + R_s']$ $G = \min[31, \{ (G_t+1) \times (R_s+1) - 1 \} / 64 + G_s']$ $B = \min[31, \{ (B_t+1) \times (R_s+1) - 1 \} / 64 + B_s']$ $A = A_t \times A_s$

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(R<sub>t</sub>, G<sub>t</sub>, B<sub>t</sub>, A<sub>t</sub>) : Texture color extended to (R:G:B = 6:6:6) bits

(R<sub>s</sub>, G<sub>s</sub>, B<sub>s</sub>, A<sub>s</sub>) : Fragment color

(R<sub>s</sub>', G<sub>s</sub>', B<sub>s</sub>') : Fragment color extended to (R:G:B = 6:6:6) bits after table conversion

**Note:** Because a color offset is added, sometimes the hue of the fragment color changes.

## 6.3.5 Textures

### 6.3.5.1 Texture Blending

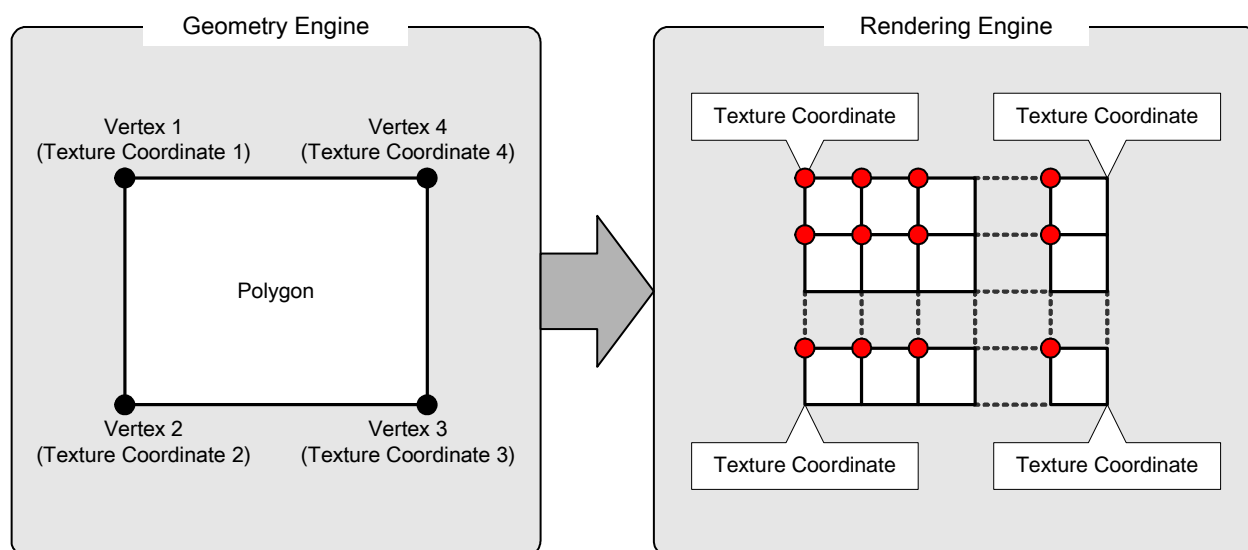
The Rendering Engine interpolates the texture coordinates corresponding to each pixel inside the polygon using the texture settings (flip/repeat) passed by the Geometry Engine, the vertex texture coordinates, and the *w* value.

The process of blending texel color with the color of each pixel of the polygon is called texture blending. The mode for this texture blending can be set to either Decal or Modulation, using the PolygonAttr register.

#### 6.3.5.1.1 Texture Image Sampling

In NITRO, the texture image is sampled by interpolating, from each vertex's texture coordinates, texture coordinates that correspond to the top-left of each pixel in a polygon (see Figure 6-38).

**Figure 6-38: Texture Image Sampling**

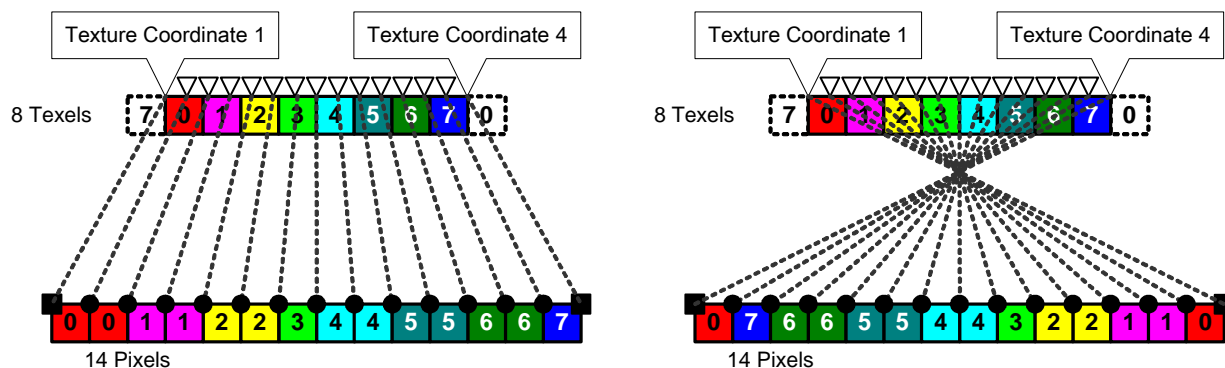


For example, if a polygon has 8x8 textures applied to it and its display width is 14 dots, pixel and texel correspondence are as shown on the left in Figure 6-39.

If a polygon is turned front to back like this, the texel/pixel correspondence slips.

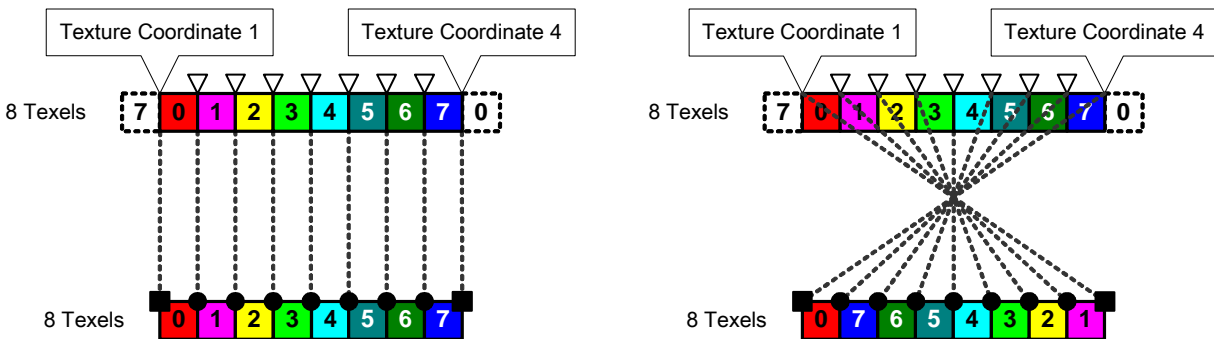
In the diagram on the right of Figure 6-39, texel 0 appears at the left edge (if the texture is not H-flipped it is omitted). On the right edge, texel 0 is only sampled once.

**Figure 6-39: When an 8x8 texel Texture is Applied to an 14-dot Wide Polygon**



If a polygon has 8x8 textures applied to it and its display width is 8 dots, pixel and texel correspondence are as shown in Figure 6-40.

**Figure 6-40: When an 8x8 texel Texture Is Applied to an 8-dot Wide Polygon**



When using polygons for 2D displays such as OBJ or BG, texels and pixels have a one-to-one correspondence. However, in a case such as this, when rendering a polygon that is mapped with an 8x8 texel texture on 8x8 pixels of an LCD, the front and back surfaces are displayed as shown in Figure 6-41.

**Figure 6-41: Displaying Front and Back Surfaces of an LCD**



### 6.3.5.1.2 Decal Mode

Depending on the texture's  $\alpha$  value, either the result of Gouraud shading of the vertex color created by the lighting process (fragment color) or the texture's color value is displayed.

For translucent textures, blending is done with the texture's  $\alpha$  value.

Table 6-14 shows the texture-blending expression used in decal mode.

The texture color has ( $R_5:G_5:B_5 = 5:5:5$ ) number of bits. Therefore, before the texture-blending computation is conducted, the following formulas are used to expand the number of bits to ( $R_6:G_6:B_6 = 6:6:6$ ).

$$R_6 = R_5 \ll 1 \quad (\text{When } R_5 \text{ is } 0)$$

$$R_6 = (R_5 \ll 1) + 1 \quad (\text{When } R_5 \text{ is non-zero})$$

**Table 6-14 : Texture Blending Equations (Decal Mode)**

Texture Type	Translucent Texture	Non-translucent Texture
<b>Texture Blending Equations</b>	$R = \{At \times Rt + (31 - At) Rs\} / 32$ $G = \{At \times Gt + (31 - At) Gs\} / 32$ $B = \{At \times Bt + (31 - At) Bs\} / 32$ $A = As$  Handling exceptions: When $At = 0$ , $(R, G, B, A) = (Rs, Gs, Bs, As)$ is used. When $At = 31$ , $(R, G, B, A) = (Rt, Gt, Bt, As)$ is used.	$R = At \times Rt + (1 - At) Rs$ $G = At \times Gt + (1 - At) Gs$ $B = At \times Bt + (1 - At) Bs$ $A = As$

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(Rt, Gt, Bt, At) : Texture color expanded to (R:G:B = 6:6:6) bits

(Rs, Gs, Bs, As) : Fragment color

### 6.3.5.1.3 Modulation Mode

The result of the Gouraud shading of the vertex color (fragment color) created by the lighting process is modulated by the texture's color value and displayed.

Table 6-15 shows the texture-blending expressions used in modulation mode.

The texture color's fragment color has ( $R_5:G_5:B_5 = 5:5:5$ ) number of bits. Therefore, before the texture-blending computation is conducted, the following formulas are used to expand the number of bits to ( $R_6:G_6:B_6 = 6:6:6$ ).

$$R_6 = R_5 \ll 1 \quad (\text{When } R_5 \text{ is } 0)$$

$$R_6 = (R_5 \ll 1) + 1 \quad (\text{When } R_5 \text{ is nonzero})$$

**Table 6-15 : Texture Blending Expressions (Modulation Mode)**

Type of Texture	Translucent Texture	Non-Translucent Texture
<b>Texture Blending Equations</b>	$R = \{ (R_t+1) \times (R_s+1) - 1 \} / 32$ $G = \{ (G_t+1) \times (G_s+1) - 1 \} / 32$ $B = \{ (B_t+1) \times (B_s+1) - 1 \} / 32$ $A = \{ (A_t+1) \times (A_s+1) - 1 \} / 32$	$R = \{ (R_t+1) \times (R_s+1) - 1 \} / 32$ $G = \{ (G_t+1) \times (G_s+1) - 1 \} / 32$ $B = \{ (B_t+1) \times (B_s+1) - 1 \} / 32$ $A = A_t \times A_s$

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(R<sub>t</sub>, G<sub>t</sub>, B<sub>t</sub>, A<sub>t</sub>) : Texture color expanded to (R:G:B = 6:6:6) bits

(R<sub>s</sub>, G<sub>s</sub>, B<sub>s</sub>, A<sub>s</sub>) : Fragment color

### 6.3.5.2 Texture Formats

NITRO can handle seven different texture formats. Table 6-16 lists the texture formats.

**Table 6-16 : List of Texture Formats**

Format	Number of Selectable Colors for 1 Texel	Palette Base Boundary (see note)	α Value Bits	Number of Bits per Texel
4-Color Palette Texture	4	0x08	0	2
16-Color Palette Texture	16	0x10	0	4
256-Color Palette Texture	256	0x10	0	8
4x4 Texel Compressed Texture	4 (every 4x4 texels)	0x10	0	3 (Includes palette index data)
A3I5 Translucent Texture	32	0x10	3	8
A5I3 Translucent Texture	8	0x10	5	8
Direct Color Texture	32,768	Palette not used	1	16

**Note:** *Palette Base Boundary* is the amount by which the address is increased when the palette base is increased by 1 by the `TexPlttBase` command

#### 6.3.5.2.1 Texture Images

The Rendering Engine references the texture image slot's texture image in the format specified by the `TexImageParam` command. The texture image is composed of texel data.

##### 6.3.5.2.1.1 4-Color Palette Textures

#### Texel Data Format

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
T7		T6		T5		T4		T3		T2		T1		T0	
8 Texels of Data (2 bits / texel)															

- T7–T0 : Texel Data  
Specifies the texture color palette color number (0–3)

#### Display Texture

T0	T1	T2	T3	T4	T5	T6	T7

6.3.5.2.1.2 16 Color Palette Textures

Texel Data Format

15	12	11	8	7	4	3	0
T3		T2		T1		T0	
4 Texels of Data (4 bits / texel)							

- T3–T0 : Texel Data  
Specifies the texture color palette color number (0–15).

Display Texture

T0	T1	T2	T3				

6.3.5.2.1.3 256-Color Palette Textures

Texel Data Format

15	8	7	0
T1		T0	
2 Texels of Data (8 bits / texel)			

- T1–T0 : Texel Data  
Specifies the texture color palette color number (0–255).

Display Texture

T0	T1						

6.3.5.2.1.4 4x4 Texel Compression Textures

This format can obtain the compression effect by dividing the image into 4x4 pixel blocks, and then converting them to images with palettes with 2-bit indexes.

Texel Data Format

31	24			23	16			15	8			7	0		
T33	T32	T31	T30	T23	T22	T21	T20	T13	T12	T11	T10	T03	T02	T01	T00
4 x 4 Texels of Data (2 bits / texel)															

- T33–T00 : Texel Data  
Specifies the color number (0–3).

Display Texture

T00	T01	T02	T03				
T10	T11	T12	T13				
T20	T21	T22	T23				
T30	T31	T32	T33				



### Palette Index Data Format

15	14	13					8	7							0
A	PTY														
Palette Settings		Palette Address													

- **Palette settings**

- **A : 3 colors / 4 colors setting flag**

<b>0</b>	3 colors + Transparent mode (Color 3 is transparent color)
<b>1</b>	4-color mode

- **PTY : Palette type selection flag**

<b>0</b>	4-color palettes (Four palettes for each 4x4 texels)
<b>1</b>	Linear interpolation 4-color palettes (Two palettes for each 4x4 texels)

- **Palette address**

The address of the texture palette slot in which color data is stored is specified in units of 2 colors (units of 4 bytes).

Color 0 is the color located at the texture palette slot address, calculated as follows:

(The value set by the TexPlttBase command x 0x10) + (the palette address setting value x 4)

The texel color value RGB components are calculated as shown in Table 6-17.

**Table 6-17 : Texel Color Values**

		<b>PTY=0</b>
<b>PTY=0</b>	<b>A=0</b>	Color 0[5:0]:(Palette 0[4:0]==0) ? (Palette 0[4:0] x 2) : (Palette 0[4:0] x 2 + 1) Color 1[5:0]:(Palette 1[4:0]==0) ? (Palette 1[4:0] x 2) : (Palette 1[4:0] x 2 + 1) Color 2[5:0]:(Palette 2[4:0]==0) ? (Palette 2[4:0] x 2) : (Palette 2[4:0] x 2 + 1) Color 3[5:0]:Transparent Color
	<b>A=1</b>	Color 0[5:0]:(Palette 0[4:0]==0) ? (Palette 0[4:0] x 2) : (Palette 0[4:0] x 2 + 1) Color 1[5:0]:(Palette 1[4:0]==0) ? (Palette 1[4:0] x 2) : (Palette 1[4:0] x 2 + 1) Color 2[5:0]:(Palette 2[4:0]==0) ? (Palette 2[4:0] x 2) : (Palette 2[4:0] x 2 + 1) Color 3[5:0]:(Palette 3[4:0]==0) ? (Palette 3[4:0] x 2) : (Palette 3[4:0] x 2 + 1)
<b>PTY=1</b>	<b>A=0</b>	Color 0[5:0]:Palette 0[4:0] x 2 Color 1[5:0]:Palette 1[4:0] x 2 Color 2[5:0]:Palette 0[4:0] + Palette 1[4:0] Color 3[5:0]:Transparent Color
	<b>A=1</b>	Color 0[5:0]:Palette 0[4:0] x 2 Color 1[5:0]:Palette 1[4:0] x 2 Color 2[5:0]:(Palette 0[4:0] x 5 + Palette 1[4:0] x 3) / 4 Color 3[5:0]:(Palette 0[4:0] x 3 + Palette 1[4:0] x 5) / 4

One set of palette index data is associated with each 4x4 texel (see "[Figure 6-42 : Texture Image Slots](#)" on page 252).

## Texture Image Slots

In 4x4 texel compression mode, the texture image data and the texture palette index data should be mapped as follows:

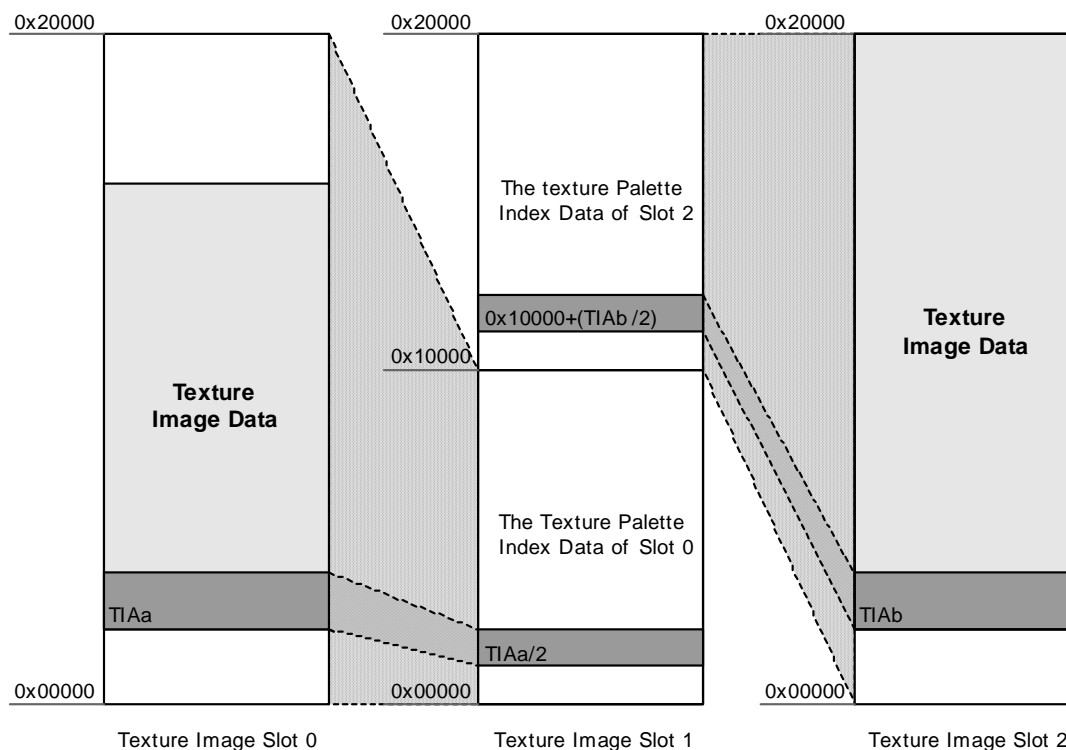
- Texel data  
Map to texture image slots 0 and 2.
- Texture palette index data  
Map to texture image slot 1.
- Correspondence between texel data and texture palette index data

The texture palette index data which corresponds to the 4x4 texel data in the TIDAa address of texture image slot 0 should be placed in the (TIDAa/2) address of texture image slot 1.

The texture palette index data which corresponds to the 4x4 texel data in the TIDAb address of texture image slot 2 should be placed in the (0x10000 + (TIDAb/2)) address of texture image slot 1. (See Figure 6-42.)

Each 4x4 texel is associated with one set of texture palette index data. Individual texel colors take the color of the address (the texture color palette slot) specified by the texture palette index data as 0, and use the color number's color designated by the texel data.

**Figure 6-42 : Texture Image Slots**



**Note:** Texture data is placed in slot 0 and slot 2. Because the address jumps, it is not possible to reference 1024x1024. In 4x4-texel compressed-texture mode, use a texture size that is no larger than 1024x512.

## 6.3.5.2.1.5 A3I5 Translucent Textures

## Texel Data Format

15	13	12	8	7	5	4	0
ALPHA				INDEX			
T1				T0			

- T1–T0 : Texel data

- ALPHA :  $\alpha$ -value

Specifies that the degree of transparency for texel. 0 is set to Transparent.

The  $\alpha$ -value is used in the Rendering Engine, extended in 5 bits as shown in the table below.

(5-bit  $\alpha = \{(3 \text{ bit } \alpha \ll 2) + (3 \text{ bit } \alpha \gg 1)\}$ )

3 bit $\alpha$	0	1	2	3	4	5	6	7
5 bit $\alpha$	0	4	9	13	18	22	27	31

- INDEX : Color number

Specifies the color number (0–31) of the texture color palette.

## Display Texture

T0	T1						

## 6.3.5.2.1.6 A5I3 Translucent Textures

## Texel Data Format

15	11	10	8	7	3	2	0
ALPHA				INDEX			
T1				T0			

- T1–T0 : Texel data

- ALPHA :  $\alpha$ -value

Specifies that the degree of transparency for texel. 0 is set to Transparent.

- INDEX : Color number

Specifies the color number (0–7) of the texture color palette.

## Display Texture

T0	T1						

### 6.3.5.2.1.7 Direct Color Textures

#### Texel Data Format

15	14		10	9	8	7	5	4	0
ALPHA	BLUE				GREEN				RED
T0									

- T0 : Texel data

Directly configures texel color. The texture color palette is not used.

#### Display Texture

T0							

### 6.3.5.2.2 Texture Palette

The texture palette slot stores the texture color data.

