HOW TO USE THIS TUTORIAL

These slides were originally developed by Sega of Europe as on-site training materials. For your convenience, we have converted them to a self-viewing tutorial.

Options for viewing the slides:

1. Manually - You may scroll through the slides using the Bookmarks as a general guide or using the arrows in the Tool Bar to go page by page.

2. Full Screen Viewing - This allows you to view the tutorial as a slide presentation. To do this, go to Edit...Preferences... Full Screen. A dialogue box appears allowing you to control the slides by two methods: 1) Manually with the Keyboard; 2) Automatically as a “Slide Show”. These are described below.

A) Manually with the keyboard and mouse.

- Select the Keyboard option in Edit...Preferences...Full Screen.
- From the Main Menu, go to View ... Full Screen. The slides will appear in a full screen view.
- You can scroll backward or forward with the arrow keys, or forward by clicking the mouse.
- To return to the screen with the Bookmarks and Tool Bar, hit escape or command-period.

B) Automatic “Slide Show”, advancing at an interval you choose.

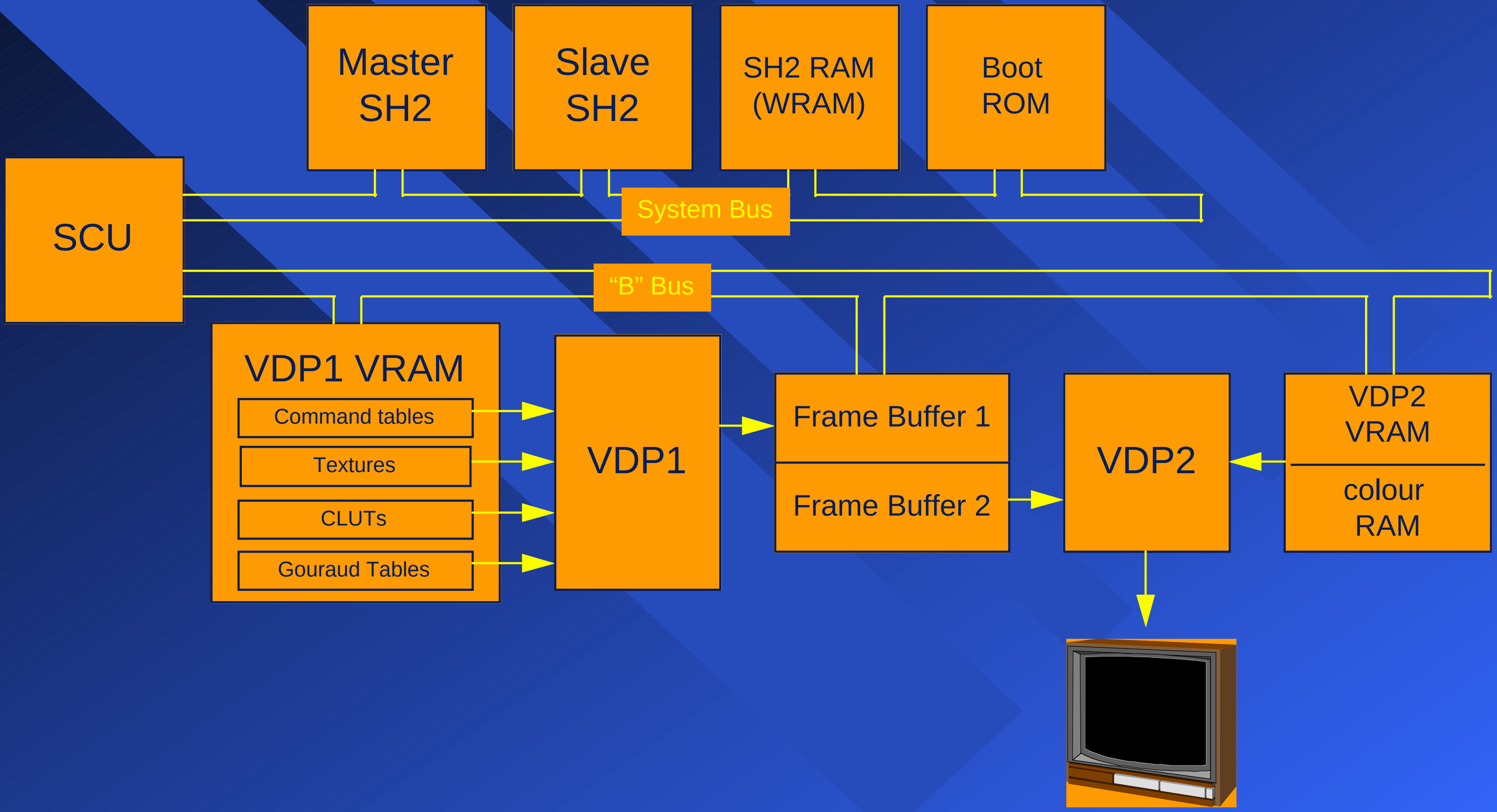
- Select the Loop option in Edit...Preferences...Full Screen.
- Type into the text field the time interval you prefer between slides.
- From the Main Menu, go to View ... Full Screen. The slides will appear in a full screen view and advance automatically.
- To halt this process at the current slide, hit escape.
- To continue with the automatic presentation, return to the View menu and re-select Full Screen. The presentation will continue where you left off.

VDP1 – The Sprite Chip

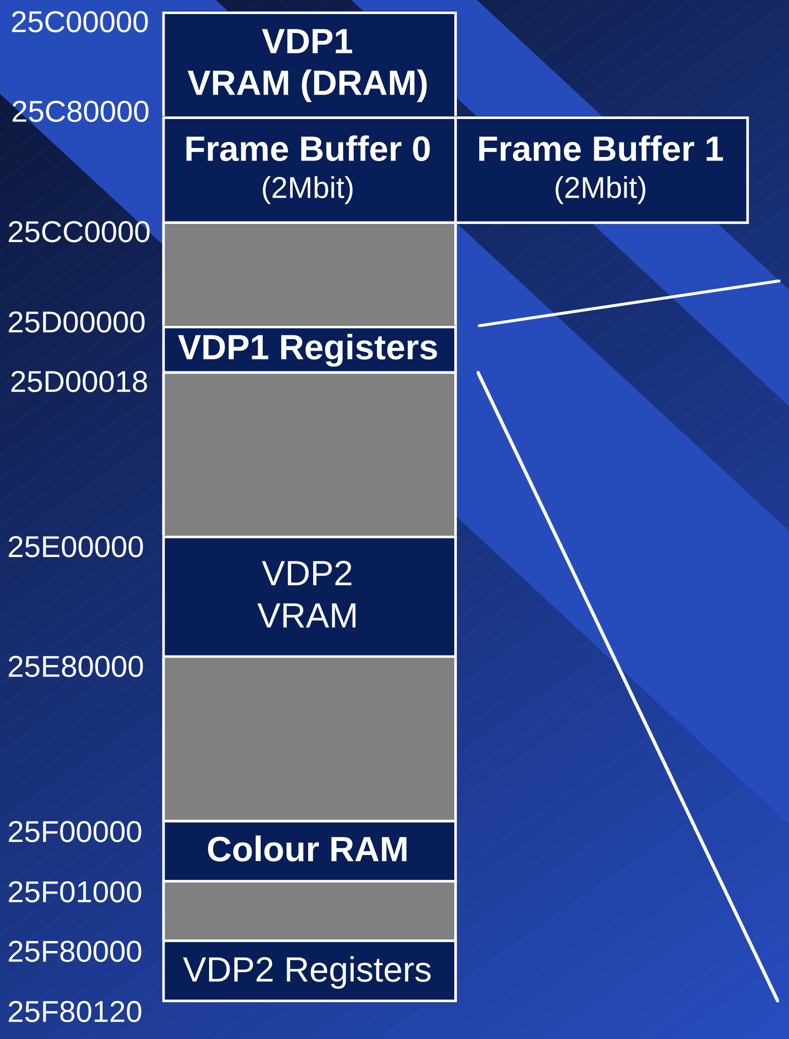
Tips and Tricks

SoE Internal Product Development

Overview - How VDP1 fits into Saturn Architecture



Memory map



25C00000

VDP1
VRAM (DRAM)

25C80000

Frame Buffer 0
(2Mbit)

Frame Buffer 1
(2Mbit)

25CC0000

25D00000

VDP1 Registers

25D00018

25E00000

VDP2
VRAM

25E80000

25F00000

Colour RAM

25F01000

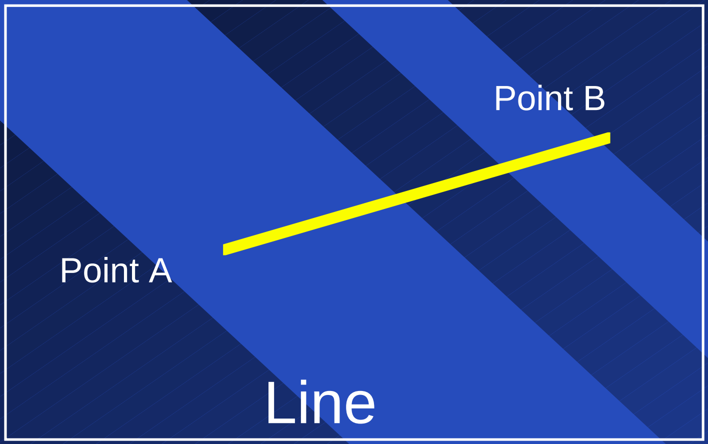
25F80000

VDP2 Registers

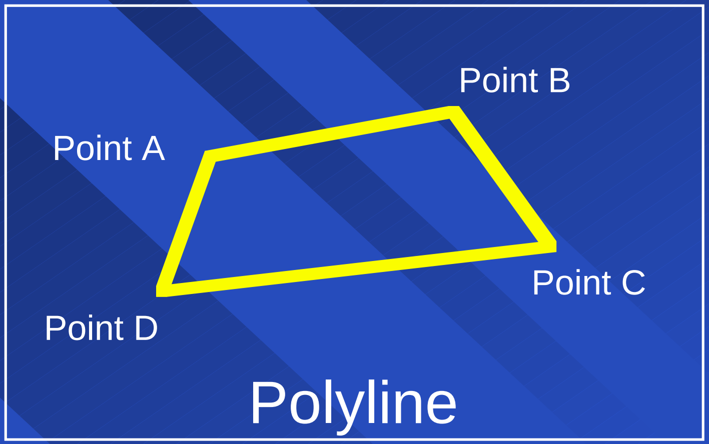
25F80120

Description	Address	Access
TV Mode	25D00000H	W/O
Frame Buffer Change Mode	25D00002H	W/O
Draw Trigger	25D00004H	W/O
Frame Buffer Erase/Write data	25D00006H	W/O
Erase/Writeupper-left coord	25D00008H	W/O
Erase/Write lower right coord	25D0000AH	W/O
Draw abnormal end	25D0000CH	W/O
Transfer end status	25D00010H	R/O
Last operation cmd address	25D00012H	R/O
Curr operation cmd address	25D00014H	R/O
Mode Status	25D00016H	R/O