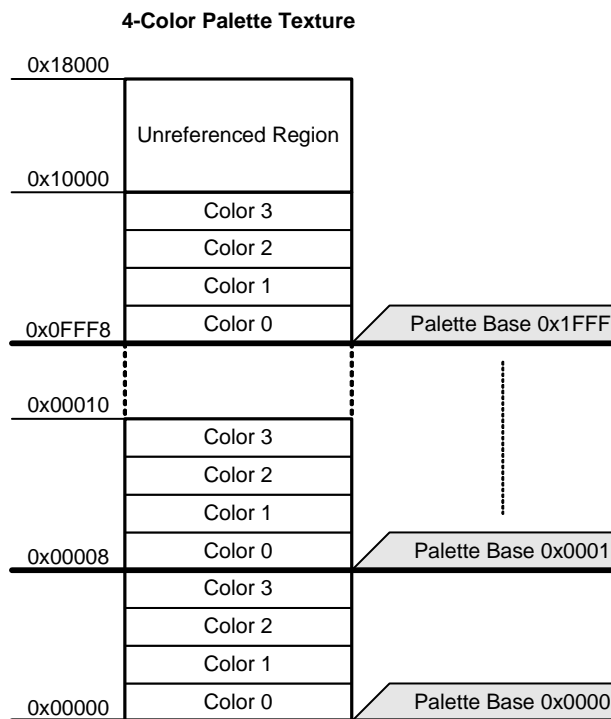
#### Texture Color Data Format

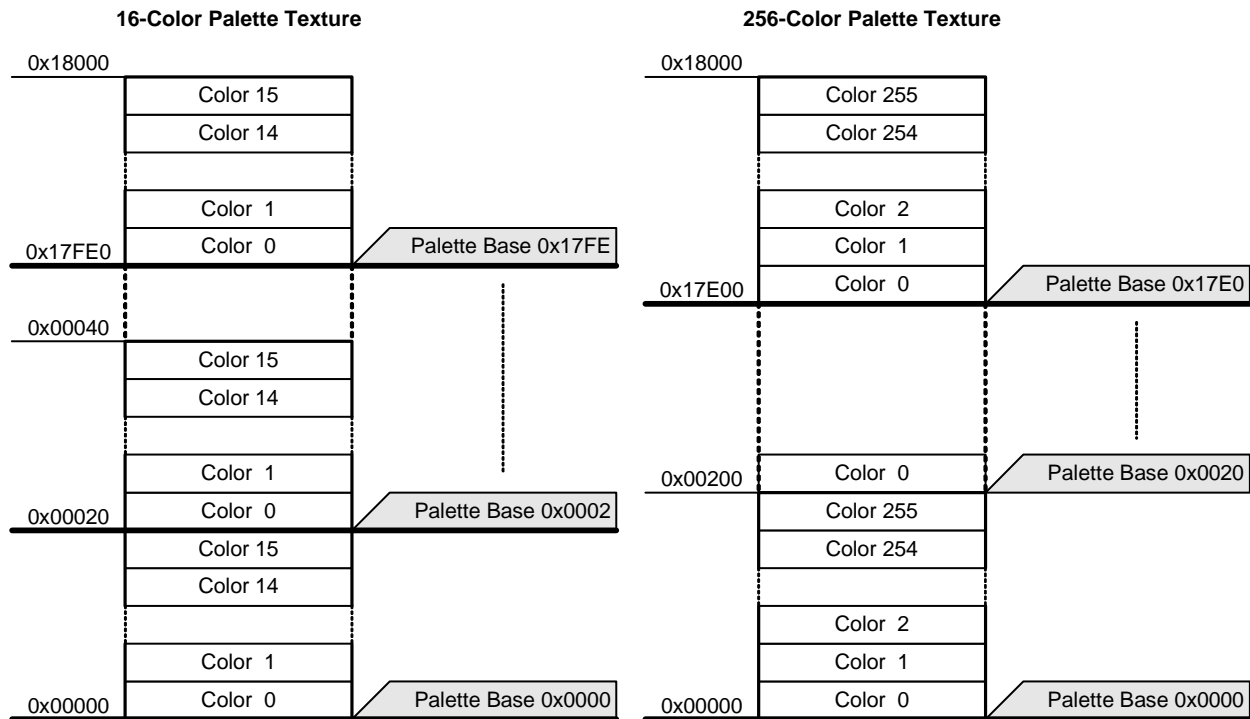
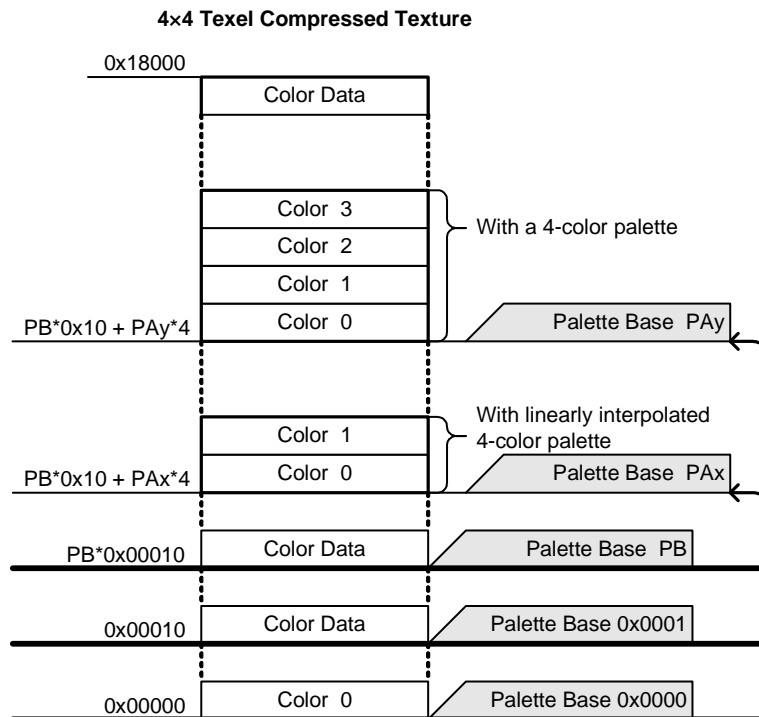
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	TEX_COLOR_BLUE					TEX_COLOR_GREEN					TEX_COLOR_RED				
	Color														

The texture format determines the way in which texture color palettes are referenced. (The texture formats are: 4-color, 16-color, 256-color, 4x4 compressed-textel, A3I5, and A5I3.)

Generally, the color number is referenced for the color determined by the texel data from the palette specified by the texture palette base. However, when the texture is a 4x4 compressed-textel-format texture, the palette is specified by setting the palette address, in addition to the texture palette base. See Figure 6-43 to Figure 6-46 for details on the palette base for each texture color palette and palette address mapping.

**Figure 6-43: Palette Base and Palette Address (4-color palette)**

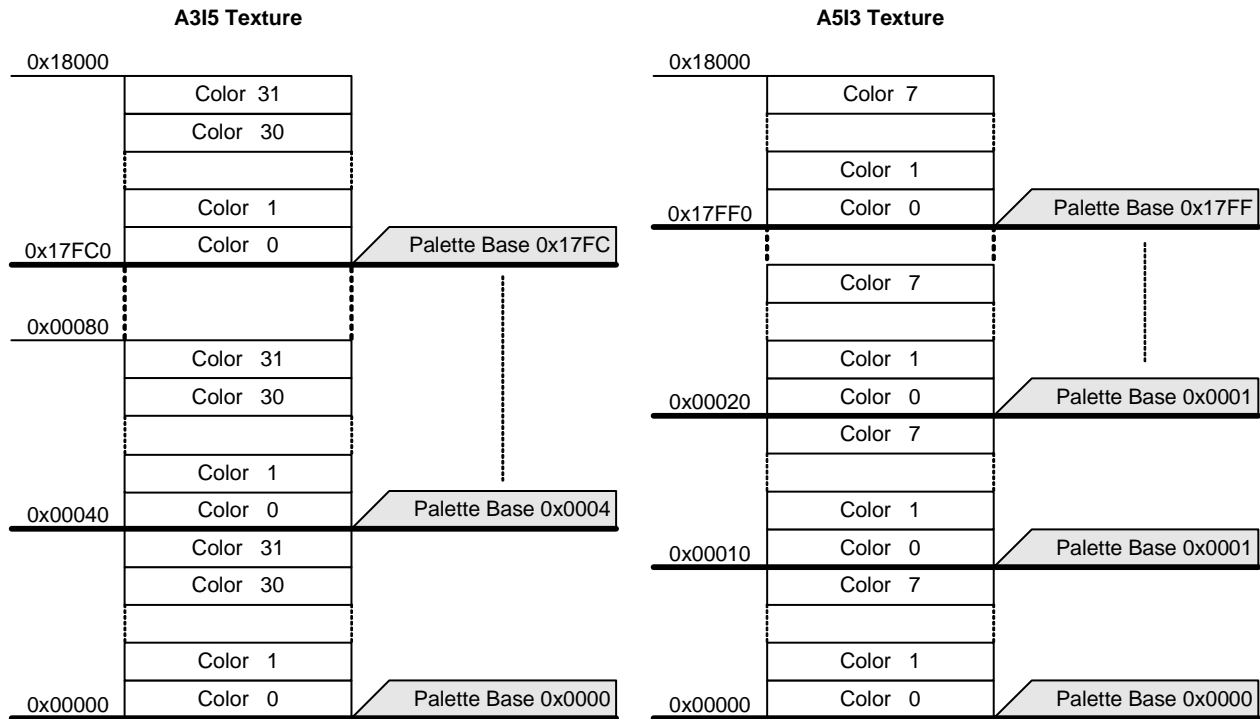


**Figure 6-44: Palette Base and Palette Address (16-Color Palette and 256-Color Palette)****Figure 6-45: Palette Base and Palette Address (4x4 Texel Compression)**

For a 4x4 compressed-textel-format texture, the palette address is set by the palette base and the palette address. Color numbers 0 to 3 are referenced for a four-color palette, but only color numbers 0 and 1 are referenced for a linear interpolated four-color palette. The remaining two colors applied to the texels are calculated from Color 0 and Color 1.

Even for a four-color palette, Transparent is used without referencing Color 3 when the mode is set to Three Colors + Transparent.

**Figure 6-46: Palette Base and Palette Address (A3I5, A5I3)**



### 6.3.6 Alpha-Test

After texture blending, the fragment's alpha value is compared with the value set in the AlphaReference register. Rendering does not occur when the  $\alpha$  value is less than this reference value.

#### AlphaReference: Alpha-Test Comparison Value Register

Name: ALPHA\_TEST\_REF

Address: 0x04000340

Attribute: W

Initial value: 0x0000

15								8	7			4			0
															ALPHA_REFERENCE
															$\alpha$ -Test Comparison Value

- ALPHA\_REFERENCE[d04–d00] :  $\alpha$ -Test comparison value

When  $\alpha$ -Test is set to ON, pixels that have an  $\alpha$  value below this specified value are not drawn.

- Wireframe display polygons

Setting  $\alpha = 0$  in the polygon attributes displays polygons as wireframes— $\alpha$  no longer retains its original meaning.

If translucent textures are unmapped, the wireframe section actually has an  $\alpha$  value of 31.

Therefore, if the  $\alpha$  test reference level is set when the  $\alpha$  test is ON, when displaying a wireframe, it is displayed for any reference value other than 31.

### 6.3.7 Alpha-Blending

The specifications call for the Rendering Engine to perform alpha-blending by first creating the 3D screen before alpha-blending with the 2D screen. Alpha-blending with the 2D screen is performed using the 2D graphics color special effect functions.

You can control the Rendering Engine's alpha-blending process by setting the DISP3DCNT register's  $\alpha$ -blending enable flag on/off.

Table 6-18 shows the equation used when alpha-blending.

**Table 6-18: Equation when  $\alpha$ -Blending**

$\alpha$ -Blending Enable Flag	Calculations for Newly Stored Data in Color Buffer
<b>When ON</b>	$R = \{(As + 1) \times Rs + (31 - As) \times Rb\} / 32$ $G = \{(As + 1) \times Gs + (31 - As) \times Gb\} / 32$ $B = \{(As + 1) \times Bs + (31 - As) \times Bb\} / 32$ $A = \max[As, Ab]$ <p>Handling exceptions:            When <math>As = 0</math>, <math>(R, G, B, A) = (Rb, Gb, Bb, \max[As, Ab])</math> is used.            When <math>As = 31</math> or <math>Ab = 0</math>, <math>(R, G, B, A) = (Rs, Gs, Bs, \max[As, Ab])</math> is used.</p>
<b>When OFF</b>	When $As = 0$ $(R, G, B, A) = (Rb, Gb, Bb, Ab)$ When $As$ is non-zero $(R, G, B, A) = (Rs, Gs, Bs, As)$

$(R, G, B, A)$  : Newly written fragment color (fractional parts resulting from calculations are truncated)

$(Rb, Gb, Bb, Ab)$  : Color buffer's color

$(Rs, Gs, Bs, As)$  : Fragment color



### 6.3.7.1 3D Alpha-blending

The color buffer's color value and the fragment's color value are blended based on the fragment's  $\alpha$  value, and the result is then written back to the color buffer.

### 6.3.7.2 2D and 3D Alpha-Blending Preprocess

In the 3D alpha-blending process described above, the color buffer's  $\alpha$  value is updated only when it is smaller than the fragment's  $\alpha$  value. This specification is set in order to approximate the  $\alpha$  value in regions where translucent polygons overlap when alpha-blending 2D and 3D.

When the color buffer's alpha value is 0 and translucent polygons are drawn, the fragment's color is written to the color buffer without any alpha-blending.

You can thus achieve a more natural composite with the 2D screen by using ClearColor to zero-clear the color buffer's alpha value, since the ClearColor's RGB values are not blended.

Refer to "[6.4.4.1 Alpha-Blending with the 2D Screen](#)" on page 270 for details on this subject.

### 6.3.8 Edge Marking

Edge marking is a feature for outlining the edges of opaque polygons with different polygon IDs in the Attribute buffer. You can control this feature by setting the DISP3DCNT register's edge-marking enable flag on or off.

The colors that are used in edge marking are the eight colors that are selected using the upper 3 bits of the polygon ID as an index.

#### EdgeColor: Edge Color Register

Name	Address	Attribute	Initial Value
EDGE_COLOR_x (x =0-3)	0x04000330, 0x04000334, 0x04000338, 0x0400033C	W	0x00000000

31	30	26	25	24	23	21	20	16	15	14	10	9	8	7	5	4	0
BLUE n1		GREEN n1		RED n1				BLUE n0		GREEN n0		RED n0					
Edge marking color when polygon ID (n1)=2x+1								Edge marking color when polygon ID (n)=2x									

- [d30–d16], [d14–d00] : Edge marking colors

Specifies the eight colors to employ for edge marking.

If a polygon is clipped, edge marking is also applied to the *clipping boundary* (a new edge created by clipping) A comparison to clear polygon ID is also made at the edge of the screen. Therefore, if the clear polygon ID and the polygon ID are the same, edge marking is not applied to the new edges created by clipping.

**Note:** Edge marking of opaque polygons in the background behind translucent polygons sometimes does not work correctly when the PolygonAttr command has set the Translucent Polygon Depth Value Update–enable flag to 1 (because the Depth buffer's value is referenced when determining edge marking).



**FogTable: Fog Density Table Register**

Name	Address	Attribute	Initial Value
FOG_TABLE_x (x=0-7)	0x04000360, 0x04000364, 0x04000368, 0x0400036C, 0x04000370, 0x04000374, 0x04000378, 0x0400037C	W	0x00000000

31	30	24	23	22	16	15	14	8	7	6	0
DENSITY n3		DENSITY n2		DENSITY n1		DENSITY n0					
Fog Density n3=4x+3		Fog Density n2=4x+2		Fog Density n1=4x+1		Fog Density n0=4x					

Specifies a 32-level fog density table.

The fog density for each pixel is the value that results from linear interpolation of the corresponding depth buffer value. You can thus approximate any fog density curve (see "[Figure 6-47 : Depth Values and Fog Density](#)" on page 262).

**6.3.9.1.1 Fog Density Equations**

Assume that  $F\_IVL = (0x400 \gg FOG\_SHIFT)$  and that  $f(i)$  is the  $i^{th}$  parameter in the fog density table. Then the fog density and the upper 15-bit depth value ( $Zd$ ) have the following relationship:

1. When  $0x0000 \leq Zd \leq (FOG\_OFFSET + F\_IVL - 1)$

Fog density  $f = f(0)$

2. When  $(FOG\_OFFSET + F\_IVL \times i) \leq Zd \leq (FOG\_OFFSET + F\_IVL \times (i + 1) - 1)$  ( $i = 1-31$ )

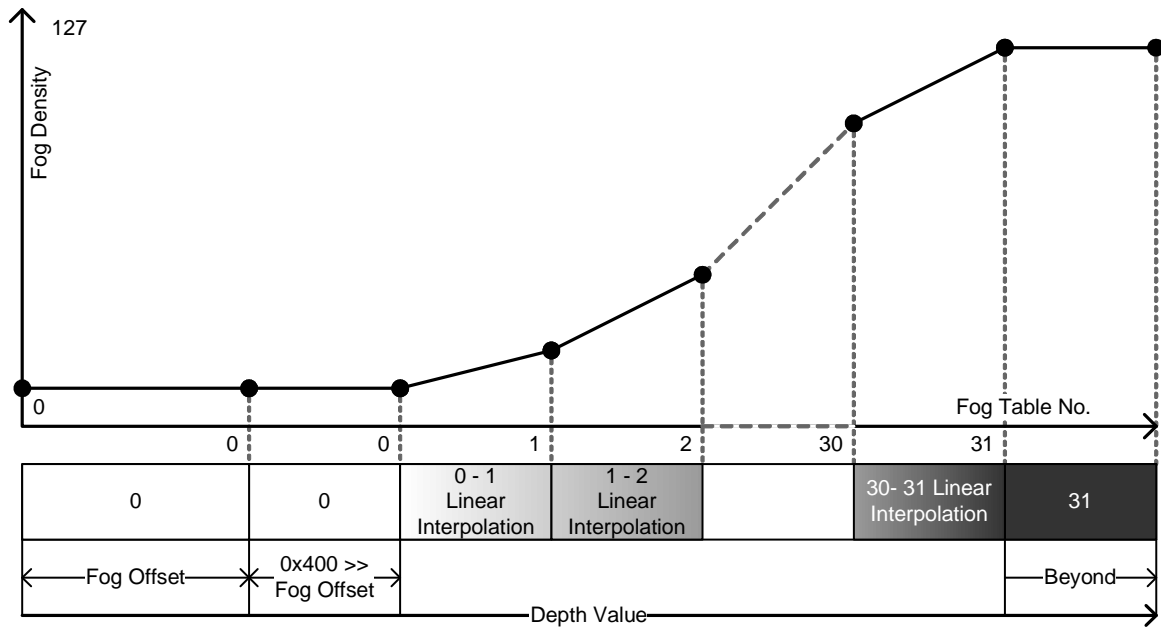
Fog density  $f =$

$$f = \frac{\{f(i) - f(i-1)\} \times \{Zd - (FOG\_OFFSET + F\_IVL \times i)\}}{F\_IVL} + f(i-1)$$

3. When  $FOG\_OFFSET + F\_IVL \times 32 \leq Zd \leq 0x7FFF$

Fog density  $f = f(31)$

Figure 6-47: Depth Values and Fog Density



The fog shift is set by the DISP3DCNT register (the 3D Display Control register).

Table 6-19 shows the fog-blending equations.

**Table 6-19: Fog-Blending Equations**

Fog Mode	Fog Blending with Pixel's Color and $\alpha$ Value	Fog Blending only with Pixel's $\alpha$ Value
<b>Fog Blending Equations</b>	$R = \{ f \times R_f + (128 - f) \times R_s \} / 128$ $G = \{ f \times G_f + (128 - f) \times G_s \} / 128$ $B = \{ f \times B_f + (128 - f) \times B_s \} / 128$ $A = \{ f \times A_f + (128 - f) \times A_s \} / 128$ <p>Handling exceptions: When <math>f = 127</math>, (R, G, B, A) = (Rf, Gf, Bf, Af) is used</p>	$R = R_s$ $G = G_s$ $B = B_s$ $A = \{ f \times A_f + (128 - f) \times A_s \} / 128$ <p>Handling exceptions: When <math>f = 127</math>, A = Af is used</p>

(R, G, B, A) : Newly written fragment color (fractional parts resulting from calculations are truncated)

(Rf, Gf, Bf, Af) : Fog color

(Rs, Gs, Bs, As) : Color in the color buffer after edge marking

### 6.3.9.2 Fog Preprocessing for 2D

If fog is disabled, the region in which the color buffer's  $\alpha$  value is zero-cleared is treated as an alpha cut-out region when compositing with the 2D screen. From this region, you can see the unmodified color of the 2D screen in the background. (See ["6.4 2D Graphics Features you can Apply to the 3D Screen after Rendering"](#) on page 268.)

If, on the other hand, fog is enabled, this otherwise transparent region is also subject to fog blending, and the color buffer value is updated. Afterwards, the 2D color special-effect feature (2D and 3D alpha-blending) can work to alpha-blend the color buffer and the 2D screen, so fog is also applied to the 2D screen in the background seen from this region.

In this way, 2D fog blending is conducted via 2D and 3D alpha-blending. (See ["6.3.7.2 2D and 3D Alpha-Blending Preprocess"](#) on page 259.)

### 6.3.10 Anti-aliasing

The anti-aliasing feature blends an edge section in the front buffer with the color in the back color buffer. The front color buffer holds the rendered results of the front polygon; the back buffer stores the rendered results of all polygons (including clear colors) that are behind the front polygon. Anti-aliasing is only applied to the edges of opaque polygons. Anti-aliasing uses the  $\alpha$  value that is newly written into the color buffer as a blending factor when alpha-blending with 2D. Therefore, even a 2D background exhibits the anti-aliasing effect.

Table 6-20 shows the anti-aliasing equations.

Figure 6-48 illustrates the concept of anti-aliasing. Figure 6-49 shows the edge that is output to the LCD.

**Table 6-20: Anti-aliasing Equations**

	Anti-aliasing Equations
<b>A2 at least 1</b>	$RA = \{ (AC + 1) \times R1 + (31 - AC) \times R2 \} / 32$ $GA = \{ (AC + 1) \times G1 + (31 - AC) \times G2 \} / 32$ $BA = \{ (AC + 1) \times B1 + (31 - AC) \times B2 \} / 32$ $AA = \{ (AC + 1) \times 31 + (31 - AC) \times A2 \} / 32$
<b>A2 = 0</b>	$RA = R1$ $GA = G1$ $BA = B1$ $AA = \{ (AC + 1) \times 31 \} / 32$

(RA, GA, BA, AA) : Color newly stored in the color buffer (fractional parts resulting from calculations are truncated).

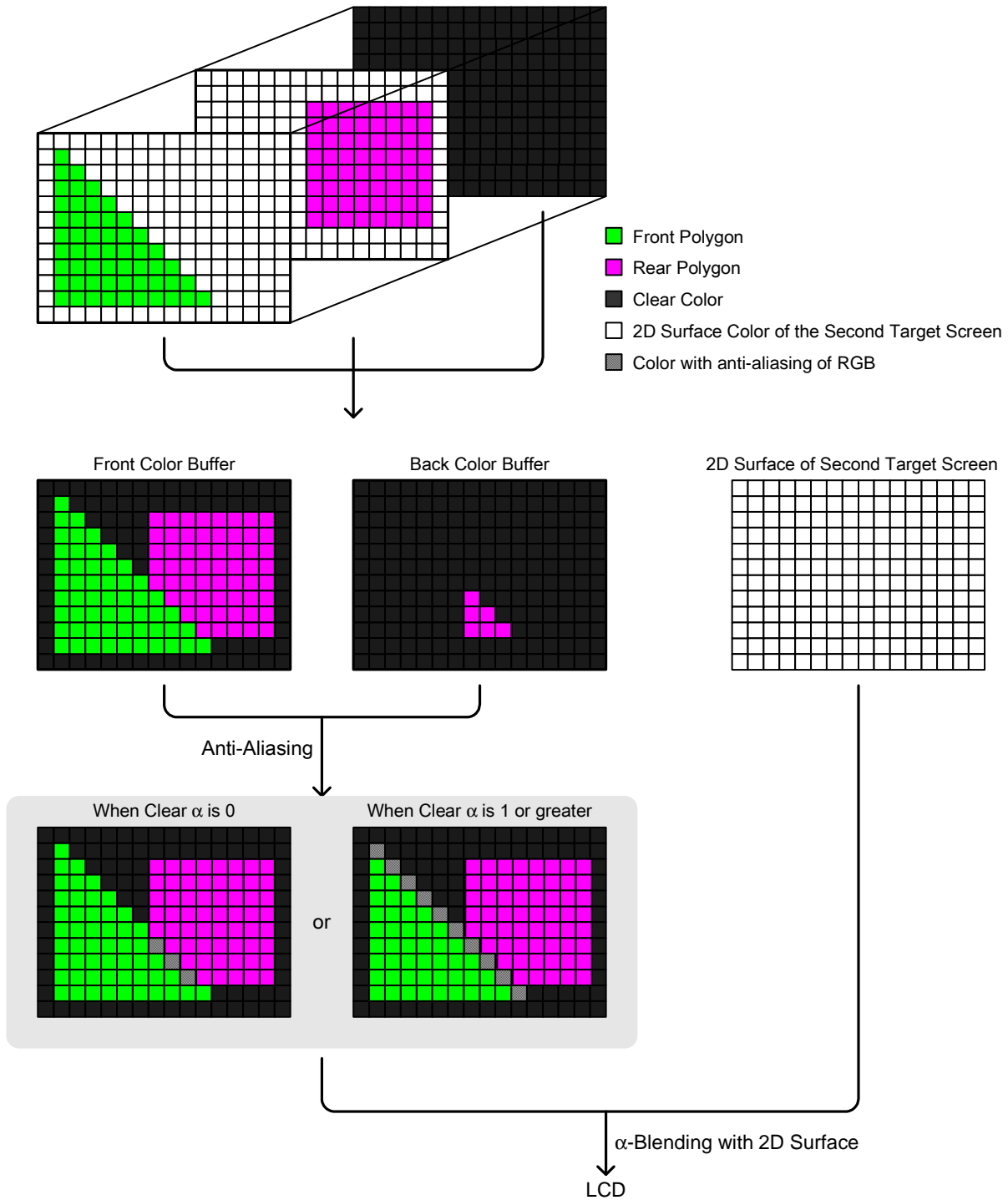
(R1, G1, B1, 31) : Color in the front color buffer (alpha = 31 since the anti-aliasing target is an opaque polygon).

(R2, G2, B2, A2) : Color in the back color buffer

AC : Anti-aliasing factor (5 bits)

The anti-aliasing factor is applied proportionately to the surface area of the pixel occupied by the polygon.

Figure 6-48: Anti-aliasing



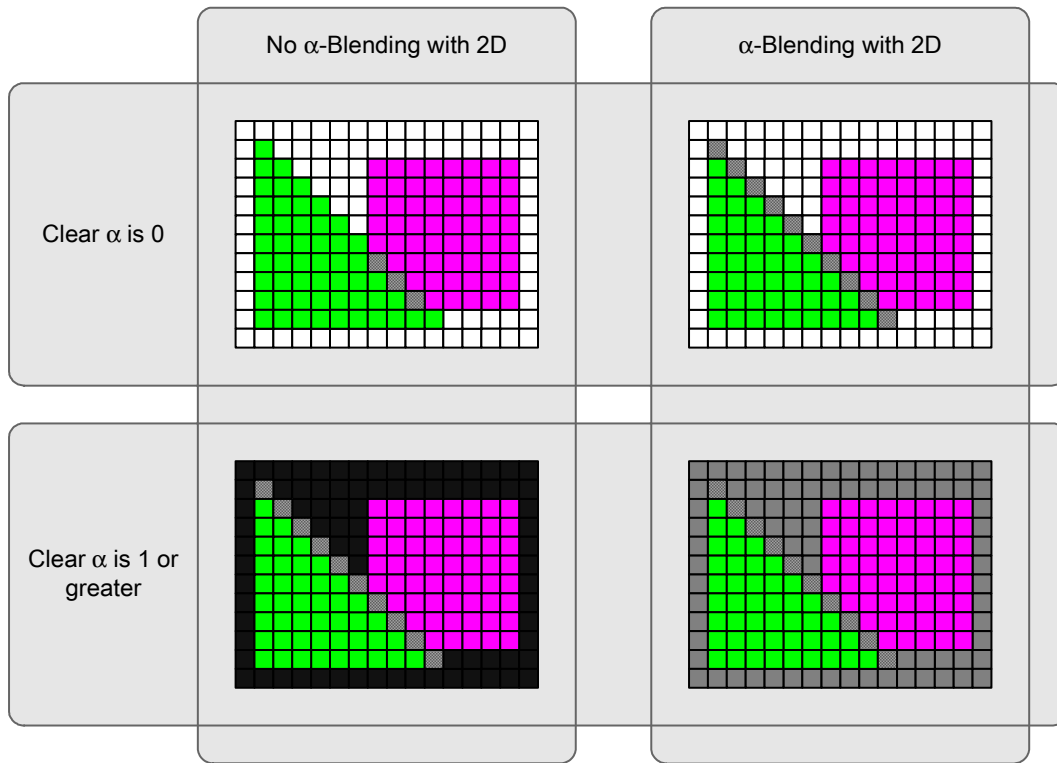
**Figure 6-49: Final LCD Image Output (Anti-Aliasing)**

Figure 6-49 shows the visual image. Table 6-21 shows the actual RGB results

**Table 6-21: Anti-Aliasing and Alpha-Blending with a 2D Surface**

A2	$\alpha$ Blending with 2D Surface	Anti-aliasing Results (RGBA)		Result of $\alpha$ Blending with 2D Surface (RGB)	
		Edge	Background	Edge	Background
0	No	(R1, G1, B1, AA)	(R2, G2, B2, 0)	(R1, G1, B1)	Color of 2D surface
0	Yes	(R1, G1, B1, AA)	(R2, G2, B2, 0)	$\alpha$ -blended color of (R1, G1, B1) and 2D surface ( $\alpha$ -blending factor AA)	Color of 2D surface
1 or More	No	(RA, GA, BA, AA)	(R2, G2, B2, A2)	(RA, GA, BA)	(R2, G2, B2)
1 or More	Yes	(RA, GA, BA, AA)	(R2, G2, B2, A2)	$\alpha$ -blended color of (RA, GA, BA) and 2D surface ( $\alpha$ -blending factor AA)	$\alpha$ -blended result of (R2, G2, B2) and 2D surface ( $\alpha$ -blending factor A2)

Because the interior of a polygon is opaque (alpha = 31), there is no alpha-blending with the 2D. Therefore, this case is omitted.

Table 6-21 shows how, when A2 is 0, anti-aliasing is applied only to alpha values. RGB color is written to the color buffer as-is, with no blending.

In this case, do alpha-blending with the 2D surface, and reflect this in RGB. (The alpha value from anti-aliasing is applied to alpha-blending.)



### 6.3.11 Status

#### Rendered Line Count Register

Name: RDLINES\_COUNT

Address: 0x04000320

Attribute: R

Initial value: 0x0000

15								8	7		5			0
														RENDERED_LINES_MIN
														Minimum value for number of rendered lines

- **RENDERED\_LINES\_MIN[d05–d00]** : Minimum value for number of rendered lines (0–46)

Use this to check the minimum number of lines in the color buffer during display of the previous frame. This register is updated during every V cycle. The color buffer holds 48 lines, but 2 lines are the current buffer, so the largest count-value for this register is 46. You cannot confirm from this value whether or not lines have overflowed, but you can determine the risk of this happening. To determine whether or not lines have overflowed, check the Color Buffer Underflow Flag of the 3D Display Control register (DISP3DCNT).

**Note:** When the counter reaches 0, there is a risk that the Rendering Engine will fail to draw lines (that is, lines will overflow). When the counter approaches 0, reduce the load on the Rendering Engine by, for example, reducing the number of polygons sent to the Geometry Engine.

#### Comparison of Rendering Buffer Methods

- **Normal frame-buffer method**

In this method, there are two or three frame buffers for rendering and display, and the buffers are swapped during the V-Blank period immediately after drawing is completed. If there are more rendering polygons and pixels than the rendering engine can process, the frame rate drops due to rendering delays.

- **FIFO line-buffer method adopted by NITRO**

In this method, drawing and display involve the same FIFO buffer. This FIFO buffer has a capacity of 48 lines. During display, data is read from the FIFO buffer in sync with the timing of the LCD. Data for the horizontal direction is read with the dot clock, whereas data for the vertical direction is read in the horizontal scanning interval (355-dot clocks). If there are more polygons and pixels for the line to be drawn than the Rendering Engine can process, the display is corrupted because the Rendering Engine cannot render it in time.

## 6.4 2D Graphics Features you can Apply to the 3D Screen after Rendering

In NITRO, the 3D screen is displayed as BG0 after it has been rendered, rather than being displayed directly on the LCD. This enables certain 2D graphic features to also be applied and displayed on the LCD. To read the basic specifications for 2D graphics see ["5 2D Graphics"](#) on page 73.

### 6.4.1 Raster Scroll

Unlike 2D screens, 3D screens cannot be scrolled vertically. However, they can be scrolled horizontally.

#### BG0 Offset Settings Register

Name: BG0OFS

Address: 0x04000010

Attribute: W

Initial value: 0x0000

15							8	7							0
							SH	INTEGER_H							
							H Offset								

Signed fixed-point number (sign + 8-bit integer)

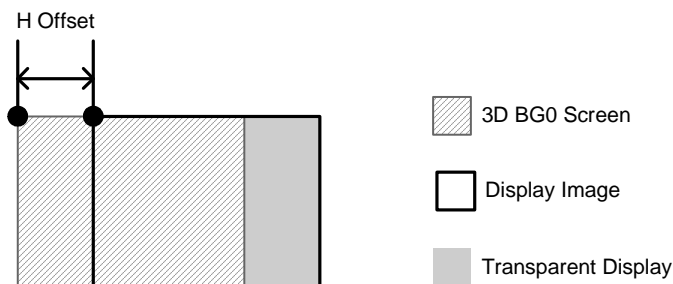
- H[d08–d00] : H offset

Changes the starting position of display in the horizontal direction.

Unlike for 2D screens, d08 is the sign bit, and the offset value can be set in the range of –256 to +256.

Portions of the display screen that go beyond the screen because of horizontal scrolling become transparent. (See Figure 6-50.)

Figure 6-50: H Offset for a 3D Surface



### 6.4.2 Order of Display Priority with a 2D Screen

#### BG0 Control Register

Name: BG0CNT (x=0, 1)

Address: 0x04000008

Attribute: R / W

Initial value: 0x0000

15							8	7							1	0
															Order of Priority	

By adjusting the display priority, you can place the 2D screen either in front of or behind the 3D screen.

See the diagram in ["5.9 Display Priority"](#) on page 151.

### 6.4.3 Windows

You can apply windows to BG0 of the 3D screen.

#### Window Position Settings Register

Name: WINxH (x=0, 1)      Address: 0x04000040, 0x04000042      Attribute: W      Initial value: 0x0000

15							8	7							0
Window's upper-left X coordinate								Window's lower-right X coordinate							

Name: WINxV (x=0, 1)      Address: 0x04000044, 0x04000046      Attribute: W      Initial value: 0x0000

15							8	7							0
Window's upper-left Y coordinate								Window's lower-right Y coordinate							

#### Window Inside Control Register

Name: WININ      Address: 0x04000048      Attribute: R / W      Initial value: 0x0000

15							8	7							0
		EFCT	OBJ	BG3	BG2	BG1	BG0			EFCT	OBJ	BG3	BG2	BG1	BG0
Window 1 Inside								Window 0 Inside							

#### Window Outside Control Register

Name: WINOUT      Address: 0x0400004A      Attribute: R / W      Initial value: 0x0000

15							8	7							0
		EFCT	OBJ	BG3	BG2	BG1	BG0			EFCT	OBJ	BG3	BG2	BG1	BG0
OBJ Window Inside								Window (0, 1 and OBJ Window) Outside							

If the 3D screen has highest priority,  $\alpha$ -blending is always enabled, regardless of the setting for the Window Control register's color-effect enable flag.

## 6.4.4 Color Effects

For details on each register parameter, see the Color Special Effects Register below.

### 6.4.4.1 Alpha-Blending with the 2D Screen

The alpha-blending feature of the 2D color effects is used for post-processing after the 2D and 3D alpha-blending process and the 2D fog process. To read about preprocessing, see ["6.3.7.2 2D and 3D Alpha-Blending Preprocess"](#) on page 259 and ["6.3.9.2 Fog Preprocessing for 2D"](#) on page 263.

#### Color Effect Control Register

Name: BLDCNT

Address: 0x04000050

Attribute: R / W

Initial value: 0x0000

15	13					8		7	6	5	0				
		BD	OBJ	BG3	BG2	BG1	BG0	0	1	BD	OBJ	BG3	BG2	BG1	BG0
		Second Target Screen					Selected Effect		First Target Screen						

- [d07–d06] : Selected effect

Perform alpha-blending by setting [d07] to 0 and [d06] to 1.

The process involved when alpha-blending the 3D screen and the 2D screen differs, depending on the relative priority of the two screens. When the 2D screen is the first target screen, the value set in the BLDALPHA register is used for alpha-blending, as per the specifications. However, when the 3D screen is the first target screen, alpha-blending with the second target screen is performed using the alpha value that is being rendered to the color buffer (that is, alpha-blending is done in units of pixels).

**Note:** Any part with a color buffer alpha value of 0 is handled in the same way as a 2D cut-out region, and thus is not subjected to alpha-blending. Any part with a color buffer alpha value of 1 or more is subjected to alpha-blending.

#### Color Special Effect / Alpha-Blending Factors Register

Name: BLDALPHA

Address: 0x04000052

Attribute: R / W

Initial value: 0x0000

15	12				8			7	4			0		
			EVB								EVA			

The factors used for the alpha-blending process are set by EVA and EVB in the BLDALPHA register. (EVA and EVB values that are 16 or above are treated as 16.)

### 6.4.4.2 Shininess Up/Down

#### Color Effect Control Register

Name: BLDCNT

Address: 0x04000050

Attribute: R / W

Initial value: 0x0000

15		13					8		7	6	5					0	
		BD	OBJ	BG3	BG2	BG1	BG0	1		BD	OBJ	BG3	BG2	BG1	BG0		
		Second Target Screen						Selected Effect		First Target Screen							

- [d07–d06] : Selection of color special effects

Perform a process to change the shininess by setting [d07] to 1.

When [d06] is set to 0, shininess is increased. When [d06] is set to 1, shininess is decreased.

All zeros must be set for the second target screen.

#### Color Special Effect / Change Shininess Factor Register

Name: BLDY

Address: 0x04000054

Attribute: W

Initial value: 0x0000

15							8		7						0
												EVY			

The factor used for changing shininess is set by EVY in the BLDY register. (EVY values that are 16 or above are treated as 16.)



## 7 DMA

DMA is a high-speed data-transfer method that bypasses the CPU. It is controlled by the DMA Controller. The ARM9 bus has four DMA channels (DMA0 – 3). (The ARM7 bus also has four channels.)

The highest priority channel is DMA0, followed by DMA1, DMA2, and DMA3, in order of priority.

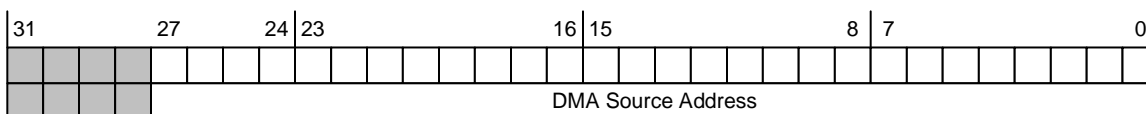
If a higher-priority DMA is activated while a lower-priority DMA is executing, the lower-priority DMA pauses, and the higher-priority DMA is executed. After the higher-priority DMA is finished, the lower-priority DMA resumes execution. Because DMA execution can be paused, consider giving higher priority to DMA transfers that must finish within a limited time frame.

However, when the CPU is operating with DMA, RAM outside the TCM or cache cannot be accessed.

Therefore, in the interval until the DMA finishes, an interrupt is delayed when processing anything other than TCM.

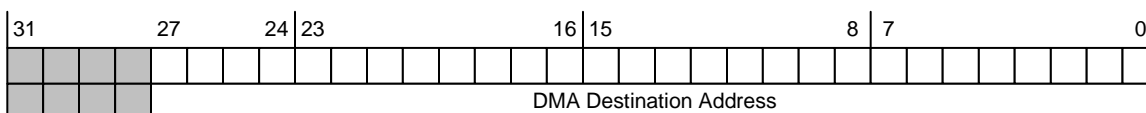
### DMAxSAD: DMAx Source Address Registers (x = 0 - 3)

Name	Address	Attribute	Initial Value
DMAxSAD (x =0 - 3)	0x040000B0, 0x040000BC, 0x040000C8, 0x040000D4	R/W	0x00000000



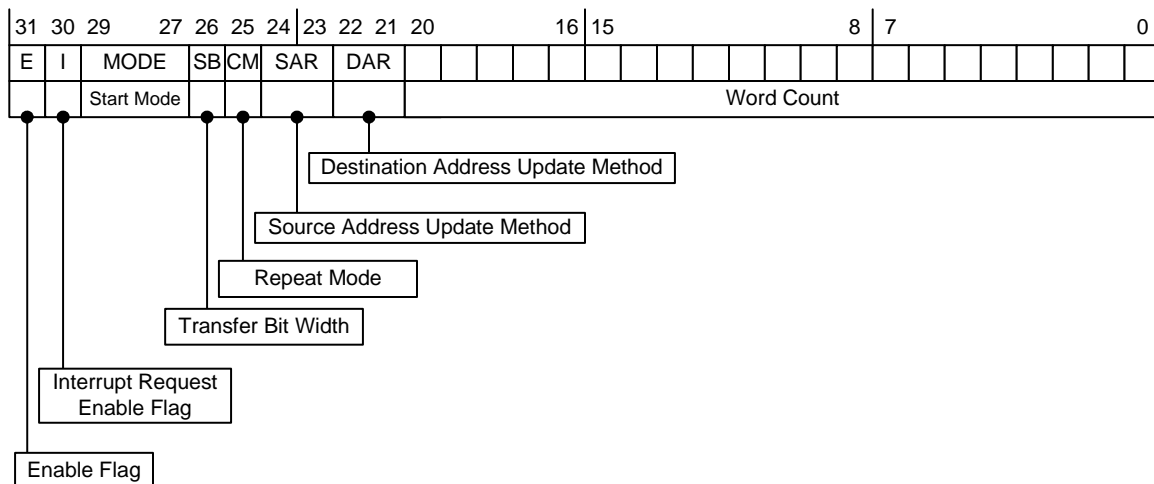
### DMAxDAD: DMAx Destination Address Registers (x = 0 - 3)

Name	Address	Attribute	Initial Value
DMAxDAD (x =0 - 3)	0x040000B4, 0x040000C0, 0x040000CC ,0x040000D8	R/W	0x00000000



**DMAxCNT: DMAx Control Registers (x=0 to 3)**

Name	Address	Attribute	Initial Value
DMAxCNT (x = 0 - 3)	0x040000B8, 0x040000C4, 0x040000D0, 0x040000DC	R/W	0x00000000



- E[d31]: DMA enable flag

0	Disable
1	Enable

- I[d30]: Interrupt request enable flag

0	Disable
1	Enable

- MODE[d29–d27]: Options for DMA start mode

000	Start immediately
001	Start at V-Blank
010	Start at H-Blank (DMA does not start during an H-Blank within a V-Blank period)
011	Synchronize to start with display (that is, synchronized to start as each horizontal line is drawn)
100	Main memory display
101	DS Game Card
110	DS Accessory
111	Geometry Command FIFO

- SB[d26]: Transfer bit-count selection

0	16 bits
1	32 bits



- CM[d25]: Repeat mode selected flag

0	Not Repeat mode
1	Repeat mode

- SAR[d24–d23]: Options to update source address

00	Increment
01	Decrement
10	Fixed
11	Setting prohibited

- DAR[d22–d21]: Options to update destination address

00	Increment
01	Decrement
10	Fixed
11	Increment/reload

- WORD\_COUNT[d20–d00]: Word count

Specifies the number of transfers.

- **Repeat mode**

When the DMA repeat mode is on, DMA starts automatically each time the start mode conditions occur.

If the repeat mode is not set, DMA stops when the transfer of the word count volume is complete.

To cancel repeat mode, set the DMA enable flag to 0, as described in Step 2 in the procedure below.

- **Address update method**

The details of processing the address update method are shown in Table 7-1.

**Table 7-1 : Processing Details for the Address Update Method**

Address Update Method	Process
Increment	The address value increases one unit with each transfer
Decrement	The address value decreases one unit with each transfer
Fixed	The address stays fixed
Increment/Reload	Increments for each transfer, and then returns to the transfer's starting address when the transfer of the word count is done

**Note:** Setting the address update method to fixed or decrement is prohibited if the source or destination is set to Game Pak space because the hardware does not support it.

- **DMA Start Mode**

Main Memory Display Start Mode

Do not set the DMA source address to any memory region outside of main memory in Main Memory Display Mode. Also, be sure to set the transfer bit mode to 32-bit and the word count value to 4.

Geometry Command FIFO Start Mode

When the Geometry Command FIFO is less than half full, DMA starts and 112 words (see the procedure below) are transferred. The process repeats until the volume transferred reaches the word count value.

**Note:** If commands have been packed, the number of words sent in each repetition equals the number of words before unpacking.

- **Procedures to Start and Stop DMA**

1. When starting DMA

A delay of 2 cycles of the system clock (33.514 Mhz) occurs from the time the DMA enable flag is set until the time DMA starts. If any of the DMA-related registers are accessed during this period, DMA might not operate correctly. To prevent a DMA problem during this period, run another process, such as inserting a dummy Load command. (The main processor executes a Load command in ½ cycle of the system clock, so you would need to insert two or more Load commands to the same register.)

2. When stopping DMA

DMA begins when the signal that serves as the start trigger is issued. If the CPU disables DMA at the same time the start trigger is issued, DMA could lock up. Therefore, be sure to disable DMA at least 4 cycles after the start trigger.

1. When the DMA repeat feature is off

Because the DMA stops automatically after it is executed once, do not forcibly clear the DMA enable flag. Instead, wait for the flag to become 0.

2. When the DMA repeat feature is on

Be sure to clear the DMA enable flag with the CPU at least 4 cycles after the signal that serves as the DMA start trigger. For example, you can safely stop DMA using the interrupt generated when DMA ends, clearing the DMA enable flag before the next start trigger is issued.

If you cannot use this method, stop DMA by using the procedure described below.

3. Stopping DMA in H-Blank or V-Blank auto-start mode

During a V-Blank period, DMA is stopped, and the start trigger is not issued, so you can safely clear the DMA enable flag at that time. If you cannot use this method, follow the procedure below:

Step 1: Write 16 bits to the DMA control register (see Table 7-2).

**Table 7-2 : Register Configuration (Step 1)**

Setting	Content
DMA Enable Flag	1 (Enable)
DMA Start Timing	00 (Start Immediately mode)
DMA Repeat Mode	0 (Disable Repeat Mode)
Other Bits	Do not change

Step 2: Carry out the process for more than four cycles.

Example: 3 NOP or 1 LDR instruction) + 1st cycle of STR instruction from Step 3 = 4 cycles

The actual writing by the STR instruction occurs in the 2nd cycle.

Step 3: Write 16 bits to the DMA control register, then stop the DMA (see Table 7-3).

**Table 7-3 : Register Configuration (Step 3)**

Setting	Content
DMA Enable Flag	0 (Disable)
DMA Start Timing	00 (Start Immediately mode)
DMA Repeat Mode	0 (Disable Repeat Mode)
Other Bits	Do not change

**Note:** DMA may run one extra time in Step 1.

### **Precautions for starting multiple, parallel DMA channels in the ARM9 System Bus**

When ARM946E-S starts accessing regions that cannot be accessed in a single system cycle (33.514 MHz), such as main memory or DS accessories, an ARM9-DMA with a lower priority (Auto) starts at the same time. The automatic startup of ARM9-DMA with a higher priority occurs immediately afterwards, and the DMA with higher priority runs out of control. This condition does not exist on ARM7 because the system bus specifications differ.

#### **Workaround**

Of the DMA Parallel Start Categories shown in Table 7-4, items in Category 3 must not be used together. In addition, start DMA from TCM. However, V-Blank start and H-Blank start can be used together.

**Table 7-4 : ARM9-DMA Parallel Start Category Chart**

DMA Parallel Start Category Number	DMA Description
1	Start immediately
2	Geometry Command FIFO (Normal)
3	Geometry Command FIFO (Auto Start) V-Blank Start (can use with H-Blank Start) H-Blank Start (can use with V-Blank Start or with multiple H-Starts) Display Synchronization Main Memory Display Game Card DS Accessory



## 8 Timer

The ARM9 bus side of NITRO comes equipped with a 4-channel, 16-bit timer.

When the timer is enabled, the Count Register counts up according to the *prescaler* (frequency divider) cycle specified with the Control Register.

An interrupt can be generated when the Count Register overflows.

If the Count Register overflows, the value set when the count began is loaded, and the count starts over.

### TM0CNT\_L: Timer 0 Count Register

Name: TM0CNT\_L

Address: 0x04000100

Attribute: R/W

Initial Value: 0x0000

15								8		7								0
Timer 0 Counter																		

- [d15–d00]: Timer 0 Counter

### TM0CNT\_H: Timer 0 Control Register

Name: TM0CNT\_H

Address: 0x04000102

Attribute: R/W

Initial Value: 0x0000

15								8		7	6					1	0
										E	I						PS
																	Prescaler

- [d07–d00]: Timer 0 Control
  - E[d07]: Timer 0 Enable Flag

0	Disable
1	Enable

- I[d06]: Interrupt Request Enable Flag

0	Disable
1	Enable

- PS[d01–d00]: Prescaler Selection Flag

00	System Clock (33.514 Mhz)
01	1/64 of System Clock
10	1/256 of System Clock
11	1/1024 of System Clock

**TMxCNT\_L: Timer x Count Register (x = 1 – 3)**

Name: TMxCNT\_L (x=1-3) Address: 0x04000104,0x04000108,0x0400010C Attribute: R/W Initial Value: 0x0000

15							8	7						0
Timer x Counter														

- [d15–d00]: Timer x Counter

**TMxCNT\_H: Timer x Control Register (x = 1 – 3)**

Name: TMxCNT\_H (x=1-3) Address: 0x04000106,0x0400010A,0x0400010E Attribute: R/W Initial Value: 0x0000

15							8	7	6				2	1	0
								E	I				CH	PS	
													Multi-Stage	Prescaler	

- [d07–d00]: Timer x Control

- E[d07]: Timer x Enable Flag

0	Disable
1	Enable

- I[d06]: Interrupt Request Enable Flag

0	Disable
1	Enable

- CH[d02]: Multistage Counter Selection Flag

0	According to Prescaler setting
1	Counts up when timer (x-1) overflows regardless of Prescaler setting

- PS[d01–d00]: Prescaler Selection Flag

00	System Clock (33.514Mhz)
01	1/64 of System Clock
10	1/256 of System Clock
11	1/1024 of System Clock

## 9 Interrupts

This chapter describes the hardware interrupts for the ARM9 main processor.

When an interrupt request signal occurs from each piece of hardware, the bit that supports the interrupt request register is set, and if interrupts are enabled, the CPU is informed of the interrupt occurrence.

Each hardware interrupt request signal can be disabled individually via the interrupt enable register.

### 9.1 Interrupt Master Enable Register

This register can disable registers as a whole, and it configures whether to disable all registers or to enable the interrupt enable register settings.