Part Types - Non-textured parts



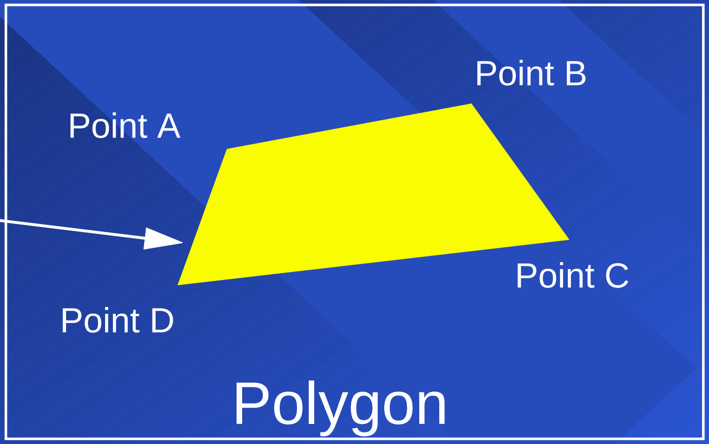
Line



Polyline

(Caveat - Half transparency processing)

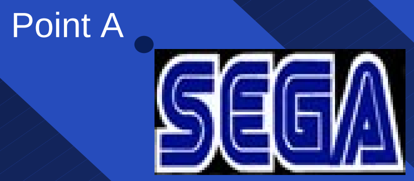
Polygon filled with nontextured colour



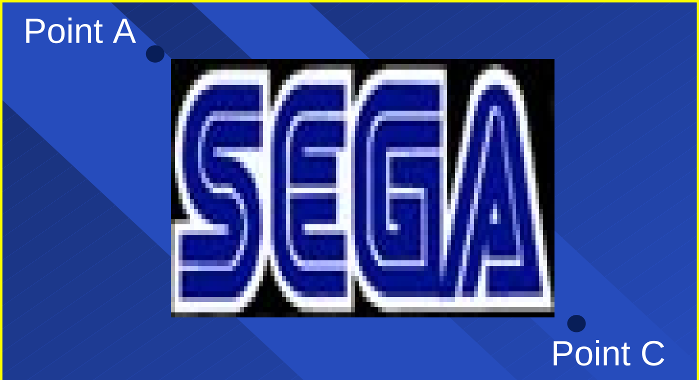
Polygon

(Caveat - Half transparency processing)

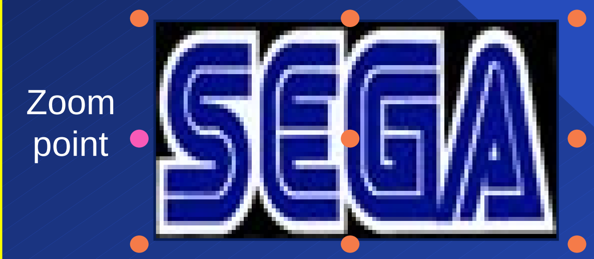
Part Types - Textured parts



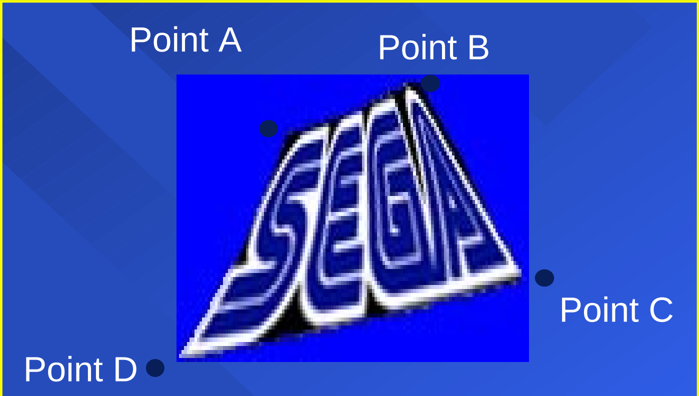
Normal sprite



Scaled sprite



Scaled sprite with zoom point



Distorted sprite

(Caveat - Half transparency processing)

Textures

- Arbitrary bitmaps that can be up to 504 pixels wide by 255 pixels tall.
- Width must be divisible by 8.
- Pixels in texture can be 4, 8, or 16 bits wide (see below).
- Textures can be unpaletted (RGB), paletted or “CLUT”ed.
- Different types of textures can be freely intermixed.

Colour mode			Mode Num	Description		Bits per pixel
Bit 5	Bit 4	Bit 3		Colours	Mode	
0	0	0	0	16	Palette (VDP2 CRAM)	4 bits
0	0	1	1	16	Lookup table mode	4 bits
0	1	0	2	64	Palette (VDP2 CRAM)	8 bits
0	1	1	3	128	Palette (VDP2 CRAM)	8 bits
1	0	0	4	256	Palette (VDP2 CRAM)	8 bits
1	0	1	5	32768	RGB mode	16 bits
Other than above				Setting prohibited (do not set)		

(See Colour Mode bits 5-3 - Com table Draw mode word @ offset +04H)

Textures - Tips/Cycles

- TIP : The number of cycles required to draw a normal textured polygon is as follows :

  $\text{cycles} = 70 + (x * y * 3) + (y * 5)$ x/y = Width/Height in pixels

- An 8 by 8 textured polygon can be drawn in 302 cycles (using above formula).
- TIP : Adding Gouraud shading increases the time to around 530 cycles. A provisional formula for Gouraud shaded sprites is as follows :

  $\text{cycles} = 302 + (x * y * 3) + (y * 5)$ x/y = Width/Height in pixels

Lookup table mode

- Defines 16 bit RGB colour values for 16 colour codes for paletted texture.
- Each display pixel Lookup table mode texture is defined using 4 bits of image data.
- Colour lookup tables are 20H bytes in size and must be located on a 20H byte boundry. In addition they must lie totally in VDP1 VRAM range 20h-7ffffh.

Textures - Unpaletted (RGB)

- Allows 32,768 simultaneous colours at 16 bits per pixel.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1	B					G					R				

- The pixels written to the frame buffer are the same as the pixels in the texture.
- Special effects (“colour calculation effects”) can be performed on RGB textured parts.

Textures - Paletted and CLUT

Paletted Textures (Can have 64, 128, or 256 colours at 8 bits per pixel).

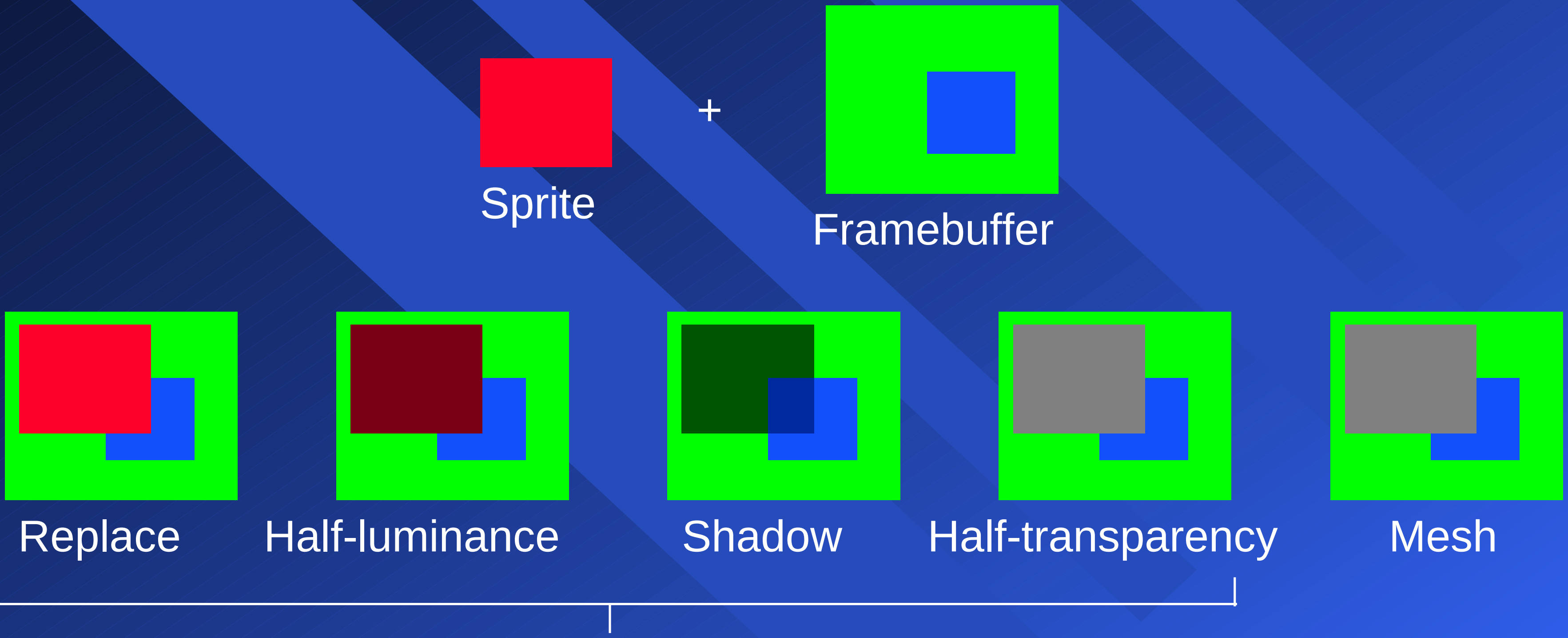
- Palettes are stored in VDP2 CRAM and shared between VDP1 and VDP2.
- Pixels written to frame buffer consist of index into colour RAM, plus other special function bits.
- Special effects eg : Gouraud shading, cannot be performed on paletted textures.
- These textures must be located on 20H byte boundry, and must lie totally in VDP1 VRAM range 20h-7ffffh

CLUT Textures

- Pixels are 4 bits wide.
- The 4-bit values index a lookup table stored in VRAM instead of a palette.
- The lookup table entries are copied directly to the frame buffer.

Colour Calculation Effects

- Performed by modifying the RGB values written to the frame buffer:



Sprite + Framebuffer

Replace

Half-luminance

Shadow

Half-transparency

Mesh

Colour calculation bits 2-0 in Draw mode word command table @ offset +04H

Msh - Bit 8 in Draw mode word Command table @ offset +04H

Colour Calculation Effects - Gouraud Shading

- One RGB code applied to each corner of the part. VDP1 interpolates RGB colour components across the surface (see Gouraud shading tables below).
- Tables are 8 bytes in size and must be located on an 8 byte boundry. In addition they must lie totally in VDP1 VRAM range 30008h-7ffffh.

Table address	Corresponding point	
	Sprites, polygons, polylines	Lines
Table top address	Point (A) - RGB code	Start point
Table top address+2	Point (B) - RGB code	End point
Table top address+4	Point (C) - RGB code	Ignored
Table top address+6	Point (D) - RGB code	Ignored

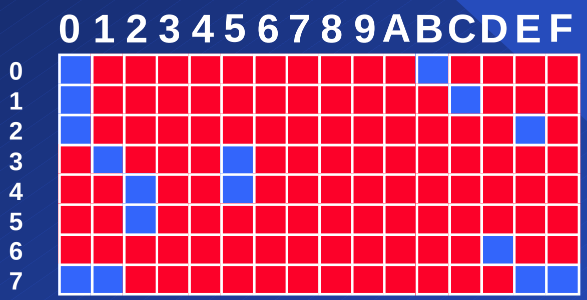
For 5-bit RGB col values : `0x00= -16, 0x10= 0, 0x1F= +15`

For example:

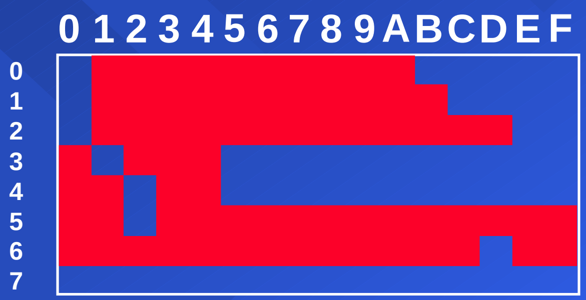
`0x0000` subtracts 16 from R, G and B
`0x4210` leaves R,G and B alone
`0x7FFF` adds 15 to R, G and B

End Codes

- End codes allow VDP1 to skip to the next line of a texture before reaching the end of the current line.
- Saves drawing time for textures that are largely transparent.
- VDP1 stops drawing when it encounters the second end code on a given line.
- When end codes are enabled, the highest possible pixel value is the end code.
- In 64-colour and 128-colour modes, the end code is 0xff, not 0x3f or 0x7f.
- End codes can be enabled or disabled on a sprite-by-sprite basis.



Original Character Pattern



Programming the VDP1

- The VDP1 draws sprites into the frame buffer by executing instructions in a display list.
- This process is asynchronous; the other processors are running at the same time.
- A VDP1 instruction is called a “command table”.
- There are three types of commands:
  - Co-ordinate setting commands
  - Drawing commands
  - The End command

- TIP : It takes VDP1 16 cycles to fetch a command table.
- TIP : It also takes 16 cycles to process a command table that will not be executed because its link specification causes it to be skipped (bit 14 command table control word is set).

Co-ordinate Setting Commands

- System Clipping
  - Sets lower-right-hand co-ordinates of the system clipping window.
  - Upper-left-hand co-ordinates are always (0,0).
  - System clipping window is always in effect.
- User Clipping
  - Each sprite specifies individually if user clipping applies or not.
  - Each sprite also specifies whether the area inside or outside the user window is clipped.
  - Sets upper-left-hand and lower-right-hand co-ordinates of the user clipping window.
- Local Co-ordinates
  - Specified Co-ordinates are added to Co-ordinates in each subsequent drawing command.
  - Temporarily relocates the origin.

Drawing Commands (& The End Command)

Drawing commands

- One for each part type: line, polyline, polygon, normal sprite, scaled sprite, scaled sprite with zoom point and distorted sprite.
- Each command specifies co-ordinates, pixel size, what palette to use, location and size of textures, etc.

The End command

- Tells the VDP1 to stop drawing.

Flow of Control

- When told to start plotting, VDP1 executes the command table found at the first address in VRAM.
- Each command table contains a link specification which tells VDP1 which command table to execute next.
- Options include:
  - Fall through to the next command table.
  - Jump to the command table at a given address.
  - Call a subroutine (only one level of nesting is supported).
  - Return from a subroutine.
- In addition, the link specification can cause the command table containing it to be skipped.

Anatomy of a Command Table 1

- Command tables are 30 bytes long and must be aligned on 32-byte boundaries:

32 Bytes

<table>
<tr><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>END</td><td colspan="3">JP (JUMP)</td><td colspan="4">ZP (ZOOM POINT)</td><td>-</td><td>-</td><td colspan="2">DIR V/H</td><td colspan="4">COMM</td></tr>
<tr><td colspan="14">LINK specification (used with JP jump/skip assign/call) / 8H</td><td>0</td><td>0</td></tr>
<tr><td>MON</td><td>-</td><td>-</td><td>HSS</td><td>Pclp</td><td>Clip</td><td>Cmod</td><td>Msh</td><td>ECD</td><td>SPD</td><td colspan="3">Colour Mode</td><td colspan="3">Colour Calc</td></tr>
<tr><td colspan="16">Colour bank, Colour lookup table /8H (LSB is set to 00), nontextured colour</td></tr>
<tr><td colspan="16">Character Address / 8H</td></tr>
<tr><td>-</td><td>-</td><td colspan="6">Character Size X/8</td><td colspan="8">Character Size Y</td></tr>
<tr><td colspan="5">Sign extension</td><td colspan="11">Parameter (A) XA</td></tr>
<tr><td colspan="5">Sign extension</td><td colspan="11">Parameter (A) YA</td></tr>
<tr><td colspan="5">Sign extension</td><td colspan="11">Parameter (B) XB</td></tr>
<tr><td colspan="5">Sign extension</td><td colspan="11">Parameter (B) YB</td></tr>
<tr><td colspan="5">Sign extension</td><td colspan="11">Parameter (C) XC</td></tr>
<tr><td colspan="5">Sign extension</td><td colspan="11">Parameter (C) YC</td></tr>
<tr><td colspan="5">Sign extension</td><td colspan="11">Parameter (D) XD</td></tr>
<tr><td colspan="5">Sign extension</td><td colspan="11">Parameter (D) YD</td></tr>
<tr><td colspan="16">Gouraud Shading Table / 8H</td></tr>
<tr><td colspan="16">(Dummy) Skipped during table fetch</td></tr>
</table>

Anatomy of a Command Table 2

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Mon			HSS	Pclp	Clip	Cmod	Msh	ECD	SPD	Colour mode			Colour calc		
-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	0

Draw Mode word
Command table
@ offset +04H

Mon - Specifies wether MSB (bit 15) is forced to 1 for each 16 bit pixel written to frame buffer to control shadowing (not RGB parts).

HSS - High speed shrink - Added to version 1 VDP1.

Pclp - Pre-clipping disable - Added to version 1 VDP1.

Clip/Cmod - Specifies wether or not part is drawn according to last user clipping coords and wether clipping is performed inside/outside zone

Msh - Specifies wether mesh processing enabled for Part draw command.

ECD - End code disable.

SPD - Transparency pixel enabled/disabled

Colour mode - Number of colours and colour usage for sprites

Colour calc - Specifies Gouraud shading, shadow, half-luminance and half-transparency.

Controlling the Frame Buffers

- There are two frame buffers. While one is written to, the other is displayed.
- When the VDP1’s program has completed, the frame buffers must be swapped.
- Frame swapping takes place either under One cycle or manual mode.
- The frame buffer being displayed must be erased before it can be redrawn.
- VDP1 can erase frame buff during display or in the vertical blanking interval.

VBE	FCM	FCT	Mode	Change time
0	0	0	One cycle mode	Every 1/60 second
0	0	1	Illegal setting	Illegal setting
0	1	0	Manual mode (erase)	Erase next field
0	1	1	Manual mode (change)	Change next field
1	0	0	Illegal setting	Illegal setting
1	0	1	Illegal setting	Illegal setting
1	1	0	Illegal setting	Illegal setting
1	1	1	Manual mode (erase/change)	Erase V blank

(VBE- TVMR register @ rel addr 0000H - Bit 3)

(FCM/FCT - FBCR register @ rel addr 0002H - Bits 1/0)

Frame display control modes

One Cycle Mode

- In this mode, the frame buffers are automatically swapped every frame (i.e. every 50th or 60th of a second).
- The display frame buffer is automatically erased as it is displayed.
- You have only one frame in which to draw sprites.