#### IME: Interrupt Master Enable Register

Name: IME                      Address: 0x04000208                      Attribute: R/W                      Initial Value: 0x0000

15								8		7							0
																	IME

- IME[d00]: Interrupt Master Enable Flag

0	Disable all interrupts
1	Enable the Interrupt Enable Register settings

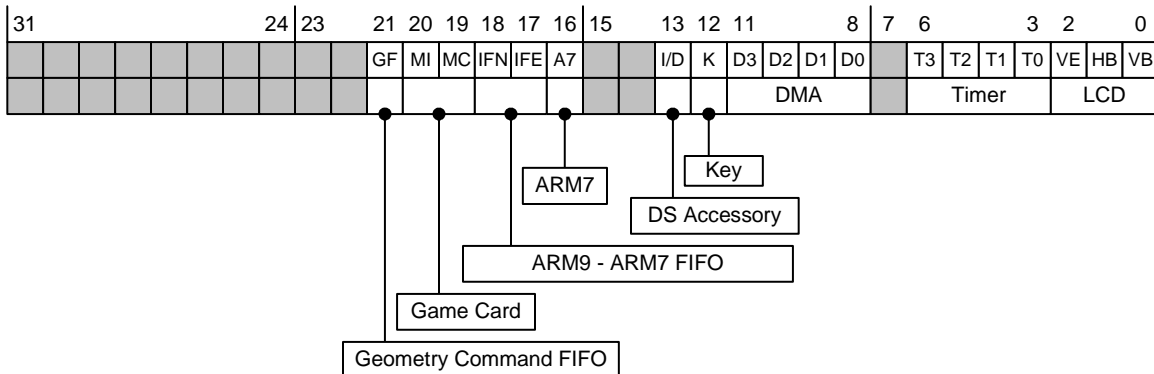
## 9.2 Interrupt Enable Register

Each hardware interrupt request can be disabled individually.

Setting each bit enables interrupt requests from the corresponding hardware. Conversely, interrupt requests from corresponding hardware are disabled when the bit is reset.

### IE: Interrupt Enable Register

Name: IE      Address: 0x04000210      Attribute: R/W      Initial Value: 0x00000000



- GF[d21]: Geometry Command FIFO Interrupt Permission Flag  
For more information, see ["6.2.16 Status"](#) on page 220.
- MI[d20]: NITRO Card IREQ\_MC Interrupt Permission Flag
- MC[d19]: NITRO Card Data Transfer Completion Interrupt Permission Flag
- IFN[d18]: ARM9 – ARM7 FIFO Not Empty Interrupt Permission Flag
- IFE[d17]: ARM9 – ARM7 FIFO Empty Interrupt Permission Flag
- A7[d16]: ARM7 Interrupt Permission Flag
- I/D[d13]: DS Accessory IREQ/DREQ Interrupt Permission Flag
- K[d12]: Key Interrupt Permission Flag  
For more information, see ["12.2 Interrupt Handling for Key Input"](#) on page 298.
- D3[d11]: DMA3 Interrupt Permission Flag
- D2[d10]: DMA2 Interrupt Permission Flag
- D1[d09]: DMA1 Interrupt Permission Flag
- D0[d08]: DMA0 Interrupt Permission Flag  
For more information, see [Chapter 7](#).
- T3[d06]: Timer 3 Interrupt Permission Flag
- T2[d05]: Timer 2 Interrupt Permission Flag
- T1[d04]: Timer 1 Interrupt Permission Flag
- T0[d03]: Timer 0 Interrupt Permission Flag  
For more information, see [Chapter 8](#).
- VE[d02]: V-Counter Match Interrupt Permission Flag
- HB[d01]: H-Blank Interrupt Permission Flag
- VB[d00]: V-Blank Interrupt Permission Flag  
For more information, see ["4.3 Display Status"](#) on page 51.



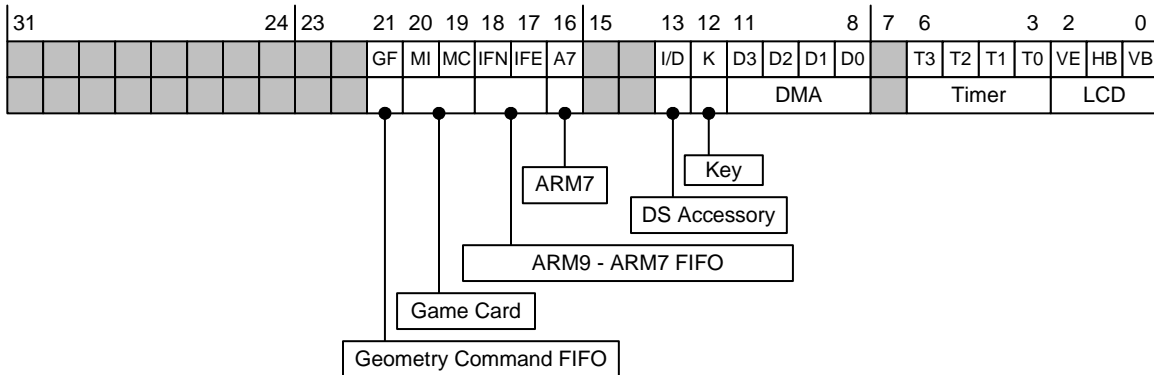
### 9.3 Interrupt Request Register

When an interrupt request from a hardware component occurs, the corresponding bit for the hardware component is set in the interrupt request register.

Also, if 1 is written to the bit where the interrupt request flag is set, the interrupt request flag is reset.

#### IF: Interrupt Request Register

Name: IF Address: 0x04000214 Attribute: R/W Initial Value: 0x00000000



- GF[d21]: Geometry Command FIFO Interrupt Request Flag  
For more information, see ["6.2.16 Status"](#) on page 220.
- MI[d20]: NITRO Card IREQ\_MC Interrupt Request Flag
- MC[d19]: NITRO Card Data Transfer Completion Interrupt Request Flag
- IFN[d18]: ARM9 – ARM7 FIFO Not Empty Interrupt Request Flag
- IFE[d17]: ARM9 – ARM7 FIFO Empty Interrupt Request Flag
- A7[d16]: ARM7 Interrupt Request Flag
- I/D[d13]: DS Accessory IREQ/DREQ Interrupt Request Flag
- K[d12]: Key Interrupt Request Flag  
For more information, see ["12.2 Interrupt Handling for Key Input"](#) on page 298.
- D3[d11]: DMA3 Interrupt Request Flag
- D2[d10]: DMA2 Interrupt Request Flag
- D1[d09]: DMA1 Interrupt Request Flag
- D0[d08]: DMA0 Interrupt Request Flag  
For more information, see [Chapter 7](#).
- T3[d06]: Timer 3 Interrupt Request Flag
- T2[d05]: Timer 2 Interrupt Request Flag
- T1[d04]: Timer 1 Interrupt Request Flag
- T0[d03]: Timer 0 Interrupt Request Flag  
For more information, see [Chapter 8](#).
- VE[d02]: V-Counter Match Interrupt Request Flag
- HB[d01]: H-Blank Interrupt Request Flag
- VB[d00]: V-Blank Interrupt Request Flag  
For more information, see ["4.3 Display Status"](#) on page 51.

## **9.4 Interrupt Cautions**

### **9.4.1 Clearing IME and IE**

Even while the command to clear all flags in the IME and IE registers is executing, relevant interrupts are generated.

When clearing the IE flags, be sure to clear IME first to avoid inconsistencies in interrupt checks.

### **9.4.2 Multiple Interrupts**

If clearing the IME and an interrupt occur at the same time, multiple interrupts are not accepted during that interrupt. Therefore, you must set the IME after clearing the IME during the interrupt routine.

### **9.4.3 Interrupt Delays During DMA Operation**

The CPU cannot access RAM other than the TCM or cache RAM during DMA operations.

Therefore, during the interval until the DMA stops, the interrupt is delayed when performing interrupt handling on anything other than TCM.

### **9.4.4 Interrupts from ARM7**

The A7, IFE, and IFN interrupts are for use by the subprocessor and the subprocessor API for communications.

The subprocessor API does not operate properly if these interrupts are disabled or if the interrupt request is reset.

## 10 Power Management

The Power Management API can be used by the application to put the NITRO into Sleep mode, to control power to various circuits, to check for the low-battery state, and to check whether the DS is open or closed.

### 10.1 Sleep Mode

The application can use the Power Management API to put the NITRO into Sleep mode. In Sleep mode, all circuits in the NITRO Processor stop. Power to the LCD and the sound is turned off, so there is nothing displayed and no sounds are played. However, the data in the NITRO Processor internal memory and in main memory are retained.

The Power LED blinks slowly in Sleep mode. In NITRO mode, a fully charged battery lasts about two weeks in Sleep mode.

Table 10-1 shows the factors that cause the NITRO to waken from Sleep mode and the timing with which this happens.

**Table 10-1 : Conditions for Waking from Sleep Mode**

Conditions for Waking	Timing
NITRO Is Opened	When NITRO is opened
RTC Alarm Feature	When the alarm reaches the set time
DS Game Card DS Option Pak	When a Game Card or DS Option Pak is accidentally removed or when an interrupt is generated from a pak that causes a cartridge interrupt (such as a pak that uses an Advance Movie or RTC).
Key Entry	When a previously specified key (with the exception of X or Y) is pressed

**Note:** When waking from Sleep mode, do not play sounds for the first 15 ms, which is the time for the sound to recover from the power-off state.

## 10.2 Controlling Various Power Supplies

The Power Management API can be used to control the power supply to the sound circuitry, LCD backlight, LCD, microphone, system, and graphics.

### 10.2.1 Sound

The power supply to the sound circuitry can be controlled using the Power Management API. When the application is not playing sounds, the Power Management API can be used to turn power off to the sound circuitry to reduce battery consumption.

**Note:** Do not play sounds for the first 15 ms, which is the time it takes for the sound circuitry to recover from the power-off state.

### 10.2.2 LCD Backlight

The Power Management API can be used to control the power supply to the backlight of the upper screen and the backlight of the lower screen separately. When the application is using only one LCD screen, the power can be turned off to the screen that is not being used to reduce battery consumption.

When no game is being played and the system is not standing by for wireless, if you choose not to display on either LCD and to turn off power to the LCD backlight in consideration of battery life, we suggest that you move to Sleep mode, which more effectively reduces power consumption.

### 10.2.3 LCD

The Power Management API can be used to control power supply to both the upper and lower screen LCDs. Furthermore, the LCD backlight can be turned off regardless of the LCD backlight settings. However, the LCD backlight settings are preserved.

When there is no display on the LCD screen, such as when the application is waiting for wireless communication, battery consumption can be controlled by turning off the power supply to the LCD.

When the system is not standing by for wireless, if you choose to turn off power to the LCD in consideration of battery life, we suggest that you move to Sleep mode, which more effectively reduces power consumption.

**Note:** Although the power supply to the LCD can be directly controlled with the graphics power control register mentioned below, the LCD circuitry may be damaged according to when the LCD is turned on or off. Therefore, directly changing the register value is prohibited. When manipulating the power supply to the LCD, always use the API.

When the LCD is turned off, the power supply for the sound amp is also turned off, and the speaker will not function. However, if you connect the headphones when the LCD is on and then turn off the LCD, the headphones will still generate sound. Additionally, the headphones may not generate sound if they are connected while the LCD is off. Any software designed to generate sound while the LCD is off should clearly describe in its Instruction Booklet a process for reliably generating sound through the headphones.

### 10.2.4 Microphone

When using the microphone, power must be turned on to the PMIC's programmable gain amplifier (PGA). The Power Management API can be used to control power to the PGA.

**Note:** Do not use the microphone for 3 seconds after it is turned on.

### 10.2.5 System

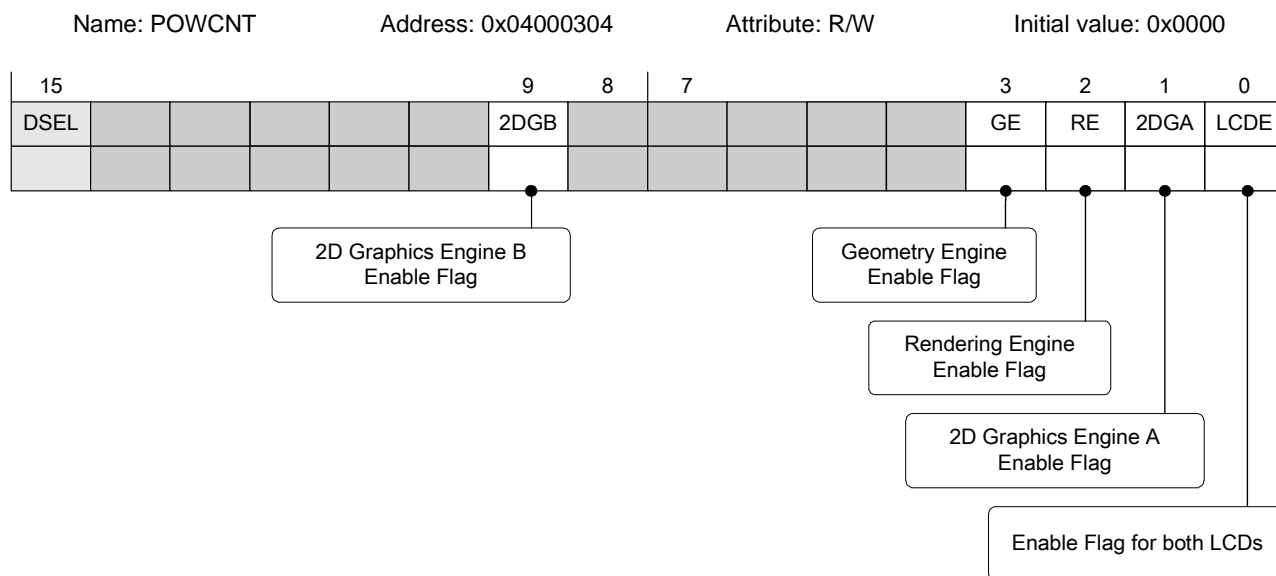
The Power Management API can be used to turn off the NITRO's system power (shut-down).

**Note:** Depending on the DS, there is no guarantee that the system will be reliably shut down if a shut-down is executed when the LCD is off (in rare instances the system may restart). Therefore, be sure to perform a shut-down only when the LCD is on.

## 10.2.6 Graphics

Power consumption can be reduced by: controlling the clock supply to the circuits of the geometry engine, the rendering engine, and the 2D graphics engine; and disabling circuits that are not being used.

**Figure 10-1 : POWCNT: Graphics Power Control Register**



- 2DGB[d09] : 2D Graphics Engine B Enable Flag

Used to reduce power consumption when the 2D Graphics Engine B is not being used.

0	Disable
1	Enable

- GE[d03] : Geometry Engine Enable Flag

When the geometry engine is enabled, issue the SwapBuffers command once.

0	Disable
1	Enable

- RE[d02] : Rendering Engine Enable Flag

If the rendering engine is enabled, issue the SwapBuffers command once.

0	Disable
1	Enable

- 2DGA[d01] : 2D Graphics Engine A Enable Flag

Used to reduce power consumption when only 3D graphics are being used.

0	Disable
1	Enable

- LCDE[d00] : Enable Flag for Both LCDs (Use Prohibited)

When disabled, both the clock supply to the LCD main and sub-controllers and the power supply to the main and sub-LCDs are stopped.

0	Disable
1	Enable

**Note:** Use the API to enable/disable the LCDs. Be careful not to change this bit when writing data to other bits.

**The memory addresses and registers to which the clock signal is stopped when each flag is disabled:**

- When the 2D Graphics Engine A is disabled:  
0x04000008 - 0x0400004D  
0x04000050 - 0x04000055  
2D Graphics Engine A's OAM and palette RAM
- When the 2D Graphics Engine B is disabled:  
0x04001008 - 0x0400104d  
0x04001050 - 0x04001055  
2D Graphics Engine B's OAM and palette RAM
- When the Geometry Engine is disabled:  
0x04000400 - 0x04000473  
0x04000480 - 0x040004AF  
0x040004C0 - 0x040004D3  
0x04000500 - 0x04000507  
0x04000540 - 0x04000543  
0x04000580 - 0x04000583  
0x040005C0 - 0x040005CB  
0x04000600 - 0x04000607  
0x04000610 - 0x04000611  
0x04000620 - 0x04000635  
0x04000640 - 0x040006A3
- When the Rendering Engine is disabled:  
0x04000320 - 0x04000321  
0x04000330 - 0x04000341  
0x04000350 - 0x0400035D

0x04000360 - 0x040003BF

Table 10-2 shows the behavior that occurs when there is access to memory and registers to which the clock signal has been stopped.

**Table 10-2 : Access to Memory and Registers when Clock Signal is Stopped**

	Write	Read
Memory	Invalid	ALL zero
Registers	Invalid	Read-enabled

## 10.3 Power Status

### 10.3.1 Low Battery State

When the remaining charge in the battery drops below 10-20%, the low battery state is entered, and the Power LED turns red. The Power Management API can be used to read the battery state data and check for the low-battery state (see Table 10-3). The amount of charge left in the battery when the power LED turns red is only a rough indication due to individual differences in the NITRO system, batteries, application, and environmental temperature.

**Table 10-3 : Battery State Data**

Data Type	Data Content
Battery State	Low battery state flag (0 – 1)

**Details about PMIC status data:**

- Low Battery State Flag
  - 0 : Battery still has a charge.
  - 1 : Battery is low.

### 10.3.2 DS Open/Closed State

The Power Management API can be used to read the status data shown in Table 10-4 to check whether the DS is open or closed.

**Table 10-4 : DS Opened/Closed State Data**

Data Type	Data Content
Device Opened/Closed State	DS Opened/Closed State Flag (0 – 1)

**Details about DS open/closed status data:**

- DS Opened/Closed State Flag
  - 0 : DS is open.
  - 1 : DS is closed.



## 11.1 Divider

## Divider Data (Numerator, Denominator, Quotient, Remainder) Registers

31 24 23 16 15 8 7 0

Divider Data Lower Word

Diagram of the 32-bit Divider Data Upper Word. The word is divided into four 8-bit sections. The first section (bits 31-24) is labeled '31' and '24'. The second section (bits 23-16) is labeled '23' and '16'. The third section (bits 15-8) is labeled '15' and '8'. The fourth section (bits 7-0) is labeled '7' and '0'. The entire word is labeled 'Divider Data Upper Word'.

Signed integer (sign + 63-bit integer part)

## DIVCNT: Divider Control Register

Initial value: 0x0000

[illegible]

- BUSY[d15]: Busy Flag

0	Divider is ready
1	Divider is busy

- DIV0[d14]: Divide-by-zero error flag

0	There is no divide-by-zero error
1	There is a divide-by-zero error

- MODE[d01–d00]: Divider Mode

00	32-bit (DIV_NUMER)/32-bit (DIV_DENOM) quotient 32-bit (DIV_RESULT), remainder 32-bit (DIVREM_RESULT)
01	64-bit (DIV_NUMER)/32-bit (DIV_DENOM) quotient 64-bit (DIV_RESULT), remainder 32-bit (DIVREM_RESULT)
10	64-bit (DIV_NUMER)/64-bit (DIV_DENOM) quotient 64-bit (DIV_RESULT), remainder 64-bit (DIVREM_RESULT)
11	Setting prohibited

**Note:** Regardless of the Divider mode, the Division by Zero Error flag is enabled only when all 64 bits of the denominator (DIV\_DENOM) are zero.

For this reason, set all of the upper 32 bits of the denominator (DIV\_DENOM) to 0 even when the Divider mode is 32-bit/32-bit or 64-bit/32-bit.

If the upper 32 bits of the denominator (DIV\_DENOM) are not set to 0, the Division by Zero Error flag will not function properly.

### 11.1.1 Number of Calculation Cycles

After writing to the Divider Data Registers, the DIVCNT register's busy flag is set during the cycles shown in Table 11-1, according to the Divider Mode. When the busy flag has been cleared, you can find the calculation result by reading the register that stores the result.

**Table 11-1 : Calculation Bit Count and Calculation Cycle Count by Divider Mode**

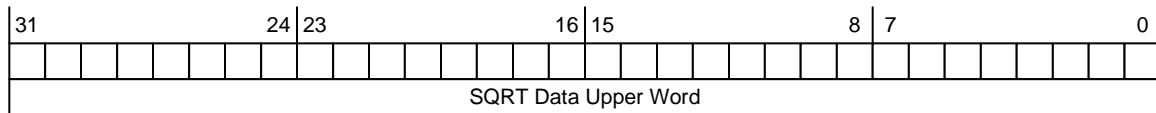
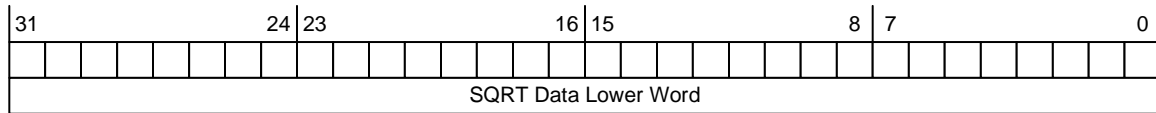
Divider Mode	Calculation Bit Count	Calculation Cycle Count
00	32 bits (DIV_NUMER)/32 bits(DIV_DENOM) Quotient 32 bits (DIV_RESULT), Remainder 32 bits (DIVREM_RESULT)	18 cycles
01	64 bits (DIV_NUMER)/32 bits (DIV_DENOM) Quotient 64 bits (DIV_RESULT), Remainder 32 bits (DIVREM_RESULT)	34 cycles
10	64 bits (DIV_NUMER)/64 bits (DIV_DENOM) Quotient 64 bits (DIV_RESULT), Remainder 64 bits (DIVREM_RESULT)	34 cycles

## 11.2 Square-Root Unit

The control registers that indicate the states of the square root calculator data register, calculation result register, calculation mode, and the square root calculator are shown below.

### SQRT\_PARAM: SQRT Data Register

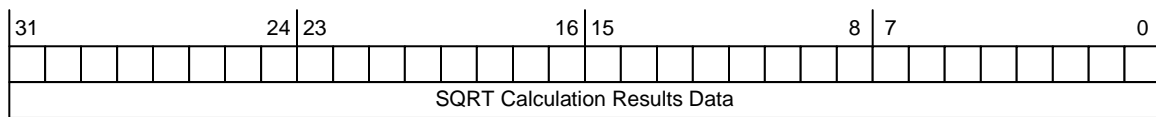
Name: SQRT\_PARAM      Address: 0x040002B8      Attribute: R/W      Initial value: 0x00000000\_00000000



Unsigned integer (64-bit integer part)

### SQRT\_RESULT: SQRT Calculation Result Register

Name: SQRT\_RESULT      Address: 0x040002B4      Attribute: R/W      Initial value: 0x00000000



### SQRTCNT: SQRT Control Register

Name: SQRTCNT	Address: 0x040002B0	Attribute: R/W	Initial value: 0x0000
---------------	---------------------	----------------	-----------------------



- BUSY[d15]: Busy Flag

0	Square root calculator ready
1	Square root calculator busy

- MODE[d00]: SQRT Computation Mode

0	32-bit input
1	64-bit input

### 11.2.1 Number of Calculation Cycles

The SQRTCNT register's busy flag is set during the cycles shown in Table 11-2, according to the Computation Mode after writing to the Data Registers. When the busy flag has been cleared, you can find the calculation result by reading the register that stores the result.

**Table 11-2 : Input Bit and Calculation Cycle Count by Computation Mode**

SQRT Computation Mode	Input Bit Count	Calculation Cycle Count
0	32-bit input	13 cycles
1	64-bit input	13 cycles



## 12 Keys

NITRO has A, B, L, R, +Control Pad, START, SELECT, X, and Y digital keys.

### 12.1 Input Keys

The status of the A, B, L, R, +Control Pad, START, and SELECT keys can be verified by reading the key input register (KEYINPUT) and checking the status of each bit. Because the X and Y keys are connected to the subprocessor, an API must be used to check the input status of these keys. The application can read all key data without regard for subprocessor operations when this API is used.

#### KEYINPUT: Key Input Register

Name: KEYINPUT

Address: 0x04000130

Attribute: R

Default Value: 0x0000

15							9	8	7						0
							L	R	DOWN	UP	LEFT	RIGHT	START	SEL	A
							Key Input								

- [d09–d00]: Key Input

0	Key is being pressed.
1	Key is not being pressed.

**Note:** ON-OFF may be repeated multiple times in a short time even if the user presses a key only once. To prevent a button from being pressed twice (chattering), allow an interval between readings of the key data (around 1 frame each). The input status of the X and Y keys cannot be read directly from the register.

## 12.2 Interrupt Handling for Key Input

Key input from the A, B, L, R, +Control Pad, START, and SELECT keys can generate interrupts. The key control register (KEYCNT) can be used to specify the key combinations and conditions for which interrupts can be generated.