Manual Mode

- You can manually instruct the VDP1 to erase the frame buffer the *next* time it is displayed.
- Once erasing is completed, VDP1 will swap the frame buffers.
- This method requires you to know one frame in advance when you'll be finished with drawing.

Manual Mode with VB erase

- Usually, you won’t know ahead of time when you’ll be ready to swap the frame buffers.
- You can tell VDP1 to start erasing the frame buffer at the beginning of the next vertical blanking interval.
- VDP1 will swap frame buffers at end of this vertical blanking interval.
- Unfortunately, the vertical blanking interval might not be long enough to erase the entire frame buffer.
- Exactly how many lines are erased depends on the display mode.
- For example, in NTSC 320 x 240 mode, only 183 lines will be erased.
- You can erase the rest yourself be drawing a sprite at the beginning of your display list.

Draw triggering

- Regardless which frame control method you use the start of drawing is controlled by the Draw Trigger register (see table below).
- Note : Setting 00b is the default at switch on or after reset.
- Values set into this register must be compatible with Frame control settings (VBE- TVMR register @ rel addr 0000H - Bit 3) / (FCM/FCT - FBCR register @ rel addr 0002H - Bits 1/0).

PTM1	PTM0	Plotting mode
0	0	Idle at frame change
0	1	Starts plotting when 01b is written
1	0	Starts plotting automatically with frame change
1	1	Setting prohibited (don’t set)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
														PTM	
-	-	-	-	-	-	-	-	-	-	-	-	-	-	1	0

Draw Trigger register @ rel addr 0004H

Erase range control 1

Erase/Write data register EWDR @ rel addr 0006H

When Sprite display = 16 bits/pixel

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Erase/write data															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

When Sprite display = 8 bits/pixel

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
E/W data even X								E/W data odd X							
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Erase/write performed two pixels at a time when sprite display = 8 bits/pixel.

16/8 bits/pixel selected via bit 0 TVMR register @ rel addr 0000H

When bit 0=0 then sprite display 16 bits/pixel
When bit 0=1 then sprite display 8 bits/pixel

Erase range control 2

Erase/Write left/right coordinate registers EWLR/EWRR @ rel addrs 0008H/000AH

When Sprite display = 16 bits/pixel

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Upper left X							Upper left Y								
6	5	4	3	2	1	0	8	7	6	5	4	3	2	1	0

When Sprite display = 8 bits/pixel

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Lower right X							Lower right Y								
6	5	4	3	2	1	0	8	7	6	5	4	3	2	1	0

The actual X coordinate is :

@ 16 bits pixel
left coord X = register value x 8
right coord X = (register value x 8) - 1

@ 8 bits pixel
left coord X = register value x 16
right coord X = (register value x 16) - 1

The actual Y coordinate is expressed in line units (range 0-255). (Caveat : Double Interlace mode).

Erase will not work unless upper left X < lower right X AND upper left Y < lower right Y

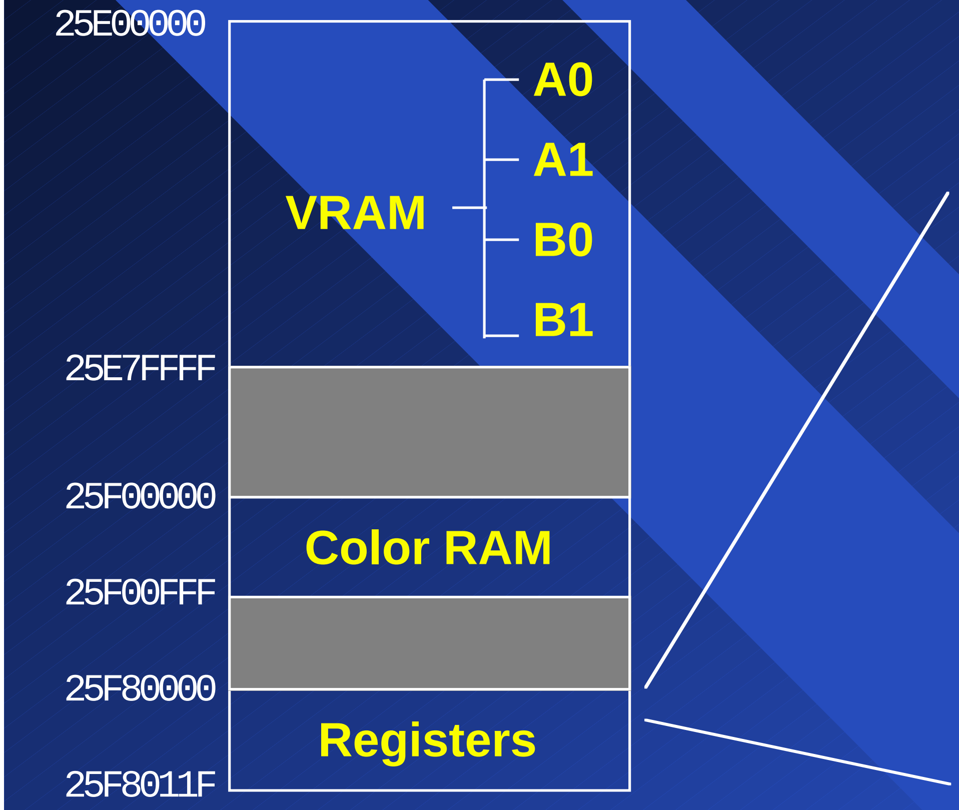
VDP2 – The Background Display Processor

Ed Hollingshead
Software Engineer
SoE Internal Product Development

Presentation aims - Section I

- Brief discussion of VDP2 address space. How it is organised. Introduce VRAM, CRAM, and control register set.

VDP2 - Address Map & Registers



Register num/name		VDP2 address/Status	
0	TVMD	25F80000	Read/Write
1	EXTEN	25F80002	Read/Write
2	TVSTAT	25F80004	Read-Only
3	VRSIZE	25F80006	Read/Write
4	HCNT	25F80008	Read-Only
5	VCNT	25F8000A	Read-Only
6	RESERVED1	25F8000C	DO NOT USE
7	RAMCTL	25F8000E	Read/Write
7F	RESERVED2	25F800FE	DO NOT USE

All other VDP2 registers are Write-Only.

VDP2 VRAM - General

- Holds tables and bit arrays required to produce VDP2 display screens as follows :

- Back screen (BACK).
- Line color screen (LNCL).
- Scroll screen NBG0.
- Scroll screen NBG1.
- Scroll screen NBG2.
- Scroll screen NBG3.
- Rotation screen RBG0.
- Rotation screen RBG1.

VDP2 CRAM - General

- Holds palettes defined for scroll screens and palette format sprites. Optionally when CRAM mode 1 is selected then the second half of the CRAM table can be used for coefficient table data.

RAMCTL register (16 bit @ rel addr 000EH)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data		Mode				VRAM-B/A		Rotation control							
n	-	n	n	-	-	n	n	n	n	n	n	n	n	n	n

Bit 15 CRAM coeff select
0 = CRAM NOT used for coeff data.
1 = CRAM used for coeff data.

(Bits 9/8 used for partition control)
(Bits 7/0 used for rotation control)

Bits 13/12 selects CRAM mode
(00) Mode 0 - RGB palette set. 15 bits. 1024 entries.
(01) Mode 1 - RGB palette set. 15 bits.
2048 entries if CRAM table NOT used for coeff data.
1024 entiries if CRAM table is used for coeff data.
(10) Mode 2 - RGB palette set. 24 bits. 1024 entiries.

Presentation aims - Section II

- Discuss basic VDP2 screen display types and VDP2 display relationship with VDP1 sprite screen.
- Discuss TV screen modes and display resolutions.

Display screens - Priority order

- Programmers can choose displayed order of scroll screens NBG0/1/2/3 and rotation screens RBG0/1.
- The back screen (BACK) and line color (LNCL) display screen order is NOT programmer selectable. These items are handled specially.
- Screen priority is derived from an integer value between 0 and 7. Lower priority screens are overlayed with higher priority screens.
- When two screens have same priority then display priority order is determined by a default scheme as follows (Sprite screen. RBG0. NBG0/RBG1. NBG1. NBG2. NBG3).

Display - Back screen (BACK)

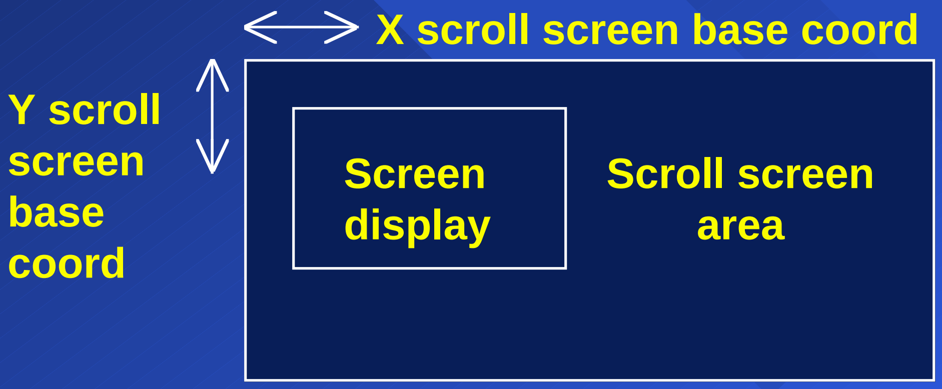
- The back screen is ALWAYS displayed overlapped by any other screen(s) which exist. It is “underneath” everything and is only displayed fully when other screens are not.
- The back screen can be either set to fill the whole display with a single color or, optionally, each line of the screen can be chosen to be a different color.
- The back screen CANNOT be switched off. It is always present.

Display - Line color screen (LNCL)

- The line color screen is not a true display screen.
- When the line color screen exists, AND when the current highest priority display screen permits a line color screen override, the line color screen is color-merged with the current highest priority screen.
- The line color screen can be either set to operate on the whole display with a single color or by means of a line color table.
- The line color screen can be used for special effects such as flashing display layer, or misting.

Display - Scroll screens I

- There are four scrolling screens NBG0, NBG1, NBG2 and NBG3. They have fixed sizes as follows :
  - **Bit mapped screens can be 512x256, 512x512, 1024x256, or 1024x512.**
  - **Character screens can be 1024x1024, 2048x1024, or 2048x2048.**
- **The image you see on-screen for any scrolling screen is a window onto the scroll area. It can be moved independently in any direction by selecting X/Y scroll screen base coords**

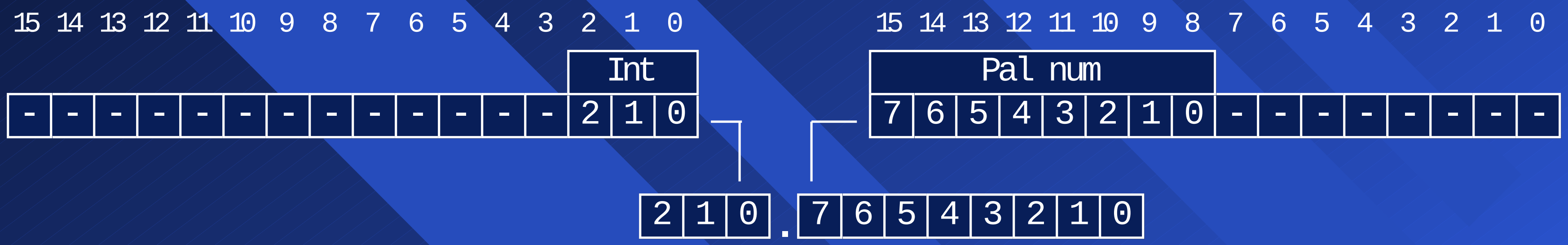


Display - Scroll screens II

- You can change the field of view for the scrolling screens NBG0/1 by setting X/Y coordinate increment values. This provides the appearance on the screen display that you are zooming in and out of the scrolling surface.
- You do not have the same facility for the scrolling screens NBG2/3. These scrolling screen windows always map on a pixel-by-pixel basis (1:1) onto the scrolling surface area.

Display - Scroll reduction I

NBG0 X coordinate increment registers ZMXIN0/ZMXDN0 @ rel addrs 0078H/007AH



Integer component

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
													Int		
-	-	-	-	-	-	-	-	-	-	-	-	-	2	1	0

Fractional component

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pal num															
7	6	5	4	3	2	1	0	-	-	-	-	-	-	-	-

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

When no scaling is to take place then these two registers are loaded with 1.0

Zoom-ins range from 1.0 (none) to 0.0 (max) and are unrestricted.

When zoom-outs take place (reduction) then the reduction enable register must used.

A similar set of registers exists for NBG0 Y coordinate increment control .

Display - Scroll reduction II

Reduction enable register ZMCTL 16 bit @ rel addr 0098H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						NBG1								NBG0	
-	-	-	-	-	-	1	0	-	-	-	-	-	-	1	0

NBG0 bits	Reduction	Coord incr range	Restrictions
00	n/a	0.0 <= range <= 1.0	None
01	1/2	0.0 <= range <= 2.0	Display 16/256 colors only, when 256 colors used NBG2 disabled
10/11	1/4	0.0 <= range <= 4.0	16 colors only, NBG2 disabled

NBG1 bits	Reduction	Coord incr range	Restrictions
00	n/a	0.0 <= range <= 1.0	None
01	1/2	0.0 <= range <= 2.0	Display 16/256 colors only, when 256 colors used NBG3 disabled
10/11	1/4	0.0 <= range <= 4.0	16 colors only, NBG3 disabled

Display - Scroll screens summary

- For NBG0/1 actual X/Y display coords are :

X display coord = (X coordinate increment * X scroll screen coord) + X scroll screen base coord

Y display coord = (Y coordinate increment * Y scroll screen coord) + Y scroll screen base coord

- **For NBG2/3 actual X/Y display coords are :**

X display coord = X scroll screen coord + X scroll screen base coord

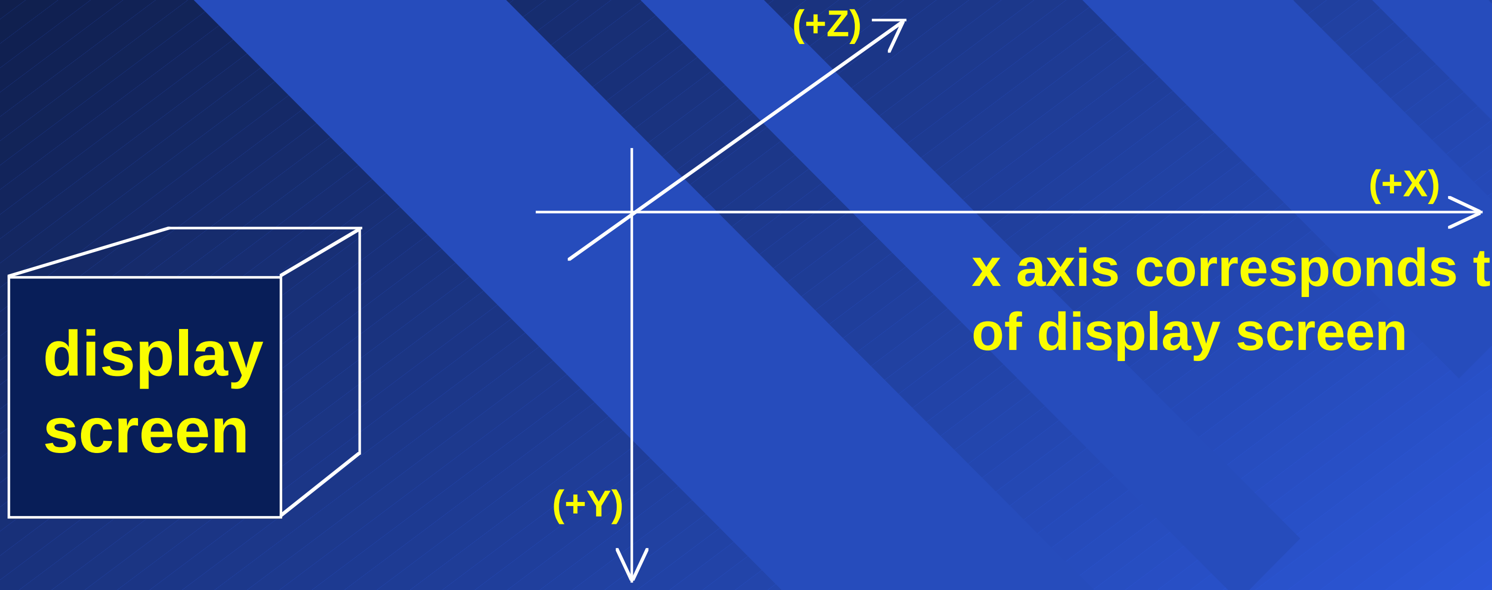
Y display coord = X scroll screen coord + X scroll screen base coord

Display - Rotation screens I

- There are two rotation screens: RBG0 and RBG1.
  - **Bit mapped screens may be 512x256 or 512x512.**
  - **Character screens may be 2048x2048, 4096x2048, or 4096x4096.**
  - **Each bit map, character set, or character map uses an entire VDP2 VRAM bank.**
  - **There are special caveats for using the second rotation screen (RBG1) as follows :**
  - **RBG1 cannot be used or defined unless RBG0 is defined.**
  - **Once RBG1 is defined then no other scrolling screens can be displayed.**

Display - Rotation screens II

z axis runs from
front of display
to back



x axis corresponds to top edge
of display screen

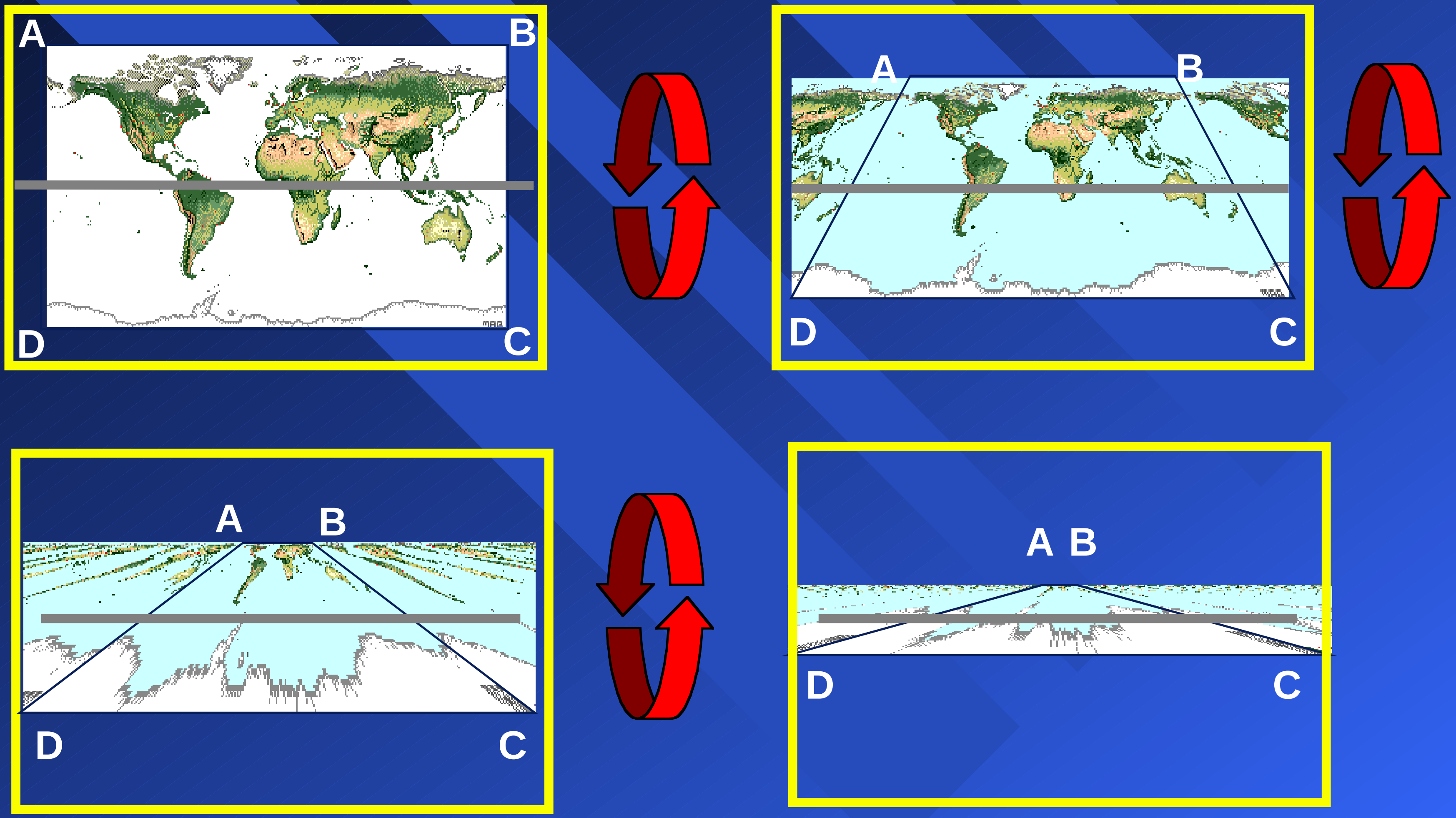
Y axis corresponds to left
edge of display screen