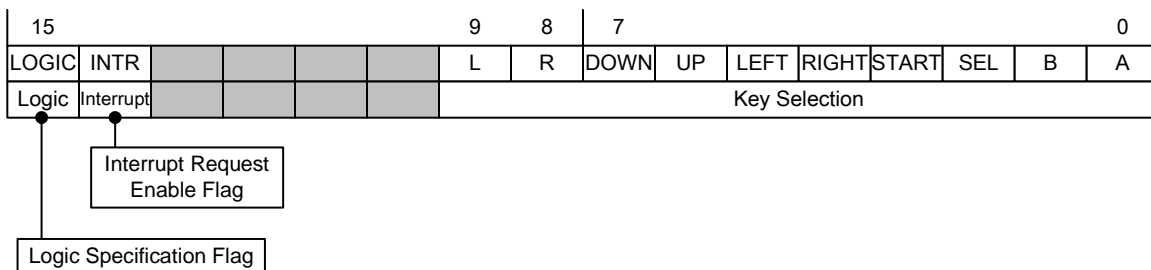
### KEYCNT: Key Control Register

Name: KEYCNT

Address: 0x04000132

Attribute: R/W

Default Value: 0x0000



- LOGIC[d15] : Logic Specification Flag

0	Detects if any of the specified keys was pressed
1	Detects if all of the specified keys were pressed

- INTR[d14] : Interrupt Request Enable Flag

0	Disable
1	Enable

- [d09–d00] : Key Selection

0	Key is not specified.
1	Key is specified.

**Note:** Interrupt handling cannot be specified for X or Y key input.



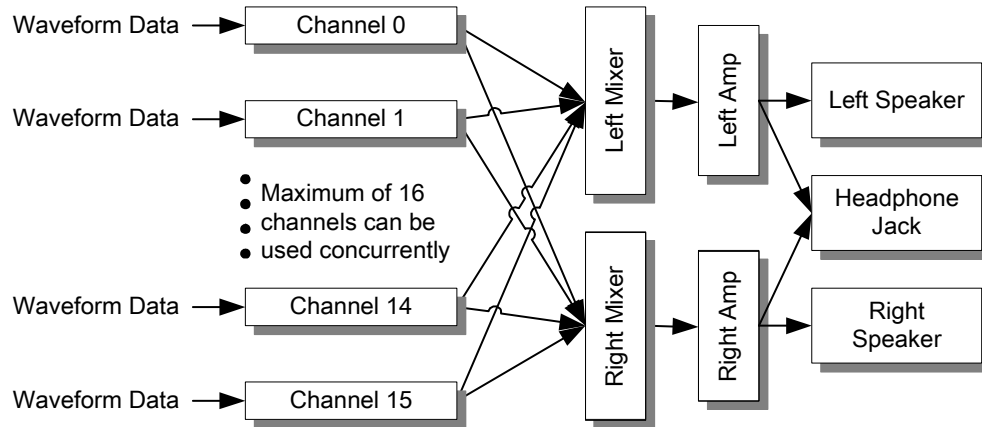
## 13 Sound

NITRO contains a sound circuit that enables the processing of 16-channels of simultaneous sound generation, two sound capture devices that can write the output from a specific channel or a mixer to memory, left and right speakers that can output sound, and a headphone output jack.

Since the subprocessor carries out the sequence processing and the sound generation processing, there will be no heavy burden on the main processor, even when sound processing is performed.

The sound circuit is illustrated in Figure 13-1.

**Figure 13-1 : Sound Circuit Outline Diagram**



13.1      **Hardware Specifications**

The specifications for the included sound circuit hardware are as follows.

13.1.1    **Data Format**

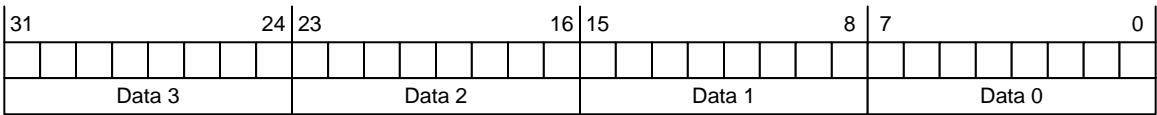
8bitPCM, 16bitPCM, IMA-ADPCM, PSG rectangular waveforms, and noise can be used as formats for waveform data.

The 8bitPCM, 16bitPCM, and IMA-ADPCM data formats, as well as descriptions of PSG rectangular waveforms and noise are shown below.

13.1.1.1   **8bitPCM Data Format**

The 8bitPCM data format is shown below.

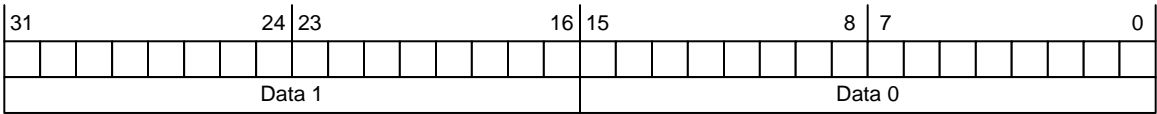
**8bitPCM Data Format**



13.1.1.2   **16bitPCM Data Format**

The 16bitPCM data format is shown below.

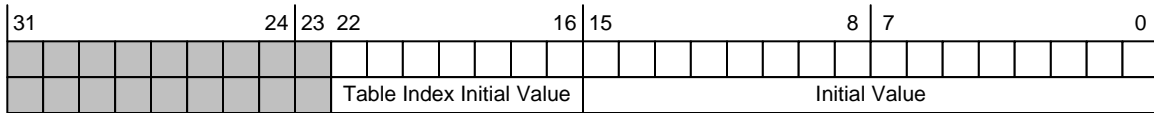
**16bitPCM Data Format**



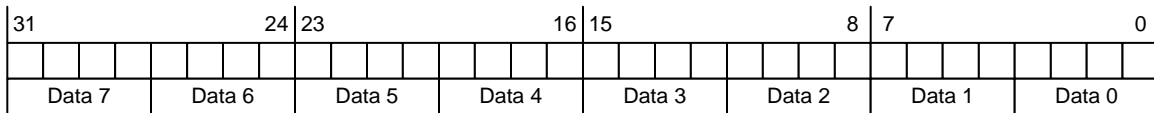
### 13.1.1.3 IMA-ADPCM Data Format

The header and data parts of the IMA-ADPCM data format are shown below.

### IMA-ADPCM Header Format (First 32 bits)



### IMA-ADPCM Data Format (From the 33<sup>rd</sup> bit)



**Note:** When repeatedly playing the ADPCM with the repeat feature, set the repeat pointer to the address of the data section, rather than that of the header section.

Also, if the repeat pointer is altered after starting playback on ADPCM, normal repeat play is not possible.

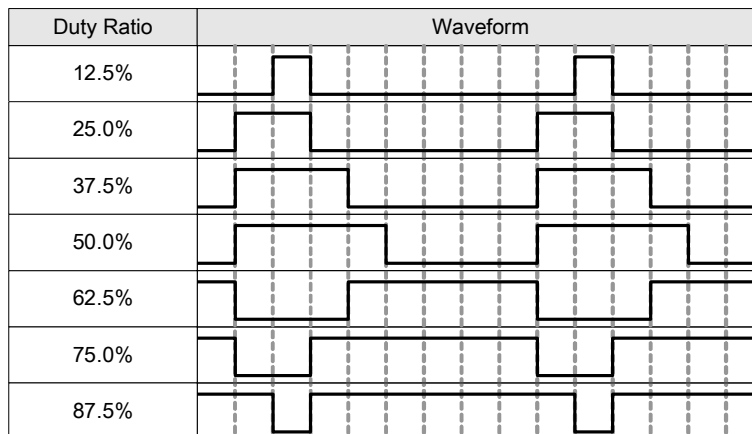
Be sure to stop playback before making changes to the repeat pointer.

#### 13.1.1.4 PSG Rectangular Waveforms

The *Programmable Sound Generator* (PSG) creates tones by altering the frequency of the output rectangular waveform (square waves) and the duty ratio.

The duty ratios of the PSG rectangular waveforms used on NITRO can be altered as shown in Table 13-1.

### Table 13-1 : Duty Ratio and PSG Rectangular Wave Waveforms



### 13.1.1.5 Noise

Noise has no configurable items.

Noise can be used to generate white noise on a channel designated for noise.

### 13.1.2 Channels

Waveform data can be played simultaneously from 16 channels. However, PSG rectangular waveforms can be played on only 6 specific channels, and noise can be played on only 2 specific channels of the 16 channels.

The volume, pitch, and pan (orientation) can be configured for each channel.

The playable channels for each format are shown in Table 13-2.

**Table 13-2 : Overview of Data Formats and Playable Channels**

Data Format	Playable Channels
8bitPCM	Can be played on all channels from 0 to 15.
16bitPCM	
ADPCM	
PSG Rectangular Waveform	Can be played on Channels 8 to 13. 8bitPCM, 16bitPCM, and ADPCM cannot be played simultaneously on a channel that is playing a PSG rectangular waveform.
Noise	Can be played on Channels 14 to 15. 8bitPCM, 16bitPCM, and ADPCM cannot be played simultaneously on a channel that is playing noise.

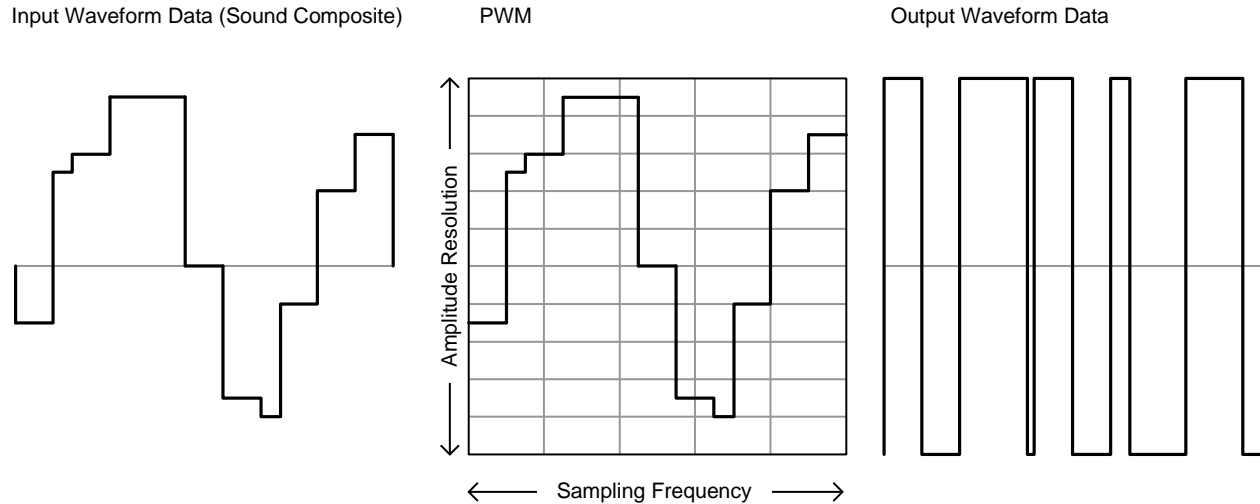
### 13.1.3 Mixer

NITRO comes equipped with both left and right mixers.

The sampling frequency of the mixer is 1.04876 MHz with an amplitude resolution of 24 bits, but the sampling frequency after mixing with PWM modulation is 32.768 kHz with an amplitude resolution of 10 bits.

PWM is an abbreviation of *Pulse Width Modulation*. As "[Figure 13-2 : Pulse Width Modulation \(PWM\)](#)" on page 303 shows, PWM converts the amplitude of a fixed-interval pulse to a duty ratio, and then outputs it.

The stronger the amplitude, the higher the duty ratio it is converted to.

**Figure 13-2 : Pulse Width Modulation (PWM)**

### 13.1.4 Master Volume

The speaker output can be adjusted in 128 steps (from 0 to 127) via the master volume.

### 13.1.5 Sound Capture

There are two built-in sound capture devices on NITRO that allow output waveform data to be written to memory.

Output from the left mixer or output from channel 0 can be written to memory with sound capture 0.

Output from the right mixer or output from channel 2 can be written to memory with sound capture 1.

The sampling frequency can be set up to 1.04876 MHz. The amplitude resolution can also be set from 8 bits to 16 bits.

### 13.1.6 Power Control

Turning off power to the sound circuit can reduce battery consumption when not using sound.

When restoring the power supply to the sound circuit (coming out of sleep mode, etc.), do not output any sound during the 15 milliseconds it takes the sound circuit to recover.

For more details about controlling power to the sound circuit, see [Chapter 10](#).

### 13.1.7 Cautions

Playing waveform data with a high sampling rate, or playing sound whose pitch is higher than the original data because of fast-forwarding, leads to more frequent DMA transfers.

If DMA transfers occur frequently, they affect the main processor processes as well as subprocessor processes, such as wireless communications and the microphone.

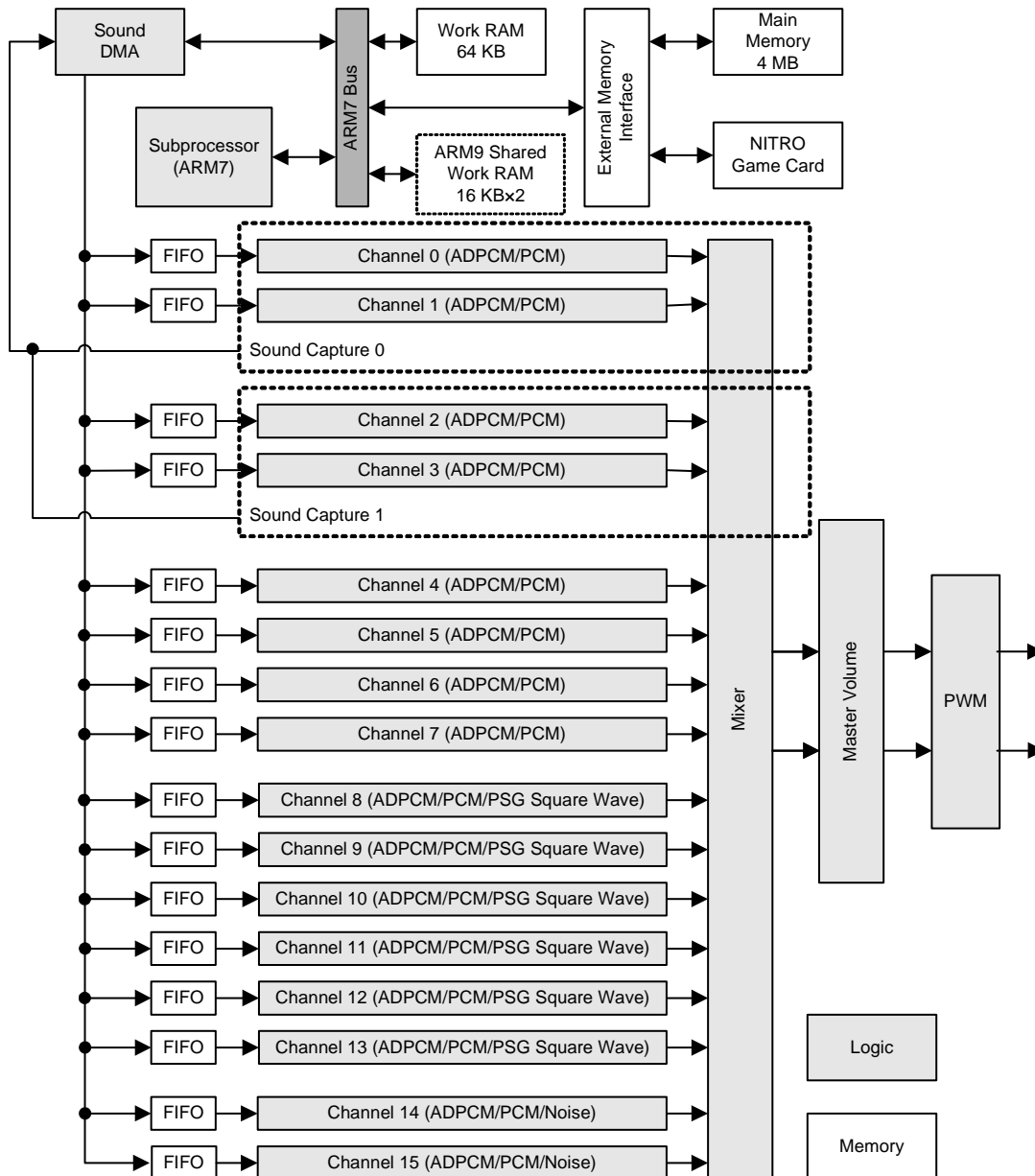
## 13.2 Sound Block Diagrams

This section shows the block diagrams for the NITRO sound circuits.

### 13.2.1 Overall Sound

The block diagram for the overall sound circuit is shown in Figure 13-3.

**Figure 13-3 : Overall Sound Block Diagram**





**Caution 1:** The switches before and after the pan blocks of channels 1 and 3 are always circuits connected to selectors A and B of the final-step output selection. Therefore, when channels 1 or 3 are selected as final-step outputs, sound will be output. Be aware of this when you do not want to output any sound.

**Caution 2:** The input signal to the mixer from the pan blocks of channels 1 and 3 is determined by the priority shown on Table 13-3. When channels 1 and 3 are set to bypass to the final-step output, the added channels are input to the mixer. Even if the output of that mixer is configured to be captured, channels 1 and 3 will not be input to the mixer (the switch immediately after the pan block will connect to GND, and the mixer input will always be 0 in this case only). As a result, there will be no reverb. Be aware that reverb using both the mixer and adder is not possible when channels 1 and 3 are set to bypass to the final-step output. If you are considering Caution 3 below, it is recommended that you use only the mixer.

**Table 13-3 : Switch Input Priority from Channels 1 and 3 to the Mixer**

Priority	Mixer Input	Switch State
High	0 Input (Connected to GND)	Sound Final-Step Output Bypass Configuration
	Pan 0 along with Pan 2	Channel Addition and Capture Configuration
Low	Pan 1 along with Pan 3	Normal Configuration

**Caution 3:** There is a fault in the logic (preliminary to selector 0 and 1) for capturing the channel adder output. The following situations can occur:

- When adding channels 0 and 1 along with channels 2 and 3, if an overflow occurs in either of the addition results, sign-inverted data will be output.
- When adding channels 0 and 1 along with channels 2 and 3, if the signs of channels 0 and 1 and channels 2 and 3 are each negative, the capture data will be forcibly converted to the maximum negative value.

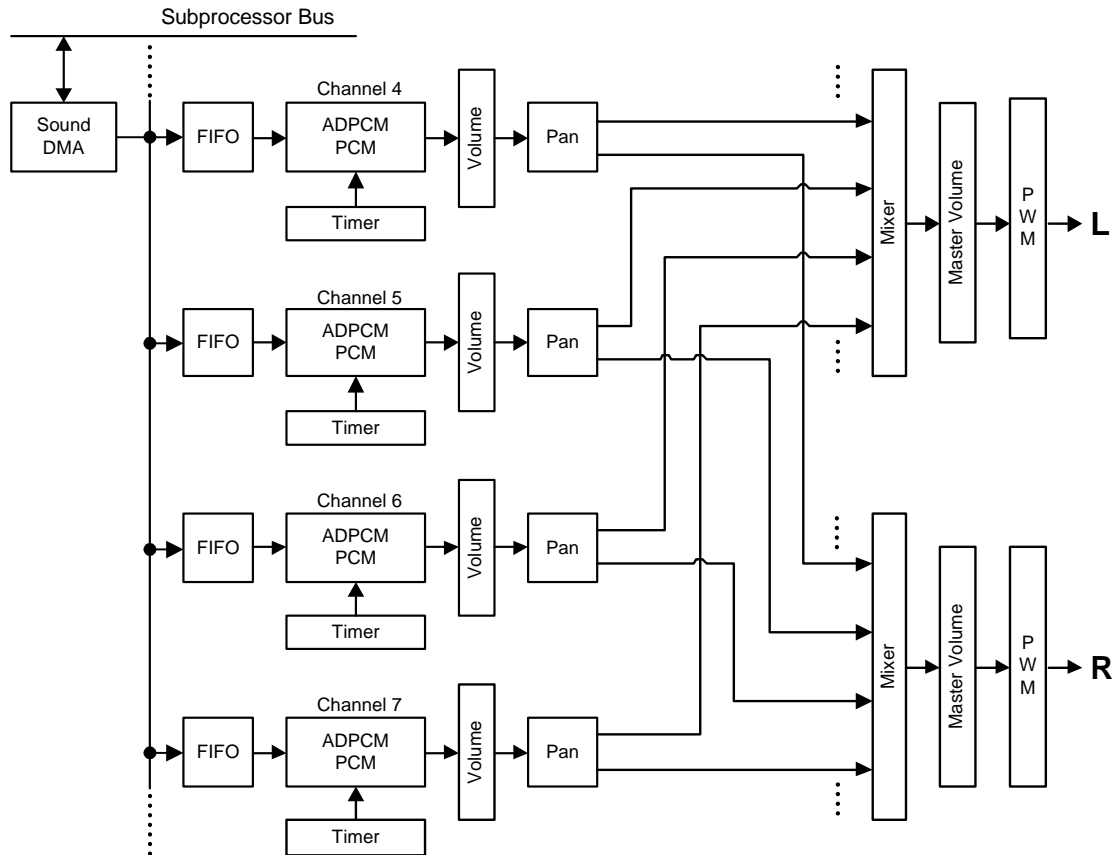
Noise will be output in the sounds from these results. To deal with these faults, make sure that the addition data does not become saturated when using the adder, and that the values for the channels 0-1 and channels 2-3 do not both become negative when not using the adder.



### 13.2.3 Channels 4 - 7

The block diagram for channels 4 to 7 is shown in Figure 13-5.

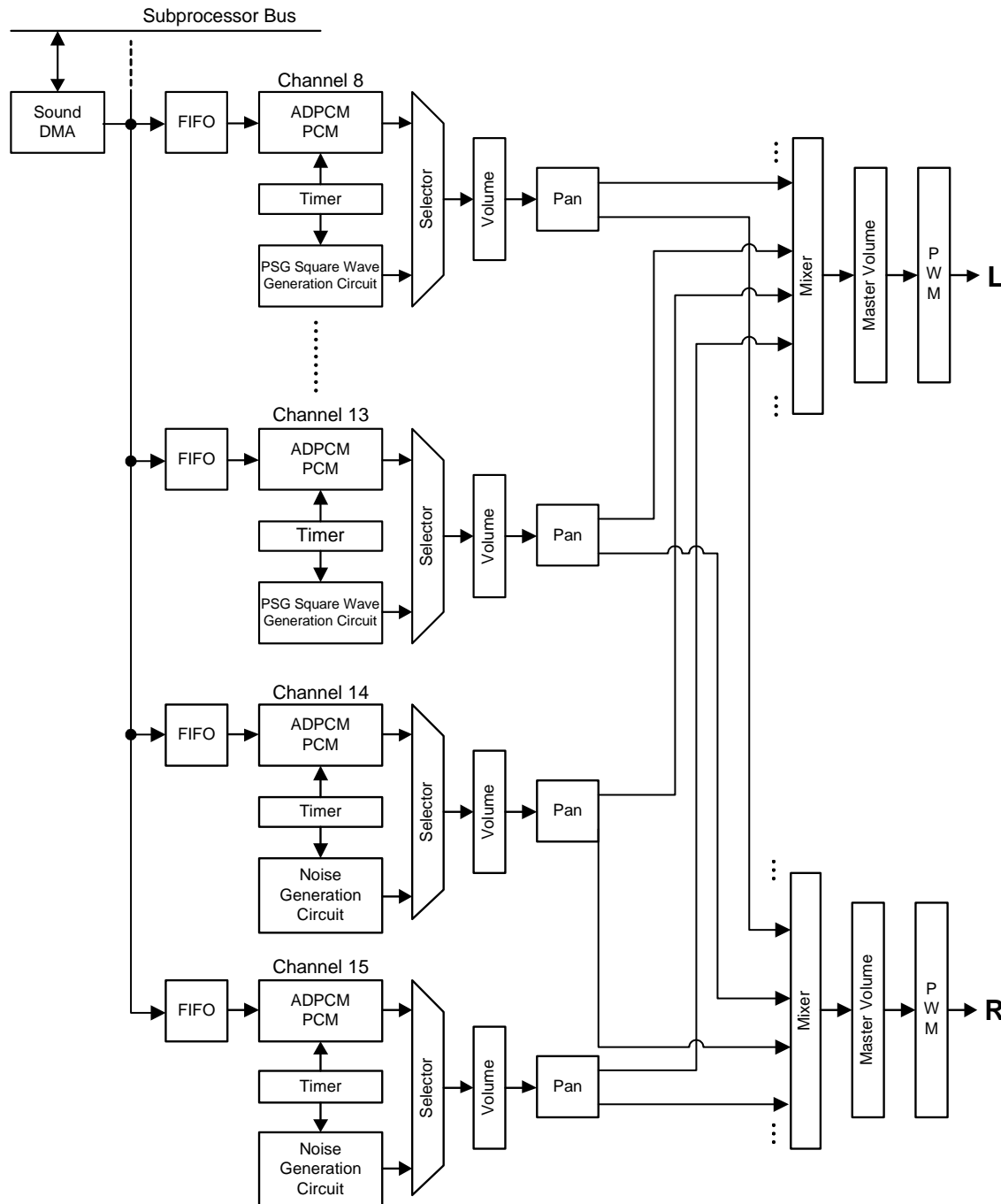
**Figure 13-5 : Channel 4-7 Block Diagram**



### 13.2.4 Channels 8 - 15

The block diagram for channels 8 to 15 is shown in Figure 13-6. PSG rectangular waveforms can be played on channels 8 to 13. Noise can be played on channels 14 and 15.