Display - Rotation Screens III

- Rotation screens provide the following facilities :

  - **Translation of the unrotated background in the XY plane.**
  - **Rotation of the translated background around the Z axis, or around any axis that is parallel to the Z axis.**
  - **Rotation around the X axis (or an axis parallel to it) or the Y axis (or an axis parallel to it), *but not both*.**
  - **Scaling and 2D projection of the rotated image onto the screen.**
  - **A final 2D rotation of the projected image around the Z axis.**

Display - Rotation Screens IV



Display - Sprite screen (VDP1)

- The sprite screen is regarded by VDP2 as an external screen.
- The sprite screen is supplied by VDP1.
- The sprite screen uses no VDP2 VRAM. It’s display is controlled by VDP2 strictly through the VDP2 control registers.
- Programmers can choose displayed order of the sprite screen through priority selection in the same way they can for screens NBG0/1/2/3 and RBG0/1.
- Various display resolutions can be generated by the Saturn and BOTH VDP1 and VDP2 have to have the screen display resolution explicitly set.
- The display resolution is normally set identically for both processors, but it need not be. See Saturn technical bulletin #22 for details.

Display - TV Screen Mode

- Whatever display resolution you select from the horizontal and vertical values available fall into one of three groups as follows :
  - **Normal mode.**
  - **High resolution mode.**
  - **Exclusive monitor mode (sometimes referred to as special monitor mode).**

Display - TV Screen Mode Register

TVMD register (16 bit @ rel addr 0000H)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Dsp							Brd	Intl			Vres		Hres		
n	-	-	-	-	-	-	n	n	n	-	n	n	n	n	n

Bit 15 (Disp) TV display

0 = Switch TV display off.
1 = Switch TV display on.

Bit 8 (Bord) - TV border

0 = Display as black.
1 = Use BACK screen col.

Bits 7/6 (Intl) Interlace

00 = Non-Interlace
01 = Not allowed
10 = Single Interlace
11 = Double Interlace

Bits 4/3 (Vres) Vert res

00 = 224 lines
01 = 240 lines
10 = 256 lines (PAL)
11 = Not allowed

Bits 2/0 (Hres) Horiz res

000 = 320 pixels - Normal mode
001 = 352 pixels - Normal mode
010 = 640 pixels - High res mode
011 = 704 pixels - High res mode
100 = 320 pixels - Exclusive mode
101 = 352 pixels - Exclusive mode
110 = 640 pixels - Exclusive mode
111 = 704 pixels - Exclusive mode

Display - Summary table

Number of Vertical Pixels	Number of Horizontal Pixels			
	320	352	640	704
224, 240, 256 (non-interlaced) 448, 480, 512 (interlaced)	Normal A	Normal B	Hi-res A	Hi-res B
480 (non-interlaced)	Special Normal A	Special Normal B	Special Hi-res A	Special Hi-res B

Notes : **Vertical resolutions of 256 and 512 pixels can be display on PAL systems only.**

The 480-line non-interlaced modes cannot be displayed on a standard television.

The “Special Hi-res” modes are subject to numerous restrictions.

Display - Special Hi-Res Mode

- NBG0 and NBG1 are combined to form a single display which must share the same display data (bitmaps or character patterns).
- CRAM mode 0 must be used.
- Reduction enable must be set at 1/2.
- NBG0/1 priority numbers must be set to be the same.
- The line color screen (LNCL) must not be used.
- Color calculation must not be used.
- The special priority function must not be used.
- NBG0/1 Y scroll base coords must always be identical.
- NBG1 X scroll screen base coord must be set to be equal to the NBG0 scroll screen base coord plus 1.
- NBG0/1 X coordinate increment values must be fixed at 2.
- Other NBG0 and NBG1 registers must be set identically.

Presentation aims - Section III

- Discuss screen internal data formats (Cell and bitmap format displays).
- Discuss global color offset manipulation.

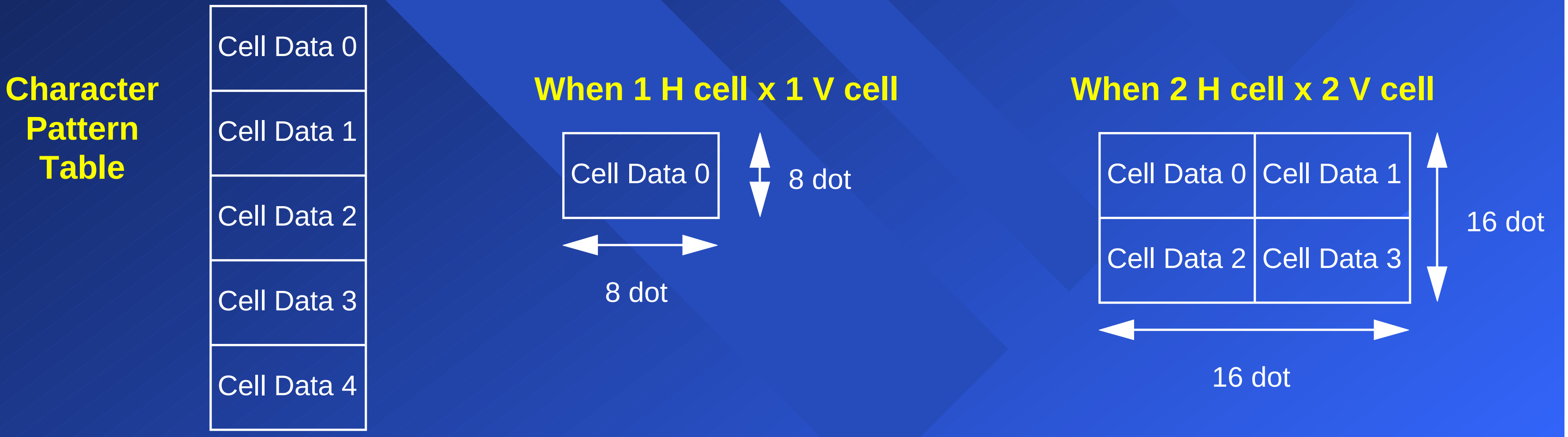
Internal data formats

- Programmers can choose to specify the data format of the scrolling screens (NBG0/1/2/3) and the rotation screens (RBG0/1) and select either:
  - **Cell format**
  - **Bit-map format**
  - **NBG2, NBG3 and RBG1 can only be dealt with as cell format screens.**

Cell format - Character cells

- A cell format screen is subdivided into 8 pixel by 8 pixel character cells which are held in a character pattern table.

- The character patterns in the character pattern table can consist optionally of just one single cell or a group of 2 by 2 cells.



Character Pattern Table

Cell Data 0

Cell Data 1

Cell Data 2

Cell Data 3

Cell Data 4

When 1 H cell x 1 V cell

Cell Data 0

8 dot

8 dot

When 2 H cell x 2 V cell

Cell Data 0	Cell Data 1
Cell Data 2	Cell Data 3

16 dot

16 dot

Cell format - Char pattern data

Colors	NBG0	NBG1	NBG2	NBG3	RBG0	RBG1	Format
16	N/H/E	N/H/E	N/H/E	N/H/E	N/H	N/H	Palette
256	N/H/E	N/H/E	N/H/E	N/H/E	N/H	N/H	Palette
2048	N/H/E	N/H/E	-	-	N/H	N/H	Palette
32768	N/H/E	N/H/E	-	-	N/H	N/H	RGB
16,770,000	N	-	-	-	N	N	RGB