**Figure 13-6 : Channel 8-15 Block Diagram**



## 13.2.5 Examples of Using Sound

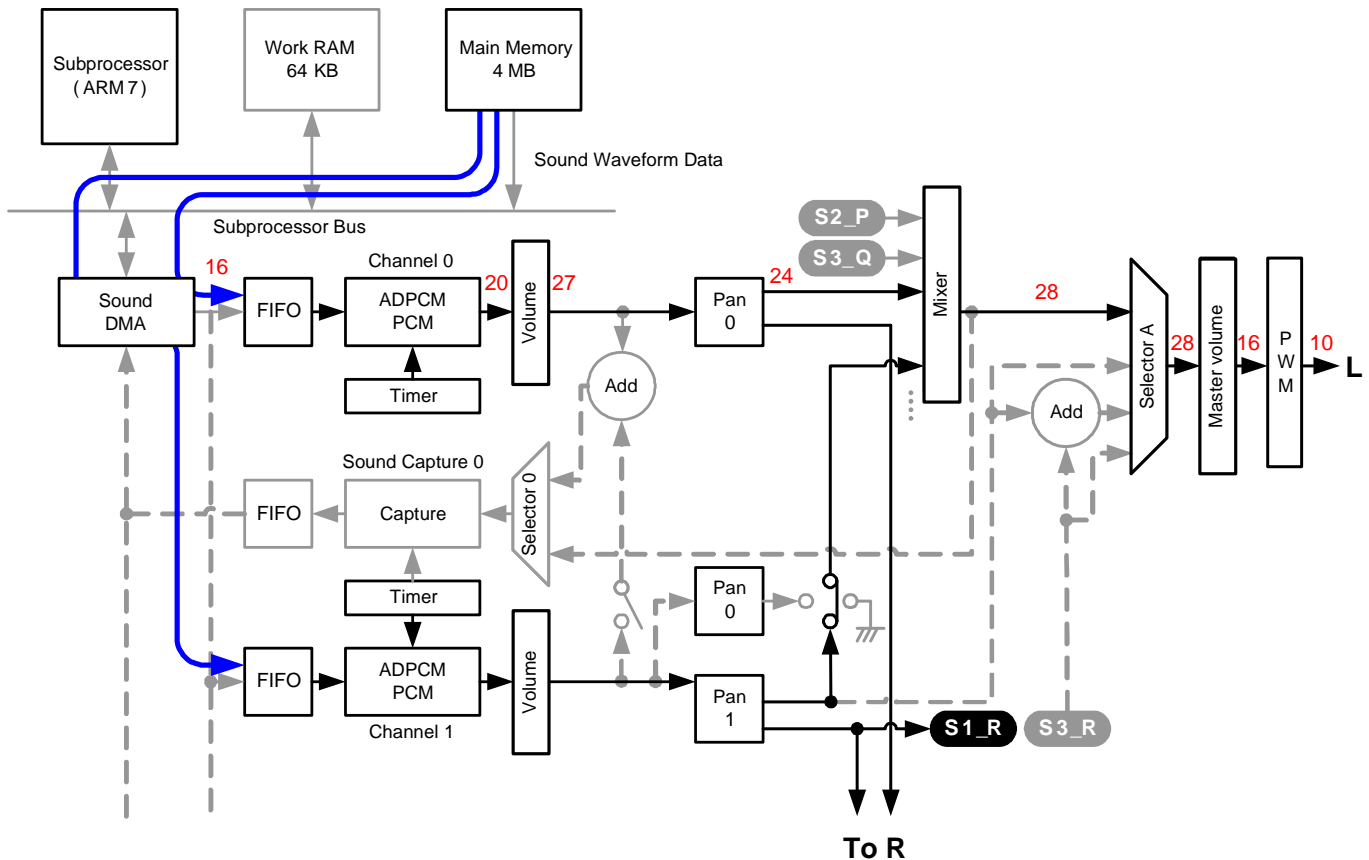
Examples of normal sound, a reverb effect using sound capture circuitry, and using the sound circuitry for sound effects are shown in the figures below.

### 13.2.5.1 Normal Use Example

Under normal use, the sound waveform data read from main memory is played on each channel, and then is output to the speaker via the mixer.

An example of a sound circuit under normal use is shown in Figure 13-7. Red numbers shown in Figure 13-7 indicate the bit count of data at the time of block input/output.

**Figure 13-7 : Example of Sound Usage (Normal)**



(Right Side Omitted)

### 13.2.5.2 Reverb Example

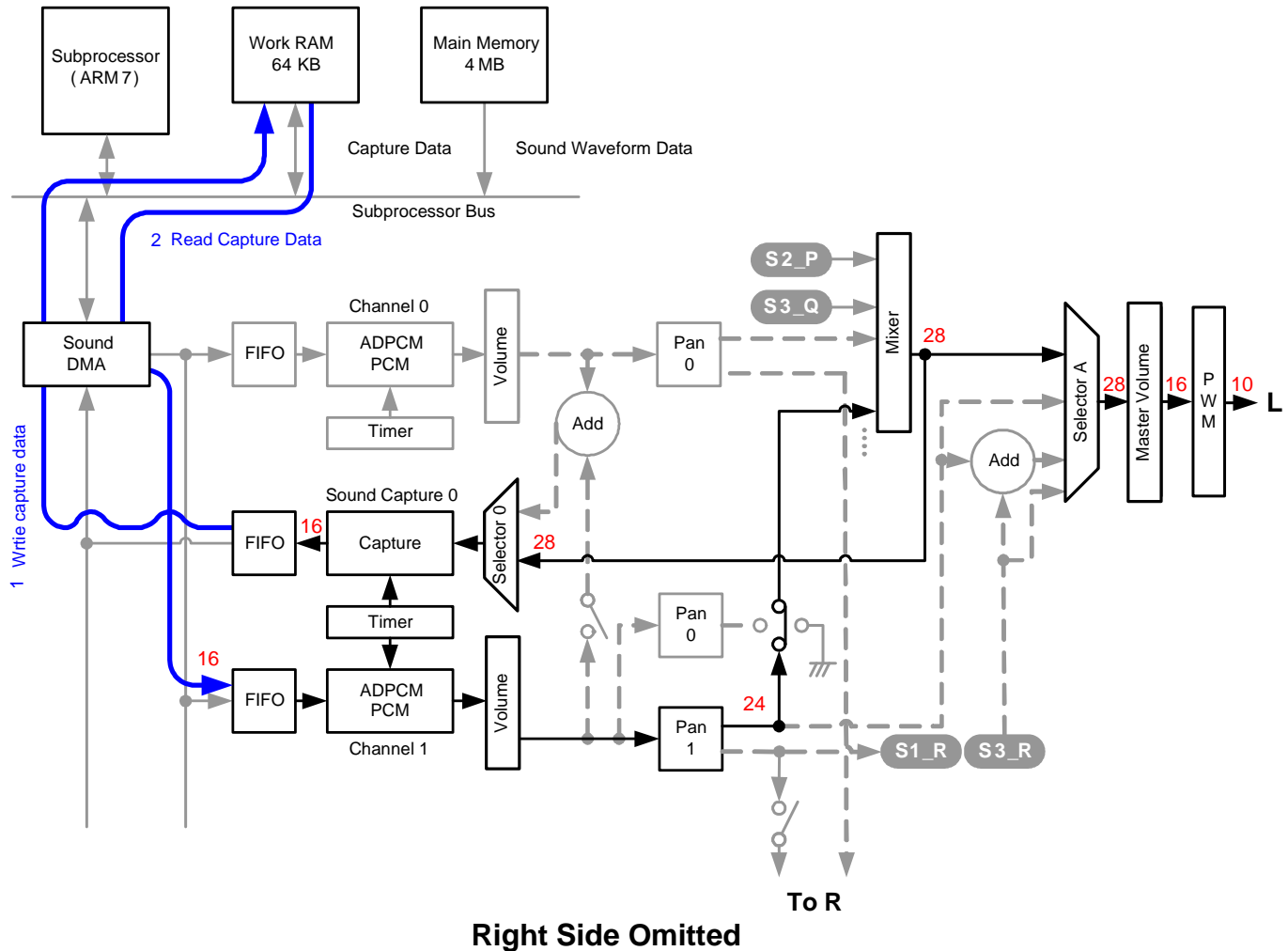
A reverb effect (echo) can be achieved using the sound capture feature.

Use sound capture to store the output from the mixer in Work RAM. A reverb effect results from playing the stored sound data on a channel and outputting it to the speakers via the mixer.

An example of sound circuit usage during reverb is shown in Figure 13-8. Red numbers shown in Figure 13-8 indicate the bit count of data at the time of block input/output.

In the example, Channel 1 is used for the reverb effect.

**Figure 13-8 : Example of Sound Usage (Reverb)**



### 13.2.5.3 Effect Example

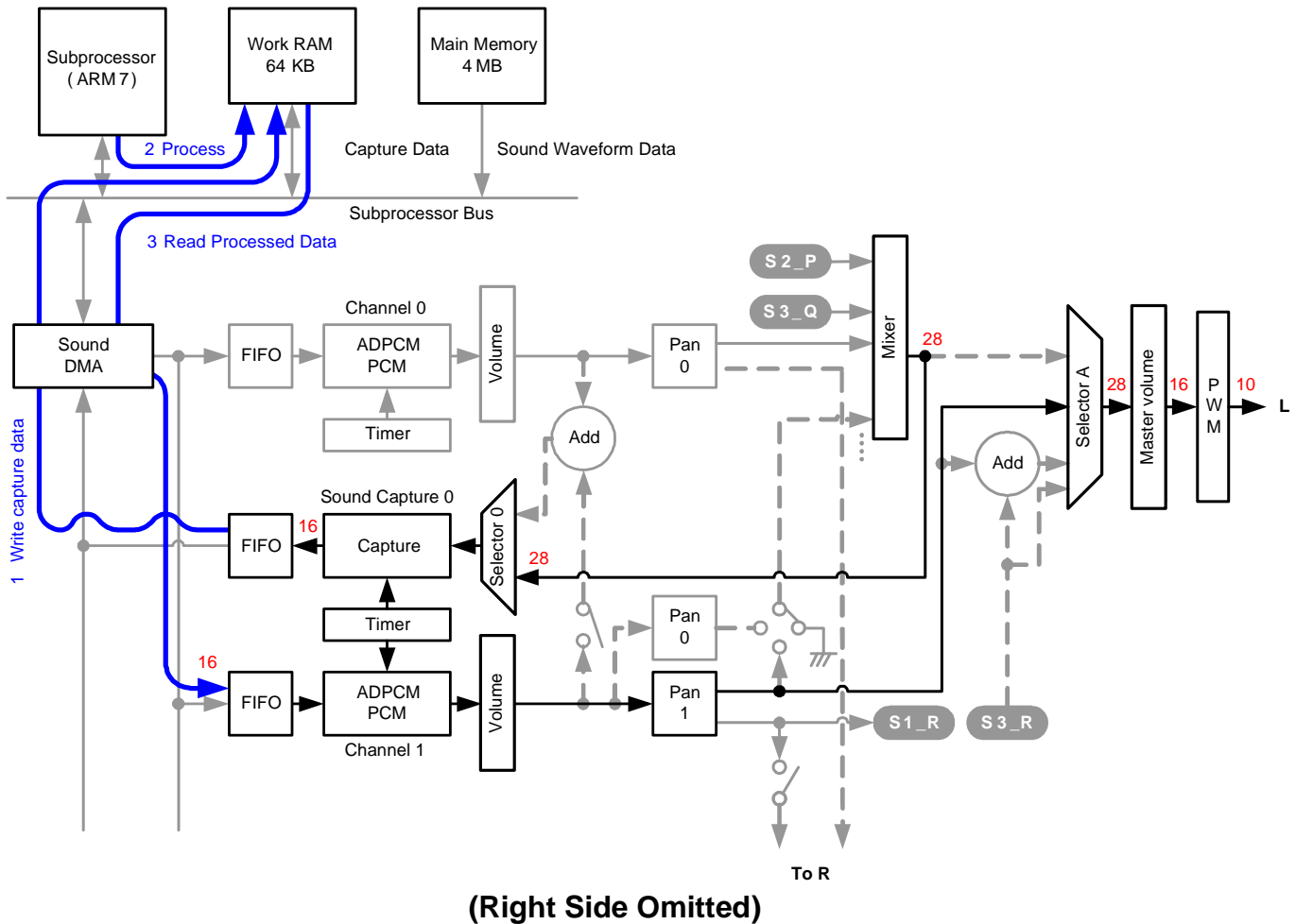
Data can be modified before outputting the sound by using the sound capture feature.

Store the captured data in Work RAM; after modifying the data using the subprocessor, output the data from a channel.

A sound effect example is shown in Figure 13-9. Red numbers shown in Figure 13-9 indicate the bit count of data at the time of block input/output.

In the example, Channel 1 is used for the effect.

**Figure 13-9 : Example of Sound Usage (Effect)**



### **13.3 NITRO-Composer**

There is no need to be concerned with the subprocessor operations when using the NITRO-Composer. The NITRO-Composer allows easy playback of even complicated sounds, such as BGM.

#### **13.3.1 NITRO-Composer Playback Method**

There are three playback methods: sequence playback, waveform playback, and stream playback.

##### **13.3.1.1 Sequence Playback**

Sequence playback plays a variety of combined sounds, such as BGM and sound effects.

A maximum of sixteen sequences can be played simultaneously.

For example, during playback of a BGM that has one sequence assigned to it, up to fifteen sound effects can be played simultaneously.

A variety of parameters for each sequence, such as tempo and volume, can be individually controlled on the application side.

##### **13.3.1.2 Waveform Playback**

Waveform playback is a method for direct playback of waveform data.

It can play back waveform data, etc., that has been sampled with a microphone.

##### **13.3.1.3 Stream Playback**

Stream playback plays back long sequences, such as movie soundtracks.

Waveform data can be played back while the data stored on the NITRO card is sequentially loaded.

## 14 Wireless Communications

NITRO contains hardware for wireless communications using the 2.4 GHz band.

### 14.1 Hardware Specifications

The hardware specifications for wireless communications are shown in Table 14-1.

**Table 14-1 : Wireless Communications Hardware Specifications**

Item	Description
Band Used	2.4 GHz band
Communications Protocol	IEEE802.11 (Internet Play) Nintendo's proprietary protocol (Multi-Card Play) Nintendo's proprietary protocol (DS Single-Card Play)
Security	WEP40 bit/128 bit compatible
Wireless Channels	13 channels
Communications Speed	1 Mbps or 2 Mbps
Communications Range	10-30 meters This can change dramatically, depending on the environment and orientation of the unit.
MAC Address	Unique for each DS and thus can be used for identification.
Interfering Devices	Devices that use the 2.4 GHz band (cordless phones, microwave ovens, the wireless adapter for Game Boy Advance, WaveBird, other WiFi devices, etc.)

**Note:** When communications is used, power consumption also increases, so the battery will be consumed more quickly. Accordingly, when not using communication, put the unit in the STOP state using the wireless communications API.

The wireless adapter for Game Boy Advance cannot be used to communicate with NITRO.

Other devices may cause interference, making communications difficult. To avoid this, make the communications packet size as small as possible.

### 14.2 Wireless Manager

By calling the wireless manager API, you can control the wireless system without worrying about the operation of the subprocessor.

The unit can use Internet Play, Multi-Card Play, and Single-Card Play.

#### 14.2.1 Internet Play

In this mode, the unit can connect to the Internet using a wireless LAN (IEEE802.11b/g) access point.

### **14.2.2 Multi-Card Play**

This mode allows wireless communication between a maximum of 16 DS devices.

Because it uses NITRO's proprietary communications method, data units can be exchanged in less than one frame.

In Multi-Card Play, the maximum communications data size is 512 bytes.

### **14.2.3 Single-Card Play**

This mode allows a child device without a NITRO card to download a game from a parent device with a NITRO card.

If the parent device sends data with a header that includes address information, the child device's system ROM stores the data in the region specified by the header.

This mode uses Nintendo's proprietary protocol.



## 15 Touch Panel

The lower screen includes a resistive-membrane touch panel that allows coordinates to be obtained in dot units. Although the touch panel can be operated with a finger, a touch pen with a 1.0 mm tip (see "[Figure 15-1 : Comparison of LCD Dot Size and Touch Pen Size](#)" on page 316) is included with the NITRO as standard equipment.

The application can use the touch panel without regard for subprocessor operations when using the touch panel API. To overcome the differences in the coordinate position data on the API side, the calibration data stored in internal flash memory must be read with the IPL correction program and set with the API before using the touch panel.

The touch panel input data shown in Table 15-1 can be read with this API. The API reads in two ways: auto-sampling that reads four times per frame and request sampling that reads in real time in response to a request. When a request for a read is generated with request sampling, an interval of at least 4.17 msec must be maintained between requests to ensure a correct reading.

**Table 15-1 : Touch Panel Input Data**

Data Type	Data Description
Touch Panel Input Data	x coordinate (8 bit), y coordinate (8 bit) Touch Determination Flag (1 bit) Data Validity Flag (2 bit)

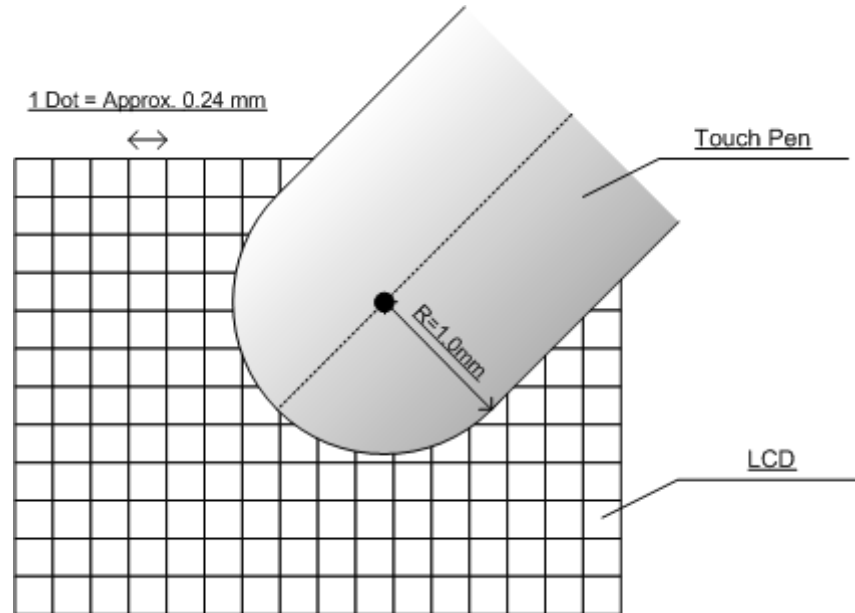
The details for input data for the touch panel are shown below.

- x-coordinate, y-coordinate
  - x-coordinate : 0 - 255 (dots)
  - y-coordinate : 0 - 191 (dots)
- Touch Determination Flag
  - 0 : The touch panel is not being touched
  - 1 : The touch panel is being touched
- Data Validity Flag
  - 00 : Both the x-coordinate and y-coordinate are valid
  - 01 : The x-coordinate is invalid
  - 10 : The y-coordinate is invalid
  - 11 : Both the x-coordinate and y-coordinate are invalid

**Note:** Structurally, the resistive-membrane touch panel can detect coordinates for only one location at a time. Therefore, if multiple locations are touched at the same time, the coordinates for each point cannot be detected. When the included stylus is used, the touch panel must be pressed down with a force of at least 80 g for the location to be detected. In some cases, the touch pen may not be able to depress areas within four dots of the screen edge as a result of built-in error between the touch panel and the DS and limitations due to the shape of the touch pen's tip.

On some occasions, irregular coordinate data are read immediately after the screen is touched or immediately after removing the touch pen from the screen. In these cases (in particular when operating a button displayed on screen), have the application use coordinates that are read in with the same value continually as valid data for processing. Note that the touch determination flag and the data validity flag are independent of each other, and that there may be a situation where, even though invalid data were stored, the touch panel is being touched. For example, cases may occur in which invalid data get stored while drawing a figure with a single stroke. In such cases, rather than determining that the user removed the touch pen from the touch panel, make the determination by reading the touch determination flag.

**Figure 15-1 : Comparison of LCD Dot Size and Touch Pen Size**

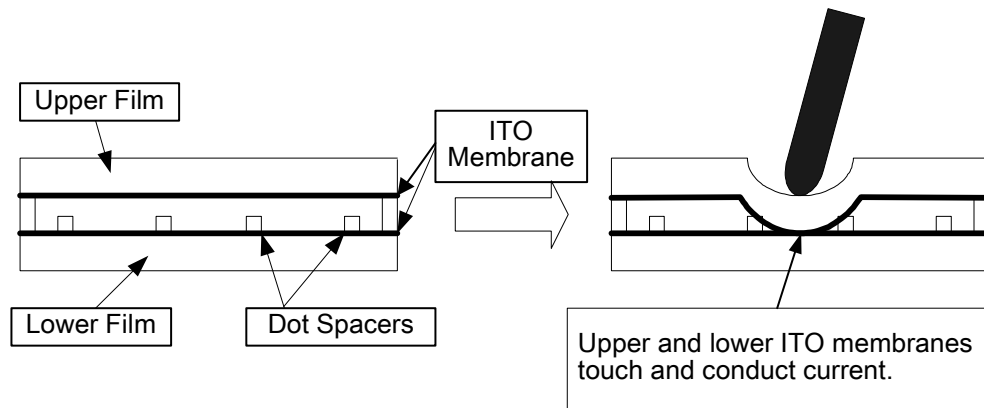


## 15.1 Touch Panel Structure

The construction of the resistive-membrane touch panel is illustrated in Figure 15-2.

Normally, the space formed between the upper and the lower films, both of which are coated with a transparent conducting membrane (ITO membrane: indium tin oxide), prevents current from being conducted. When a finger or stylus presses on the panel, the pressure causes the upper and lower films to touch and conduct current. The dot spacers prevent erroneous input, and the NITRO from being continuously on.

**Figure 15-2 : Touch Panel Structure**

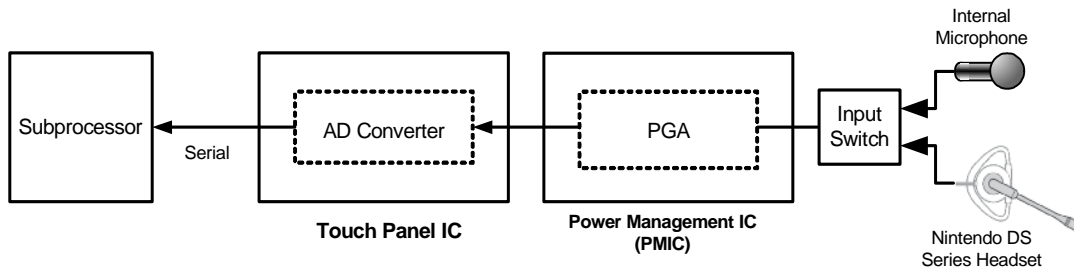




## 16 Microphone

Nintendo DS is equipped with an omnidirectional condenser microphone that can be used for audio sampling. In addition, the user can use a Nintendo DS Series Headset. Microphone sensitivity can vary by up to a factor of two, depending on the console. Figure 16-1 shows a schematic diagram of the microphone.

**Figure 16-1 : Microphone Schematic**



In preparation for audio sampling, the gain for audio picked up by the microphone can be adjusted to 4 levels using the programmable gain amplifier (PGA). If the audio input is low and the sampling resolution is low or if the audio input is loud, you can adjust the gain to balance the amplitude resolution. The amplitude resolution for audio sampling can be set to 8 or 12 bits.

The range of possible settings for gain and amplitude resolution are indicated in Table 16-1.

**Table 16-1 : Ranges of Possible Settings for Gain and Amplitude Resolution**

Data Type	Data Description
Gain	4 levels: 20x, 40x, 80x, and 160x Default is 40x (API specifications)
Amplitude Resolution	8- or 12-bit If 8-bit is used, one byte is used for one item of sampling data If 12-bit is used, two bytes are used for one item of sampling data

To use the microphone, power must be supplied to the PGA. For instructions on supplying power to the PGA, see [Chapter 10](#).

The microphone can be used without regard for subprocessor operations when the microphone API is used. The sampling rates that can be specified range from several kHz to 32 kHz. However, the sampling rates for normal operation depend on the status of subprocessor use. The recording time depends on the memory size and sampling rate provided by the application.

The 60-Hz noise that is synched to the V-Blank is superimposed on the microphone input. However, this frequency is very low and the noise level is sufficiently low compared to audio input. Therefore, this will not cause a problem as audible sound.

**Note:** Wireless communication and sound processing also use the subprocessor. Therefore, if the microphone is used at the same time as these features, specify a sampling rate that takes into account the load on the subprocessor. In addition, the same serial bus is used to read touch panel data, access the internal flash memory, and control the PMIC. If the microphone is used at the same time as these features, ensure that conflicts do not occur. Do not use the microphone for 3 seconds after the power is turned on.

Depending on the console, the microphone input value while there is no sound will have individual differences in the range as shown in Table 16-2. In order to prevent a false determination that there is microphone input even when there is none, avoid determining that there is microphone input within these ranges.

**Table 16-2 : Microphone Input Values when there is no Sound**

Amplitude Resolution		Amplitude Deviation	Noise Component	Total (Microphone input values when there is no sound)
8 bit	Signed	-13 to +13	-14 to +14	-27 to +27
	Unsigned	115 to 141	114 to 142	101 to 155
12 bit	Signed	-192 to +192	-224 to +224	-416 to +416
	Unsigned	1856 to 2240	1824 to 2272	1632 to 2464

Also, depending on the console, it will not be possible to pick up the full range of microphone input values listed under amplitude resolution. Differences are shown in Table 16-3. In cases such as determining the magnitude of microphone input with threshold values, avoid using values that include ranges outside of the guaranteed input in order to prevent false determinations.

**Table 16-3: Guaranteed Microphone Input Ranges**

Amplitude Resolution		Outside of Guaranteed Input Range LOWER LIMIT	Outside of Guaranteed Input Range UPPER LIMIT	Guaranteed Microphone Input Value Ranges
8 bit	Signed	-128 to -108	+107 to +127	-107 to +106
	Unsigned	0 to 20	235 to 255	21 to 234
12 bit	Signed	-2048 to -1728	+1727 to +2047	-1727 to +1726
	Unsigned	0 to 320	3775 to 4095	321 to 3774

There may be feedback howl and incorrect playback, depending on the system, if the recording of the sound input from the microphone and playback of that recorded sound were performed at the same time. Use appropriate caution.

## 17 Real-Time Clock (RTC)

The DS has an internal Real-Time Clock (RTC). Time is kept by means of an auto-calendar feature that extends through 2099 and accounts for leap years. The maximum error for the clock is  $\pm 4$  seconds/day.

The time is set on the following occasions:

1. When the power is turned on for the first time after the unit is purchased.
2. When the power is turned on after changing the battery.
3. When the unit is restarted after the batteries have been drained (unit has been sitting for several months with no charge in the batteries).
4. When the date and time are set from the boot menu.

When the RTC API is used, the RTC can be used without regard for subprocessor operations. The real-time data shown in Table 17-1 can be read with this API.

**Table 17-1 : Real-Time Data**

Data Type	Data Description
Real-Time Data	Year (00 - 99), month (01 - 12), date (01 - 31), day (00 - 06), hours (00 - 23), minutes (00 - 59), seconds (00 - 59)  Each value is stored as a binary coded decimal (BCD) value.

In addition, two types of alarm functions are provided. By setting the alarm from the application and engaging sleep mode, the unit can be awakened from sleep mode at a specified time. For information on sleep mode, see the section titled Power Management (cross-reference). The API can be used to read and write the following settings for Alarm 1 and Alarm 2.

**Table 17-2 : Settings for Alarm 1 and Alarm 2**

Data Type	Data Description
Alarm 1 and Alarm 2	Day (00 - 06), hour (00 - 23), minutes (00 - 59)  Separate settings for alarms 1 and 2 can be specified. Each value is stored as a binary coded decimal (BCD) value. Day, hour, and time can be enabled or disabled.  Example: The alarm can be set to activate at the same time every day by disabling the day setting.

**Note:** Real-time data cannot be written from the application to the RTC.





## 18 Internal Flash Memory

The NITRO has internal flash memory to store touch panel calibration data, owner information, the NITRO initial setting data, and RTC operation information data. The internal flash memory is dedicated memory for the storage of NITRO setting data; the application cannot write to the internal flash memory.

### 18.1 Touch Panel Calibration Data

This calibration data is used to compensate for the variation in coordinate positioning data between individual touch panels. If an application uses the touch panel, the touch panel must be set by reading the calibration data with the API.

### 18.2 Owner Information Data

The owner information data stores information about the owner of the DS. Using the API, the owner information in Table 18-1 can be read.

**Table 18-1 : Owner Information Data**

Data Type	Data Contents
Owner Information Data	User ID (22 bytes), User color (1 byte), Birthday (2 bytes), Comment (46 bytes)

The details of the owner information data are shown below.

- User ID (Nickname)
  - Nickname string : Maximum of 10 Unicode (UTF16) characters (20 bytes). No termination code.
  - String length : Nickname string length (2 bytes).
- User Color (Favorite color)
  - 0-15 : Selected from a set of 16 colors determined by IPL. (RGB values are enclosed by parentheses.)

0: GRAY	(12,16,19)	1: BROWN	(23, 9, 0)
2: RED	(31, 0, 3)	3: PINK	(31,17,31)
4: ORANGE	(31,18, 0)	5: YELLOW	(30,28, 0)
6: LIME GREEN	(21,31, 0)	7: GREEN	( 0,31, 0)
8: DARK GREEN	( 0,20, 7)	9: SEA GREEN	( 9,27,17)
10: TURQUOISE	( 6,23,30)	11: BLUE	( 0,11,30)
12: DARK BLUE	( 0, 0,18)	13: PURPLE	(17, 0,26)
14: VIOLET	(26, 0,29)	15: MAGENTA	(31, 0,18)
- Birthday (month and day) (Each is stored as a binary-coded decimal number.)
  - Month (1 byte) : Month of birth (01-12)
  - Day (1 byte) : Day of birth (01-31)
- Comment
  - A comment of two lines of a maximum of 23 characters each (23 bytes x 2 = 46 bytes) using Unicode (UTF16).

### 18.3 NITRO Initial Setting Data

The NITRO initial setting data stores which LCD screen is used in AGB mode and the language setting. Using the API, the NITRO initial setting data in Table 18-2 can be read.

**Table 18-2 : NITRO Initial Setting Data**

Data Type	Data Content
NITRO Initial Setting Data	The LCD screen used in AGB mode (2 bytes) and the language setting (4 bytes)

The details of the NITRO initial setting data are shown below:

- LCD screen used in AGB mode
  - 0 : Upper Screen
  - 1 : Lower Screen
- Language setting
  - 0 : Japanese
  - 1 : English
  - 2 : French
  - 3 : German
  - 4 : Italian
  - 5 : Spanish
  - 6 : Chinese
  - 7 : Korean

### 18.4 RTC Operation Information Data

When setting the real-time clock (RTC), the difference in seconds with the initial data is set. This information can be used to determine if the user has changed the RTC data.

Using the API, the RTC operation information data in Table 18-3 can be read.

**Table 18-3 : RTC Operation Information Data**

Data Type	Data Content
RTC Operation Information Data	RTC offset value. This value changes each time the RTC setting is changed.

## Appendix A. Register List

### A.1 Addresses 0x04000000 and Higher

Address Offset	ARM9 Register Name	Page	Description
0x000	DISPCNT	<a href="#">55</a>	2D Graphics Engine A display control
0x002			
0x004	DISPSTAT	<a href="#">51</a>	Display status
0x006	VCOUNT	<a href="#">53</a>	V count comparison
0x008	BG0CNT	<a href="#">81</a>	2D Graphics Engine A BG0 control
0x00a	BG1CNT	<a href="#">81</a>	2D Graphics Engine A BG1 control
0x00c	BG2CNT	<a href="#">83</a>	2D Graphics Engine A BG2 control
0x00e	BG3CNT	<a href="#">83</a>	2D Graphics Engine A BG3 control
0x010	BG0HOFS	<a href="#">105</a>	2D Graphics Engine A BG0 display H offset
0x012	BG0VOFS	<a href="#">105</a>	2D Graphics Engine A BG0 display V offset
0x014	BG1HOFS	<a href="#">105</a>	2D Graphics Engine A BG1 display H offset
0x016	BG1VOFS	<a href="#">105</a>	2D Graphics Engine A BG1 display V offset
0x018	BG2HOFS	<a href="#">105</a>	2D Graphics Engine A BG2 display H offset
0x01a	BG2VOFS	<a href="#">105</a>	2D Graphics Engine A BG2 display V offset
0x01c	BG3HOFS	<a href="#">105</a>	2D Graphics Engine A BG3 display H offset
0x01e	BG3VOFS	<a href="#">105</a>	2D Graphics Engine A BG3 display V offset
0x020	BG2PA	<a href="#">108</a>	2D Graphics Engine A BG2 affine transformation parameters (same line X-direction reference shift dx)
0x022	BG2PB	<a href="#">108</a>	2D Graphics Engine A BG2 affine transformation parameters (next line X-direction reference shift dmx)
0x024	BG2PC	<a href="#">108</a>	2D Graphics Engine A BG2 affine transformation parameters (same line Y-direction reference shift dy)
0x026	BG2PD	<a href="#">108</a>	2D Graphics Engine A BG2 affine transformation parameters (next line Y-direction reference shift dmy)
0x028	BG2X	<a href="#">107</a>	2D Graphics Engine A BG2 reference start point (x coordinate)
0x02a			
0x02c	BG2Y	<a href="#">107</a>	2D Graphics Engine A BG2 reference start point (y coordinate)
0x02e			
0x030	BG3PA	<a href="#">108</a>	2D Graphics Engine A BG3 affine transformation parameters (same line X-direction reference shift dx)
0x032	BG3PB	<a href="#">108</a>	2D Graphics Engine A BG3 affine transformation parameters (next line X-direction reference shift dmx)

Address Offset	ARM9 Register Name	Page	Description
0x034	BG3PC	<a href="#">108</a>	2D Graphics Engine A BG3 affine transformation parameters (same line Y-direction reference shift dy)
0x036	BG3PD	<a href="#">108</a>	2D Graphics Engine A BG3 affine transformation parameters (next line Y-direction reference shift dmy)
0x038	BG3X	<a href="#">107</a>	2D Graphics Engine A BG3 reference start point (x coordinate)
0x03a			
0x03c	BG3Y	<a href="#">107</a>	2D Graphics Engine A BG3 reference start point (y coordinate)
0x03e			

Address Offset	ARM9 Register Name	Page	Description
0x040	WIN0H	<a href="#">143</a>	2D Graphics Engine A window 0 H size
0x042	WIN1H	<a href="#">143</a>	2D Graphics Engine A window 1 H size
0x044	WIN0V	<a href="#">143</a>	2D Graphics Engine A window 0 V size
0x046	WIN1V	<a href="#">143</a>	2D Graphics Engine A window 1 V size
0x048	WININ	<a href="#">142</a>	2D Graphics Engine A window inside
0x04a	WINOUT	<a href="#">142</a>	2D Graphics Engine A window outside
0x04c	MOSAIC	<a href="#">150</a>	2D Graphics Engine A mosaic size
0x04e			
0x050	BLDCNT	<a href="#">146</a>	2D Graphics Engine A color special effects
0x052	BLDALPHA	<a href="#">148</a>	2D Graphics Engine A alpha blending factor
0x054	BLDY	<a href="#">149</a>	2D Graphics Engine A brightness change factor
0x056			
0x058			
0x05a			
0x05c			
0x05e			
0x060	DISP3DCNT	<a href="#">155</a>	3D display control
0x062			
0x064	DISPCAPCNT	<a href="#">67</a>	Display capture
0x066			
0x068	DISP_MMEN_FIFO	<a href="#">65</a>	Main memory display FIFO
0x06a			
0x06c	MASTER_BRIGHT	<a href="#">71</a>	Image output A master brightness
0x06e			
0x070			
0x072			
0x074			
0x076			
0x078			
0x07a			
0x07c			
0x07e			
0x080			
0x082			
0x084			
0x086			
0x088			
0x08a			

Address Offset	ARM9 Register Name	Page	Description
0x08c			
0x08e			
0x090			
0x092			
0x094			
0x096			
0x098			
0x09a			
0x09c			
0x09e			

Address Offset	ARM9 Register Name	Page	Explanation
0x0a0			
0x0a2			
0x0a4			
0x0a6			
0x0a8			
0x0aa			
0x0ac			
0x0ae			
0x0b0	DMA0SAD	<a href="#">273</a>	DMA0 source address
0x0b2			
0x0b4	DMA0DAD	<a href="#">273</a>	DMA0 destination address
0x0b6			
0x0b8	DMA0CNT	<a href="#">274</a>	DMA0 control
0x0ba			
0x0bc	DMA1SAD	<a href="#">273</a>	DMA1 source address
0x0be			
0x0c0	DMA1DAD	<a href="#">273</a>	DMA1 destination address
0x0c2			
0x0c4	DMA1CNT	<a href="#">274</a>	DMA1 control
0x0c6			
0x0c8	DMA2SAD	<a href="#">273</a>	DMA2 source address
0x0ca			
0x0cc	DMA2DAD	<a href="#">273</a>	DMA2 destination address
0x0ce			
0x0d0	DMA2CNT	<a href="#">274</a>	DMA2 control
0x0d2			
0x0d4	DMA3SAD	<a href="#">273</a>	DMA3 source address
0x0d6			
0x0d8	DMA3DAD	<a href="#">273</a>	DMA3 destination address
0x0da			
0x0dc	DMA3CNT	<a href="#">274</a>	DMA3 control
0x0de			
0x0e0			
0x0e2			
0x0e4			
0x0e6			
0x0e8			
0x0ea			

Address Offset	ARM9 Register Name	Page	Explanation
0x0ec			
0x0ee			
0x0f0			
0x0f2			
0x0f4			
0x0f6			
0x0f8			
0x0fa			
0x0fc			
0x0fe			



Address Offset	ARM9 Register Name	Page	Explanation
0x100	TM0CNT_L	<a href="#">279</a>	Timer 0 counter
0x102	TM0CNT_H	<a href="#">279</a>	Timer 0 control
0x104	TM1CNT_L	<a href="#">280</a>	Timer 1 counter
0x106	TM1CNT_H	<a href="#">280</a>	Timer 1 control
0x108	TM2CNT_L	<a href="#">280</a>	Timer 2 counter
0x10a	TM2CNT_H	<a href="#">280</a>	Timer 2 control
0x10c	TM3CNT_L	<a href="#">280</a>	Timer 3 counter
0x10e	TM3CNT_H	<a href="#">280</a>	Timer 3 control
0x110			
0x112			
0x114			
0x116			
0x118			
0x11a			
0x11c			
0x11e			
0x120			
0x122			
0x124			
0x126			
0x128			
0x12a			
0x12c			
0x12e			
0x130	KEYINPUT	<a href="#">297</a>	Key input
0x132	KEYCNT	<a href="#">298</a>	Key control
0x134			
0x136			
0x138			
0x13a			
0x13c			
0x13e			
0x140			
0x142			
0x144			
0x146			
0x148			
0x14a			

Address Offset	ARM9 Register Name	Page	Explanation
0x14c			
0x14e			
0x150			
0x152			
0x154			
0x156			
0x158			
0x15a			
0x15c			
0x15e			

Address Offset	ARM9 Register Name	Page	Explanation
0x160			
0x162			
0x164			
0x166			
0x168			
0x16a			
0x16c			
0x16e			
0x170			
0x172			
0x174			
0x176			
0x178			
0x17a			
0x17c			
0x17e			
0x180			
0x182			
0x184			
0x186			
0x188			
0x18a			
0x18c			
0x18e			
0x190			
0x192			
0x194			
0x196			
0x198			
0x19a			
0x19c			
0x19e			
0x1a0			
0x1a2			
0x1a4			
0x1a6			
0x1a8			
0x1aa			

Address Offset	ARM9 Register Name	Page	Explanation
0x1ac			
0x1ae			
0x1b0			
0x1b2			
0x1b4			
0x1b6			
0x1b8			
0x1ba			
0x1bc			
0x1be			

Address Offset	ARM9 Register Name	Page	Explanation
0x1c0			
0x1c2			
0x1c4			
0x1c6			
0x1c8			
0x1ca			
0x1cc			
0x1ce			
0x1d0			
0x1d2			
0x1d4			
0x1d6			
0x1d8			
0x1da			
0x1dc			
0x1de			
0x1e0			
0x1e2			
0x1e4			
0x1e6			
0x1e8			
0x1ea			
0x1ec			
0x1ee			
0x1f0			
0x1f2			
0x1f4			
0x1f6			
0x1f8			
0x1fa			
0x1fc			
0x1fe			
0x200			
0x202			
0x204	EXMEMCNT	<a href="#">11</a>	External memory control
0x206			
0x208	IME	<a href="#">281</a>	Interrupt master flag
0x20a			

Address Offset	ARM9 Register Name	Page	Explanation
0x20c			
0x20e			
0x210	IE	<a href="#">282</a>	Interrupt enable flag
0x212			
0x214	IF	<a href="#">283</a>	Interrupt request flag
0x216			
0x218			
0x21a			
0x21c			
0x21e			

Address Offset	ARM9 Register Name	Page	Explanation
0x220			
0x222			
0x224			
0x226			
0x228			
0x22a			
0x22c			
0x22e			
0x230			
0x232			
0x234			
0x236			
0x238			
0x23a			
0x23c			
0x23e			
0x240	VRAMCNT	<a href="#">19</a>	RAM bank control 0
0x242			
0x244	WVRAMCNT	<a href="#">23</a>	RAM bank control 1
0x246			
0x248	VRAM_HI_CNT	<a href="#">26</a>	RAM bank control 2
0x24a			
0x24c			
0x24e			
0x250			
0x252			
0x254			
0x256			
0x258			
0x25a			
0x25c			
0x25e			
0x260			
0x262			
0x264			
0x266			
0x268			
0x26a			

Address Offset	ARM9 Register Name	Page	Explanation
0x26c			
0x26e			



Address Offset	ARM9 Register Name	Page	Explanation
0x270			
0x272			
0x274			
0x276			
0x278			
0x27a			
0x27c			
0x27e			
0x280	DIVCNT	<a href="#">291</a>	Divider control
0x282			
0x284			
0x286			
0x288			
0x28a			
0x28c			
0x28e			
0x290	DIV_NUMER	<a href="#">291</a>	Numerator
0x292			
0x294			
0x296			
0x298	DIV_DENOM	<a href="#">291</a>	Denominator
0x29a			
0x29c			
0x29e			
0x2a0	DIV_RESULT	<a href="#">291</a>	Quotient
0x2a2			
0x2a4			
0x2a6			
0x2a8	DIVREM_RESULT	<a href="#">291</a>	Remainder
0x2aa			
0x2ac			
0x2ae			
0x2b0	SQRTCNT	<a href="#">294</a>	Square root unit control
0x2b2			
0x2b4	SQRT_RESULT	<a href="#">294</a>	Square root unit result
0x2b6			

Address Offset	ARM9 Register Name	Page	Explanation
0x2b8	SQRT_PARAM	<a href="#">294</a>	Square root unit data
0x2ba			
0x2bc			
0x2be			
0x2c0			
0x2c2			
0x2c4			
0x2c6			
0x2c8			
0x2ca			
0x2cc			
0x2ce			

Address Offset	ARM9 Register Name	Page	Explanation
0x2d0			
0x2d2			
0x2d4			
0x2d6			
0x2d8			
0x2da			
0x2dc			
0x2de			
0x2e0			
0x2e2			
0x2e4			
0x2e6			
0x2e8			
0x2ea			
0x2ec			
0x2ee			
0x2f0			
0x2f2			
0x2f4			
0x2f6			
0x2f8			
0x2fa			
0x2fc			
0x2fe			
0x300			
0x302			
0x304	POWCNT	<a href="#">54</a>	Power control
0x306			
0x308			
0x30a			
0x30c			
0x30e			
0x310			
0x312			
0x314			
0x316			
0x318			
0x31a			

Address Offset	ARM9 Register Name	Page	Explanation
0x31c			
0x31e			

Address Offset	ARM9 Register Name	Page	Explanation
0x320	RDLINE_COUNT	<a href="#">267</a>	Rendering minimum fill
0x322			
0x324			
0x326			
0x328			
0x32a			
0x32c			
0x32e			
0x330	EDGE_COLOR_0_L	<a href="#">259</a>	Edge marking color (Polygon ID's 3 upper bits are 000)
0x332	EDGE_COLOR_0_H	<a href="#">259</a>	Edge marking color (Polygon ID's 3 upper bits are 001)
0x334	EDGE_COLOR_1_L	<a href="#">259</a>	Edge marking color (Polygon ID's 3 upper bits are 010)
0x336	EDGE_COLOR_1_H	<a href="#">259</a>	Edge marking color (Polygon ID's 3 upper bits are 011)
0x338	EDGE_COLOR_2_L	<a href="#">259</a>	Edge marking color (Polygon ID's 3 upper bits are 100)
0x33a	EDGE_COLOR_2_H	<a href="#">259</a>	Edge marking color (Polygon ID's 3 upper bits are 101)
0x33c	EDGE_COLOR_3_L	<a href="#">259</a>	Edge marking color (Polygon ID's 3 upper bits are 110)
0x33e	EDGE_COLOR_3_H	<a href="#">259</a>	Edge marking color (Polygon ID's 3 upper bits are 111)
0x340	ALPHA_TEST_REF	<a href="#">258</a>	Alpha test
0x342			
0x344			
0x346			
0x348			
0x34a			
0x34c			
0x34e			
0x350	CLEAR_COLOR	<a href="#">231</a>	Color buffer initial value
0x352			
0x354	CLEAR_DEPTH	<a href="#">231</a>	Depth buffer initial value
0x356	CLRIMAGE_OFFSET	<a href="#">233</a>	Clear image offset

Address Offset	ARM9 Register Name	Page	Explanation
0x358	FOG_COLOR	<a href="#">260</a>	Fog color
0x35a			
0x35c	FOG_OFFSET	<a href="#">260</a>	Fog offset
0x35e			
0x360	FOG_TABLE_0_L	<a href="#">261</a>	Fog density table (0, 1)
0x362	FOG_TABLE_0_H	<a href="#">261</a>	Fog density table (2, 3)
0x364	FOG_TABLE_1_L	<a href="#">261</a>	Fog density table (4, 5)
0x366	FOG_TABLE_1_H	<a href="#">261</a>	Fog density table (6, 7)
0x368	FOG_TABLE_2_L	<a href="#">261</a>	Fog density table (8, 9)
0x36a	FOG_TABLE_2_H	<a href="#">261</a>	Fog density table (10, 11)
0x36c	FOG_TABLE_3_L	<a href="#">261</a>	Fog density table (12, 13)
0x36e	FOG_TABLE_3_H	<a href="#">261</a>	Fog density table (14, 15)
0x370	FOG_TABLE_4_L	<a href="#">261</a>	Fog density table (16, 17)
0x372	FOG_TABLE_4_H	<a href="#">261</a>	Fog density table (18, 19)
0x374	FOG_TABLE_5_L	<a href="#">261</a>	Fog density table (20, 21)
0x376	FOG_TABLE_5_H	<a href="#">261</a>	Fog density table (22, 23)
0x378	FOG_TABLE_6_L	<a href="#">261</a>	Fog density table (24, 25)
0x37a	FOG_TABLE_6_H	<a href="#">261</a>	Fog density table (26, 27)
0x37c	FOG_TABLE_7_L	<a href="#">261</a>	Fog density table (28, 29)
0x37e	FOG_TABLE_7_H	<a href="#">261</a>	Fog density table (30, 31)
0x380	TOON_TABLE_0_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 0)
0x382	TOON_TABLE_0_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 1)
0x384	TOON_TABLE_1_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 2)
0x386	TOON_TABLE_1_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 3)
0x388	TOON_TABLE_2_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 4)
0x38a	TOON_TABLE_2_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 5)
0x38c	TOON_TABLE_3_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 6)
0x38e	TOON_TABLE_3_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 7)
0x390	TOON_TABLE_4_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 8)
0x392	TOON_TABLE_4_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 9)