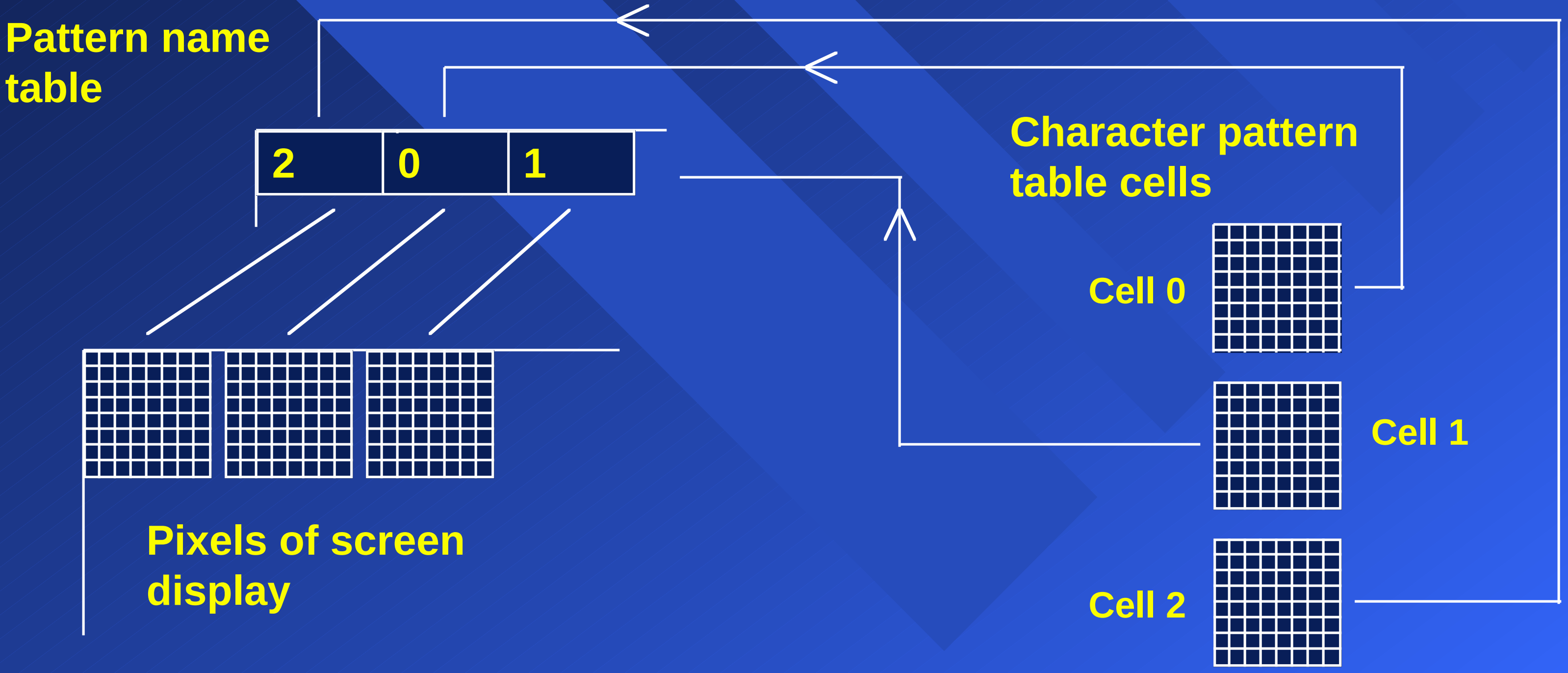
N/H/E = Normal mode / High-Res mode / Exclusive mode

Colors	Bytes/dot	Bytes/cell	Bytes/quad
16	1/2	32	128
256	1	64	256
2048	2	128	512
32768	2	128	512
16,770,000	4	256	1024

Cell format -Pattern name table I

- You choose which character pattern you display in any 8 pixel by 8 or 16 by 16 pixel location on screen by setting a word or long containing the appropriate character pattern number in a pattern name table.

(Assuming that character patterns consist of 16 color single cells)



Cell format -Pattern name table II

- If each character consists of a single cell, then each pattern name table (or page) is 64 by 64 characters in size.
- If each character consists of a 2-by-2 block of cells, then each page is 32 by 32 characters in size.
- In either case, each page is always 64 by 64 cells in size.
- Pattern name table data are either 16 bit words or 32 bit longs. When the pattern name table is set up as an array of 16 bit words then you can select two options for character pattern control as follows :
  - **Either use 10 bits of the 16 for character pattern numbers or**
  - **Use 12 bits of the 16 for character pattern numbers.**

Cell format -Pattern name data I

When Character patterns are represented by 1 16-color cell

Using 10 bits of the 16 for Character Pattern numbers :

Pattern name table data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pal num				Flip		Char pat num									
3	2	1	0	V	H	9	8	7	6	5	4	3	2	1	0

Pattern name control register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Pal num			Char pat num				
						PR	CC	6	5	4	14	13	12	11	10

Using 12 bits of the 16 for Character Pattern numbers :

Pattern name table data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pal num				Char pat num											
3	2	1	0	11	10	9	8	7	6	5	4	3	2	1	0

Pattern name control register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Pal num			Char pat num				
						PR	CC	6	5	4	14	13	12	▨	▨

Note: Shaded bits are ignored

Cell format -Pattern name data II

When Character patterns consist of 2-by-2 groups of 16-color cells

Using 10 bits of the 16 for Character Pattern numbers :

Pattern name table data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pal num				Flip		Char pat num									
3	2	1	0	V	H	11	10	9	8	7	6	5	4	3	2

Pattern name control register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Pal num			Char pat num				
						PR	CC	6	5	4	14	13	12	1	0

Using 12 bits of the 16 for Character Pattern numbers :

Pattern name table data

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pal num				Char pat num											
3	2	1	0	13	12	11	10	9	8	7	6	5	4	3	2

Pattern name control register

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Pal num			Char pat num				
						PR	CC	6	5	4	14	▨	▨	1	0

Note: Shaded bits are ignored

Cell format - Pattern name data III

- A Pattern name control register provides supplementary bits used by the pattern name table.
- There is a seperate Pattern name control register defined for EACH scrolling screen and rotation screen as follows :

Display screen name	Pattern control register name	VDP2 address
NBG0 (or RBG1)	PNCN0	25F80030
NBG1	PNCN1	25F80032
NBG2	PNCN2	25F80034
NBG3	PNCN3	25F80036
RBG0	PNCR	25F80038

Cell format -Pattern name data IV

- When the pattern name table is set up as an array of 32 bit longs then you do not use any pattern name control registers.

Pattern name table data

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
Flip									Pal num						
V	H	PR	CC	▨	▨	▨	▨	▨	6	5	4	3	2	1	0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Char pat num														
▨	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Note: Shaded bits are ignored

Cell format -Pattern name data V

A character pattern is represented by 1 cell

			Pattern name table data		
Colors	Bytes/dot	Bytes/cell	1st CP	2nd CP	3rd CP
16	1/2	32 (32*1)	0	1	2
256	1	64 (32*2)	0	2	4
2048	2	128 (32*4)	0	4	8
32768	2	128 (32*4)	0	4	8
16,770,000	4	256 (32*8)	0	8	16

Character pattern consist of groups of 2 by 2 cells

Colors	Bytes/dot	Bytes/quad	1st CP	2nd CP	3rd CP
16	1/2	128 (32*4)	0	4	8
256	1	256 (32*8)	0	8	16
2048	2	512 (32*16)	0	16	32
32768	2	512 (32*16)	0	16	32
16,770,000	4	1024 (32*32)	0	32	64

CP = Character pattern

Cell format - Hierachy I

- One or more pages determine layout of character cells for displayed screen.
- The hierachy of cells/character patterns and pages can be extended as follows :
  - **The pages can be further organised into planes.**
  - **A number of planes can be further organised into a map.**

Cell Format - Hierachy II

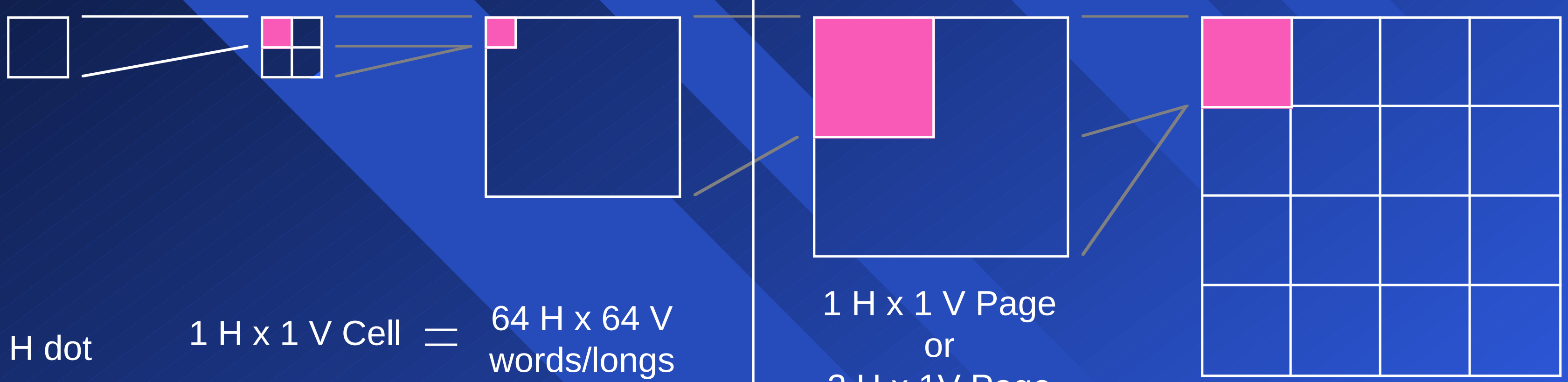
Cell

Character Pattern

Pattern name table (page)

Plane

Map



8 H dot
x
8 V dot

1 H x 1 V Cell = 64 H x 64 V words/longs

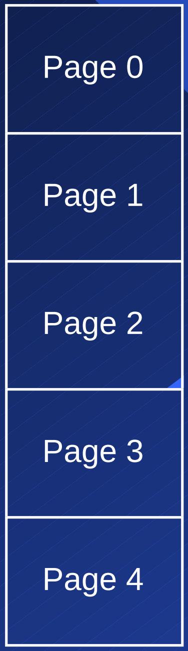
2H x 2V cell = 32 H x 32 V words/longs

1 H x 1 V Page
or
2 H x 1V Page
or
2 H x 2V Page

2 H x 2 V Plane
(normal screen)
or
4 H x 4 V Plane
(rotation scroll screen)

Cell format - Plane arrangement

Pattern Name Tables



Page 0

Page 1

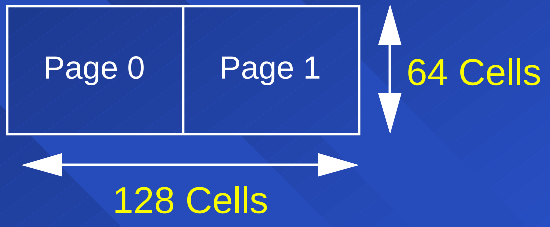
Page 2

Page 3

Page 4

Plane setups arrange the pattern name table (pages) in groups of 1x1, 2x1 or 2x2.