Address Offset	ARM9 Register Name	Page	Explanation
0x394	TOON_TABLE_5_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 10)
0x396	TOON_TABLE_5_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 11)
0x398	TOON_TABLE_6_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 12)
0x39a	TOON_TABLE_6_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 13)
0x39c	TOON_TABLE_7_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 14)
0x39e	TOON_TABLE_7_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 15)
0x3a0	TOON_TABLE_8_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 16)
0x3a2	TOON_TABLE_8_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 17)
0x3a4	TOON_TABLE_9_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 18)
0x3a6	TOON_TABLE_9_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 19)
0x3a8	TOON_TABLE_10_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 20)
0x3aa	TOON_TABLE_10_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 21)
0x3ac	TOON_TABLE_11_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 22)
0x3ae	TOON_TABLE_11_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 23)
0x3b0	TOON_TABLE_12_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 24)
0x3b2	TOON_TABLE_12_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 25)
0x3b4	TOON_TABLE_13_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 26)
0x3b6	TOON_TABLE_13_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 27)
0x3b8	TOON_TABLE_14_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 28)
0x3ba	TOON_TABLE_14_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 29)
0x3bc	TOON_TABLE_15_L	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 30)
0x3be	TOON_TABLE_15_H	<a href="#">241</a>	Toon table (RGB conversion value when brightness is 31)

Address Offset	ARM9 Register Name	Page	Explanation
0x3c0			
0x3c2			
0x3c4			
0x3c6			
0x3c8			
0x3ca			
0x3cc			
0x3ce			
0x3d0			
0x3d2			
0x3d4			
0x3d6			
0x3d8			
0x3da			
0x3dc			
0x3de			
0x3e0			
0x3e2			
0x3e4			
0x3e6			
0x3e8			
0x3ea			
0x3ec			
0x3ee			
0x3f0			
0x3f2			
0x3f4			
0x3f6			
0x3f8			
0x3fa			
0x3fc			
0x3fe			



Address Offset	ARM9 Register Name	Page	Explanation
0x400	GXFIFO	<a href="#">168</a>	Geometry command FIFO
0x402			
0x404	GXFIFO image		
0x406			
0x408			
0x40a			
0x40c			
0x40e			
0x410			
0x412			
0x414			
0x416			
0x418			
0x41a			
0x41c			
0x41e			
0x420			
0x422			
0x424			
0x426			
0x428			
0x42a			
0x42c			
0x42e			
0x430			
0x432			
0x434			
0x436			
0x438			
0x43a			
0x43c			
0x43e			

Address Offset	ARM9 Register Name	Page	Explanation
0x440	MTX_MODE	<a href="#">181</a>	Current matrix mode setting
0x442			
0x444	MTX_PUSH	<a href="#">186</a>	Push current matrix onto the stack
0x446			
0x448	MTX_POP	<a href="#">186</a>	Pop current matrix from the stack
0x44a			
0x44c	MTX_STORE	<a href="#">187</a>	Store current matrix in specified position in the stack
0x44e			
0x450	MTX_RESTORE	<a href="#">187</a>	Read matrix from specified position in the stack
0x452			
0x454	MTX_IDENTITY	<a href="#">182</a>	Initialize unit matrix
0x456			
0x458	MTX_LOAD_4x4	<a href="#">182</a>	Set 4x4 matrix
0x45a			
0x45c	MTX_LOAD_4x3	<a href="#">183</a>	Set 4x3 matrix
0x45e			
0x460	MTX_MULT_4x4	<a href="#">183</a>	Multiply by 4x4 matrix
0x462			
0x464	MTX_MULT_4x3	<a href="#">183</a>	Multiply by 4x3 matrix
0x466			
0x468	MTX_MULT_3x3	<a href="#">184</a>	Multiply by 3x3 matrix
0x46a			
0x46c	MTX_SCALE	<a href="#">185</a>	Multiply by the Scale matrix
0x46e			
0x470	MTX_TRANS	<a href="#">184</a>	Multiply by the Translation matrix
0x472			
0x474			
0x476			
0x478			
0x47a			
0x47c			
0x47e			

Address Offset	ARM9 Register Name	Page	Explanation
0x480	COLOR	<a href="#">201</a>	Vertex color
0x482			
0x484	NORMAL	<a href="#">202</a>	Normal vector
0x486			
0x488	TEXCOORD	<a href="#">206</a>	Texture coordinates
0x48a			
0x48c	VTX_16	<a href="#">202</a>	Vertex coordinates
0x48e			
0x490	VTX_10	<a href="#">203</a>	Vertex coordinates
0x492			
0x494	VTX_XY	<a href="#">203</a>	Vertex XY coordinates
0x496			
0x498	VTX_XZ	<a href="#">203</a>	Vertex XZ coordinates
0x49a			
0x49c	VTX_YZ	<a href="#">203</a>	Vertex YZ coordinates
0x49e			
0x4a0	VTX_DIFF	<a href="#">204</a>	Vertex coordinates difference value specification
0x4a2			
0x4a4	POLYGON_ATTR	<a href="#">196</a>	Polygon-related attribute values
0x4a6			
0x4a8	TEXIMAGE_PARAM	<a href="#">207</a>	Texture parameters
0x4aa			
0x4ac	TEXPLTT_BASE	<a href="#">212</a>	Texture palette base address
0x4ae			
0x4b0			
0x4b2			
0x4b4			
0x4b6			
0x4b8			
0x4ba			
0x4bc			
0x4be			
0x4c0	DIF_AMB	<a href="#">192</a>	Material's diffuse and ambient colors
0x4c2			
0x4c4	SPE_EMI	<a href="#">192</a>	Material's specular reflection and emitted light colors
0x4c6			
0x4c8	LIGHT_VECTOR	<a href="#">189</a>	Light direction vector
0x4ca			

Address Offset	ARM9 Register Name	Page	Explanation
0x4cc	LIGHT_COLOR	<a href="#">189</a>	Light color
0x4ce			
0x4d0	SHININESS	<a href="#">193</a>	Specular reflection shininess table
0x4d2			
0x4d4			
0x4d6			
0x4d8			
0x4da			
0x4dc			
0x4de			

Address Offset	ARM9 Register Name	Page	Explanation
0x4e0			
0x4e2			
0x4e4			
0x4e6			
0x4e8			
0x4ea			
0x4ec			
0x4ee			
0x4f0			
0x4f2			
0x4f4			
0x4f6			
0x4f8			
0x4fa			
0x4fc			
0x4fe			
0x500	BEGIN_VTXS	<a href="#">200</a>	Vertex list start
0x502			
0x504	END_VTXS	<a href="#">201</a>	Vertex list end
0x506			
0x508			
0x50a			
0x50c			
0x50e			
0x510			
0x512			
0x514			
0x516			
0x518			
0x51a			
0x51c			
0x51e			
0x520			
0x522			
0x524			
0x526			
0x528			
0x52a			

Address Offset	ARM9 Register Name	Page	Explanation
0x52c			
0x52e			
0x530			
0x532			
0x534			
0x536			
0x538			
0x53a			
0x53c			
0x53e			

Address Offset	ARM9 Register Name	Page	Explanation
0x540	SWAP_BUFFERS	<a href="#">178</a>	Swap data group
0x542			
0x544			
0x546			
0x548			
0x54a			
0x54c			
0x54e			
0x550			
0x552			
0x554			
0x556			
0x558			
0x55a			
0x55c			
0x55e			
0x560			
0x562			
0x564			
0x566			
0x568			
0x56a			
0x56c			
0x56e			
0x570			
0x572			
0x574			
0x576			
0x578			
0x57a			
0x57c			
0x57e			
0x580	VIEWPORT	<a href="#">180</a>	Viewport
0x582			
0x584			
0x586			
0x588			
0x58a			

Address Offset	ARM9 Register Name	Page	Explanation
0x58c			
0x58e			
0x590			
0x592			
0x594			
0x596			
0x598			
0x59a			
0x59c			
0x59e			



Address Offset	ARM9 Register Name	Page	Explanation
0x5a0			
0x5a2			
0x5a4			
0x5a6			
0x5a8			
0x5aa			
0x5ac			
0x5ae			
0x5b0			
0x5b2			
0x5b4			
0x5b6			
0x5b8			
0x5ba			
0x5bc			
0x5be			
0x5c0	BOX_TEST	<a href="#">216</a>	Box test
0x5c2			
0x5c4	POS_TEST	<a href="#">218</a>	Position coordinate test
0x5c6			
0x5c8	VEC_TEST	<a href="#">218</a>	Direction vector test
0x5ca			
0x5cc			
0x5ce			
0x5d0			
0x5d2			
0x5d4			
0x5d6			
0x5d8			
0x5da			
0x5dc			
0x5de			
0x5e0			
0x5e2			
0x5e4			
0x5e6			
0x5e8			
0x5ea			

Address Offset	ARM9 Register Name	Page	Explanation
0x5ec			
0x5ee			
0x5f0			
0x5f2			
0x5f4			
0x5f6			
0x5f8			
0x5fa			
0x5fc			
0x5fe			

Address Offset	ARM9 Register Name	Page	Explanation
0x600	GXSTAT	<a href="#">220</a>	Geometry engine status
0x602			
0x604	LISTRAM_COUNT	<a href="#">222</a>	Polygon list RAM count
0x606	VTXRAM_COUNT	<a href="#">222</a>	Vertex RAM count
0x608			
0x60a			
0x60c			
0x60e			
0x610	DISP_1DOT_DEpTH	<a href="#">199</a>	1-dot polygon display boundary depth value
0x612			
0x614			
0x616			
0x618			
0x61a			
0x61c			
0x61e			
0x620	POS_RESULT_X	<a href="#">218</a>	Result of position coordinate test (clip coordinate X component)
0x622			
0x624	POS_RESULT_Y	<a href="#">218</a>	Result of position coordinate test (clip coordinate Y component)
0x626			
0x628	POS_RESULT_Z	<a href="#">218</a>	Result of position coordinate test (clip coordinate Z component)
0x62a			
0x62c	POS_RESULT_W	<a href="#">218</a>	Result of position coordinate test (clip coordinate W component)
0x62e			
0x630	VEC_RESULT_X	<a href="#">219</a>	Result of direction vector test (X component)
0x632	VEC_RESULT_Y	<a href="#">219</a>	Result of direction vector test (Y component)
0x634	VEC_RESULT_Z	<a href="#">219</a>	Result of direction vector test (Z component)
0x636			
0x638			
0x63a			
0x63c			
0x63e			

Address Offset	ARM9 Register Name	Page	Explanation
0x640	CLIPMTX_RESULT_0	<a href="#">188</a>	Current clip coordinate matrix (element m0)
0x642			
0x644	CLIPMTX_RESULT_1	<a href="#">188</a>	Current clip coordinate matrix (element m1)
0x646			
0x648	CLIPMTX_RESULT_2	<a href="#">188</a>	Current clip coordinate matrix (element m2)
0x64a			
0x64c	CLIPMTX_RESULT_3	<a href="#">188</a>	Current clip coordinate matrix (element m3)
0x64e			
0x650	CLIPMTX_RESULT_4	<a href="#">188</a>	Current clip coordinate matrix (element m4)
0x652			
0x654	CLIPMTX_RESULT_5	<a href="#">188</a>	Current clip coordinate matrix (element m5)
0x656			
0x658	CLIPMTX_RESULT_6	<a href="#">188</a>	Current clip coordinate matrix (element m6)
0x65a			
0x65c	CLIPMTX_RESULT_7	<a href="#">188</a>	Current clip coordinate matrix (element m7)
0x65e			
0x660	CLIPMTX_RESULT_8	<a href="#">188</a>	Current clip coordinate matrix (element m8)
0x662			
0x664	CLIPMTX_RESULT_9	<a href="#">188</a>	Current clip coordinate matrix (element m9)
0x666			
0x668	CLIPMTX_RESULT_10	<a href="#">188</a>	Current clip coordinate matrix (element m10)
0x66a			
0x66c	CLIPMTX_RESULT_11	<a href="#">188</a>	Current clip coordinate matrix (element m11)
0x66e			
0x670	CLIPMTX_RESULT_12	<a href="#">188</a>	Current clip coordinate matrix (element m12)
0x672			
0x674	CLIPMTX_RESULT_13	<a href="#">188</a>	Current clip coordinate matrix (element m13)
0x676			
0x678	CLIPMTX_RESULT_14	<a href="#">188</a>	Current clip coordinate matrix (element m14)
0x67a			
0x67c	CLIPMTX_RESULT_15	<a href="#">188</a>	Current clip coordinate matrix (element m15)
0x67e			

Address Offset	ARM9 Register Name	Page	Explanation
0x680	VECMTX_RESULT_0	<a href="#">188</a>	Current direction vector matrix (element m0)
0x682			
0x684	VECMTX_RESULT_1	<a href="#">188</a>	Current direction vector matrix (element m1)
0x686			
0x688	VECMTX_RESULT_2	<a href="#">188</a>	Current direction vector matrix (element m2)
0x68a			
0x68c	VECMTX_RESULT_3	<a href="#">188</a>	Current direction vector matrix (element m3)
0x68e			
0x690	VECMTX_RESULT_4	<a href="#">188</a>	Current direction vector matrix (element m4)
0x692			
0x694	VECMTX_RESULT_5	<a href="#">188</a>	Current direction vector matrix (element m5)
0x696			
0x698	VECMTX_RESULT_6	<a href="#">188</a>	Current direction vector matrix (element m6)
0x69a			
0x69c	VECMTX_RESULT_7	<a href="#">188</a>	Current direction vector matrix (element m7)
0x69e			
0x6a0	VECMTX_RESULT_8	<a href="#">188</a>	Current direction vector matrix (element m8)
0x6a2			
0x6a4			
0x6a6			
0x6a8			
0x6aa			
0x6ac			
0x6ae			
0x6b0			
0x6b2			
0x6b4			
0x6b6			
0x6b8			
0x6ba			
0x6bc			
0x6be			
0x6c0			
0x6c2			
0x6c4			
0x6c6			
0x6c8			
0x6ca			

Address Offset	ARM9 Register Name	Page	Explanation
0x6cc			
0x6ce			
0x6d0			
0x6d2			
0x6d4			
0x6d6			
0x6d8			
0x6da			
0x6dc			
0x6de			

Address Offset	ARM9 Register Name	Page	Explanation
0x6e0			
0x6e2			
0x6e4			
0x6e6			
0x6e8			
0x6ea			
0x6ec			
0x6ee			
0x6f0			
0x6f2			
0x6f4			
0x6f6			
0x6f8			
0x6fa			
0x6fc			
0x6fe			

**A.2 Addresses 0x04001000 and higher (2D Graphics Engine B-related)**

Address Offset	ARM9 Register Name	Page	Explanation
0x000	DB_DISPCNT	<a href="#">57</a>	2D Graphics Engine B display control
0x002			
0x004			
0x006			
0x008	DB_BG0CNT	<a href="#">81</a>	2D Graphics Engine B BG0 control
0x00a	DB_BG1CNT	<a href="#">81</a>	2D Graphics Engine B BG1 control
0x00c	DB_BG2CNT	<a href="#">83</a>	2D Graphics Engine B BG2 control
0x00e	DB_BG3CNT	<a href="#">83</a>	2D Graphics Engine B BG3 control
0x010	DB_BG0HOFS	<a href="#">105</a>	2D Graphics Engine B BG0 display H offset
0x012	DB_BG0VOFS	<a href="#">105</a>	2D Graphics Engine B BG0 display V offset
0x014	DB_BG1HOFS	<a href="#">105</a>	2D Graphics Engine B BG1 display H offset
0x016	DB_BG1VOFS	<a href="#">105</a>	2D Graphics Engine B BG1 display V offset
0x018	DB_BG2HOFS	<a href="#">105</a>	2D Graphics Engine B BG2 display H offset
0x01a	DB_BG2VOFS	<a href="#">105</a>	2D Graphics Engine B BG2 display V offset
0x01c	DB_BG3HOFS	<a href="#">105</a>	2D Graphics Engine B BG3 display H offset
0x01e	DB_BG3VOFS	<a href="#">105</a>	2D Graphics Engine B BG3 display V offset
0x020	DB_BG2PA	<a href="#">108</a>	2D Graphics Engine B BG2 affine transformation parameters (same line X-direction reference shift dx)
0x022	DB_BG2PB	<a href="#">108</a>	2D Graphics Engine B BG2 affine transformation parameters (next line X-direction reference shift dmx)
0x024	DB_BG2PC	<a href="#">108</a>	2D Graphics Engine B BG2 affine transformation parameters (same line Y-direction reference shift dy)
0x026	DB_BG2PD	<a href="#">108</a>	2D Graphics Engine B BG2 affine transformation parameters (next line Y-direction reference shift dmy)
0x028	DB_BG2X	<a href="#">107</a>	2D Graphics Engine B BG2 reference start point (x coordinate)
0x02a			
0x02c	DB_BG2Y	<a href="#">107</a>	2D Graphics Engine B BG2 reference start point (y coordinate)
0x02e			
0x030	DB_BG3PA	<a href="#">108</a>	2D Graphics Engine B BG3 affine transformation parameters (same line X-direction reference shift dx)
0x032	DB_BG3PB	<a href="#">108</a>	2D Graphics Engine B BG3 affine transformation parameters (next line X-direction reference shift dmx)



Address Offset	ARM9 Register Name	Page	Explanation
0x034	DB_BG3PC	<a href="#">108</a>	2D Graphics Engine B BG3 affine transformation parameters (same line Y-direction reference shift dy)
0x036	DB_BG3PD	<a href="#">108</a>	2D Graphics Engine B BG3 affine transformation parameters (next line Y-direction reference shift dmy)
0x038	DB_BG3X	<a href="#">107</a>	2D Graphics Engine B BG3 reference start point (x coordinate)
0x03a			
0x03c	DB_BG3Y	<a href="#">107</a>	2D Graphics Engine B BG3 reference start point (y coordinate)
0x03e			

Address Offset	ARM9 Register Name	Page	Explanation
0x040	DB_WIN0H	<a href="#">143</a>	2D Graphics Engine B window 0H size
0x042	DB_WIN1H	<a href="#">143</a>	2D Graphics Engine B window 1H size
0x044	DB_WIN0V	<a href="#">143</a>	2D Graphics Engine B window 0V size
0x046	DB_WIN1V	<a href="#">143</a>	2D Graphics Engine B window 1V size
0x048	DB_WININ	<a href="#">142</a>	2D Graphics Engine B window inside
0x04a	DB_WINOUT	<a href="#">142</a>	2D Graphics Engine B window outside
0x04c	DB_MOSAIC	<a href="#">150</a>	2D Graphics Engine B mosaic size
0x04e			
0x050	DB_BLDCNT	<a href="#">146</a>	2D Graphics Engine B color special effects
0x052	DB_BLDALPHA	<a href="#">148</a>	2D Graphics Engine B alpha blending factor
0x054	DB_BLDY	<a href="#">149</a>	2D Graphics Engine B brightness conversion factor
0x056			
0x058			
0x05a			
0x05c			
0x05e			
0x060			
0x062			
0x064			
0x066			
0x068			
0x06a			
0x06c	DB_MASTER_BRIGHT	<a href="#">71</a>	Image output B master brightness

**A.3 Addresses 0x04100000 and higher**

Address Offset	ARM9 Register Name	Page	Explanation
0x000			
0x002			
0x004			
0x006			
0x008			
0x00a			
0x00c			
0x00e			
0x010			
0x012			
0x014			
0x016			
0x018			
0x01a			
0x01c			
0x01e			
0x020			
0x022			
0x024			
0x026			
0x028			
0x02a			
0x02c			
0x02e			
0x030			
0x032			
0x034			
0x036			
0x038			
0x03a			
0x03c			
0x03e			
0x040			
0x042			
0x044			
0x046			
0x048			

Address Offset	ARM9 Register Name	Page	Explanation
0x04a			
0x04c			
0x04e			
0x050			
0x052			
0x054			
0x056			
0x058			
0x05a			
0x05c			
0x05e			

Address Offset	ARM9 Register Name	Page	Explanation
0x060			
0x062			
0x064			
0x066			
0x068			
0x06a			
0x06c			
0x06e			
0x070			
0x072			
0x074			
0x076			
0x078			
0x07a			
0x07c			
0x07e			



## Appendix B. List of VRAM Data Capacities

(Data capacity unit: bytes)

Format \ Size	8 x 8	16 x 16	32 x 32	64 x 64	128 x 128	256 x 192	256 x 256	512 x 256	512 x 512	1024 x 512	1024 x 1024
16-Color Character	32	128	512	2K	X	X	X	X	X	X	X
256-Color Character	64	256	1K	4K	X	X	X	X	X	X	X
Direct Color Bitmap OBJ	128	512	2K	8K	X	X	X	X	X	X	X
Normal Character BG Screen	X	X	X	X	X	X	2K	4K	8K	X	X
Rotated Character BG Screen	X	X	X	X	256	X	1K	X	4K	X	16K
Extended/Rotated Character BG Screen	X	X	X	X	512	X	2K	X	8K	X	32K
256-Color Bitmap BG	X	X	X	X	16K	X	64K	128K	256K	X	X
Large Screen 256-Color Bitmap BG	X	X	X	X	X	X	X	X	X	512K	X
Direct Color Bitmap BG	X	X	X	X	32K	X	128K	256K	512K	X	X
Clear Color Image	X	X	X	X	X	96K	X	X	X	X	X
Clear Depth Image	X	X	X	X	X	96K	X	X	X	X	X
Display Capture	X	X	X	X	32K	96K	X	X	X	X	X
4-Color Texture	16	64	256	1K	4K	—	16K	—	64K	—	256K
16-Color Texture	32	128	512	2K	8K	—	32K	—	128K	—	512K
256-Color Texture	64	256	1K	4K	16K	—	64K	—	256K	512K	X
A3I5 Translucent Texture	64	256	1K	4K	16K	—	64K	—	256K	512K	X
A5I3 Translucent Texture	64	256	1K	4K	16K	—	64K	—	256K	512K	X
Direct Color Texture	128	512	2K	8K	32K	—	128K	—	512K	X	X
Compressed Texture Image	16	64	256	1K	4K	—	16K	—	64K	—	256K
Compressed Texture Index	8	32	128	512	2K	—	8K	—	32K	—	128K
Maximum No. of Compressed Texture Interpolation Palettes	16	64	256	1K	4K	—	16K	—	64K	(96K)	(96K)

X: Outside of specifications

—: Omitted

Bold: Maximum value

( ): The maximum data value exceeds the RAM capacity. Therefore the maximum usable RAM capacity is specified.





## Appendix C. Data Formats

### BG, OBJ Character Data

Format \ d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
16-color Character	P3				P2				P1				P0			
256-color Character	P1								P0							

### Bitmap OBJ

Format \ d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Direct Color Bitmap OBJ	ALP HA	BLUE				GREEN				RED						

### BG Screen Data

Format \ d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Normal Character BG Screen	Palette				Vflip	Hflip	Character name									
Rotate Character BG Screen										Character name						
Expand/rotate Character BG Screen	Palette				Vflip	Hflip	Character name									

### Bitmap BG Data

Format \ d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
256-color Bitmap BG										Color No.						
Large Screen 256-color Bitmap BG										Color No.						
Direct Color Bitmap BG	ALP HA	BLUE				GREEN				RED						

### Palette Data

Format \ d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
BG, OBJ Palette Color	-	BLUE				GREEN				RED						

**Other Graphics Function Data**

Format \ d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Clear Color Image	ALP HA	BLUE					GREEN					RED				
Clear Depth Image	FOG	Integer portion												Fractional portion		
		Clear depth														
Display Capture	ALP HA	BLUE					GREEN					RED				

**Texture Data**

Format \ d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
4-color Texture																T0
16-color Texture																T0
256-color Texture																T0
A5I3 Translucent Texture																INDEX
A3I5 Translucent Texture																INDEX
Direct Color Texture	ALP HA	BLUE						GREEN						RED		

**Compressed Texture Data (Note: 32-bit notation)**

Format \ d	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0		
Compressed texture image	T33	T32		T31		T30		T23		T22		T21		T20		T13		T12		T11		T10		T03		T02		T01		T00				
Compressed texture index																	3/4	T	Palette Address															

**Texture Palette Data**

Format	d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Texture palette color	-	BLUE						GREEN				RED					

**OAM Data**

Format	d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
OBJ attribute 0		Form		Color mode	Mosaic	OBJ mode		Double size	Rotation / Scaling	y-Coordinate							
OBJ attribute 1		Size		V-flip	H-flip				x-Coordinate								
	Affine transformation parameter number																
OBJ attribute 2		Palette Number				Order of Priority		Character name									
Affine transformation parameter PA	Sign	Integer part								Decimal part							
	Distance in x direction for same line																
Affine transformation parameter PB	Sign	Integer part								Decimal part							
	Distance in in x direction for next line																
Affine transformation parameter PC	Sign	Integer part								Decimal part							
	Distance in in y direction for same line																
Affine transformation parameter PD	Sign	Integer part								Decimal part							
	Distance in in y direction for next line																

**Sound Data**

Format	d	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PCM 8 bit										Data 0							
PCM16 bit	Data 0																
ADPCM														Data 0			