When 2 H pages x 1 V page

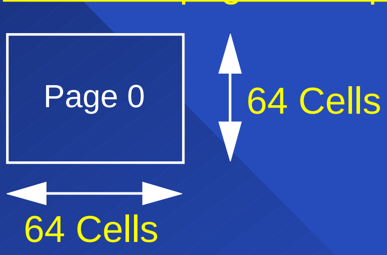


Page 0 | Page 1

64 Cells

128 Cells

When 1 H page x 1 V page

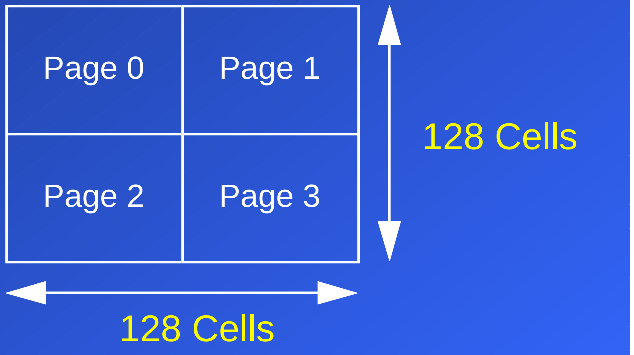


Page 0

64 Cells

64 Cells

When 2 H pages x 2 V pages



Page 0 | Page 1

Page 2 | Page 3

128 Cells

128 Cells

Cell format - Map arrangement

Plane A	Plane B
Plane C	Plane D

A map of a normal scroll surface consists of an arrangement of 2 by 2 planes.

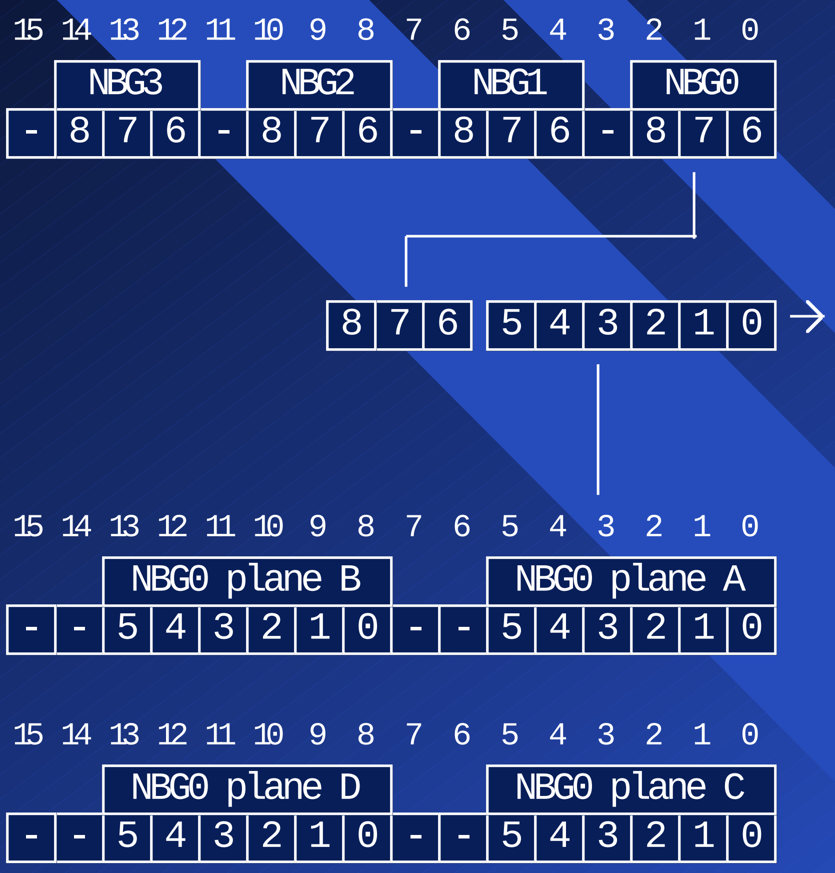
Plane A	Plane B	Plane C	Plane D
Plane E	Plane F	Plane G	Plane H
Plane I	Plane J	Plane K	Plane L
Plane M	Plane N	Plane O	Plane P

A map of a rotation scroll surface consists of an arrangement of 4 by 4 planes.

The start address of each plane within a map is selected by setting up a plane lead address for each plane within a map register.

Cell format - Map Registers I

For normal scroll screens NBG0/1/2/3



15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	NBG3				NBG2				NBG1				NBG0		
-	8	7	6	-	8	7	6	-	8	7	6	-	8	7	6

Map offset register (MPOFN)
(16 bit @ rel addr 003CH)

8	7	6	5	4	3	2	1	0

→ **In this case forms plane address for NBG0 plane A. All other plane addresses are formed in the same way for each normal screen using the same three lead bits.**

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		NBG0 plane B								NBG0 plane A					
-	-	5	4	3	2	1	0	-	-	5	4	3	2	1	0

Map register (MPABN0)
(16 bit @ rel addr 0040H)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		NBG0 plane D								NBG0 plane C					
-	-	5	4	3	2	1	0	-	-	5	4	3	2	1	0

Map register (MPCDN0)
(16 bit @ rel addr 0042H)

The map registers for NBG1/2 and 3 are arranged similarly

Cell format - Map Registers II

For rotation screens RBG0/1

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									RBG1				RBG0		
-	-	-	-	-	-	-	-	-	8	7	6	-	8	7	6

Map offset register (MPOFR)
(16 bit @ rel addr 003EH)

8	7	6	5	4	3	2	1	0

In this case forms plane address for RBG0 plane A. All other plane addresses are formed in the same way for each rotation screen using the same three lead bits.

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		RBG0 plane B								RBG0 plane A					
-	-	5	4	3	2	1	0	-	-	5	4	3	2	1	0

Map register (MPABRA)
(16 bit @ rel addr 0050H)

{ RBG0 planes cd,ef,gh,ij,kl,mn }

{ Map registers MP CD/EF/GH/IJ/KL/MN RA }

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		RBG0 plane O								RBG0 plane P					
-	-	5	4	3	2	1	0	-	-	5	4	3	2	1	0

Map register (MPOPRA)
(16 bit @ rel addr 005EH)

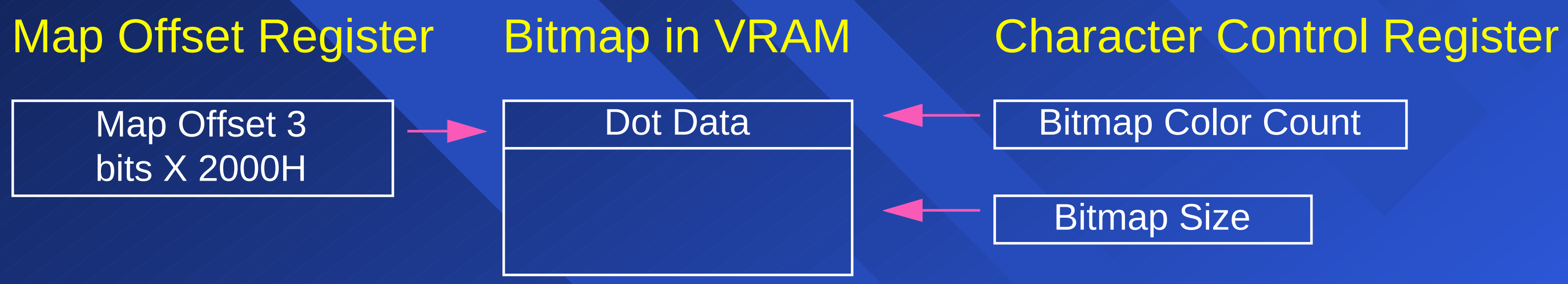
The map registers for RBG1 are arranged similarly

Cell format - Map Registers III

Plane size	Pattern name data size	Character size	Plane address
1H by 1V pages	1 word	1H by 1V cell	(Bits 6-0) X 2000H
		2H by 2V cells	(Bits 8-0) X 800H
	2 words	1H by 1V cell	(Bits 5-0) X 4000H
		2H by 2V cells	(Bits 7-0) X 1000H
2H by 1V pages	1 word	1H by 1V cell	(Bits 6-1) X 4000H
		2H by 2V cells	(Bits 8-1) X 1000H
	2 words	1H by 1V cell	(Bits 5-1) X 8000H
		2H by 2V cells	(Bits 7-1) X 2000H
2H by 2V pages	1 word	1H by 1V cell	(Bits 6-2) X 8000H
		2H by 2V cells	(Bits 8-2) X 2000H
	2 words	1H by 1V cell	(Bits 5-2) X 10000H
		2H by 2V cells	(Bits 7-2) X 4000H

Bitmap format - Data I

- VRAM start addresses for bitmaps are derived only from the map offset register for the appropriate VDP2 display screen.



Map Offset Register

Map Offset 3 bits X 2000H

Bitmap in VRAM

Dot Data

Character Control Register

Bitmap Color Count

Bitmap Size

Bitmap format - Data II

Character control registers CHCTLA/B (16 bit @ rel addrs 0028H/002AH)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		NBG1 clrs		NBG1 size		N1 Be			NBG0 clrs			NBG0 size		N0 Be	
		1	0	1	0	0			2	1	0	1	0	0	

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	RBG0 clrs				R0 sz	R0 Be									
	2	1	0		0	0									

Sizes

00 = 512H X 256V dots
01 = 512H X 512H dots
10 = 1024H X 256 dots
11 = 1024H X 512V dots

Bitmap enable

0 = Cell format
1 = Bitmap format

Colors

000 = 16 colors (PAL)
001 = 256 colors (PAL)
010 = 2048 colors (PAL)
011 = 32768 colors (RGB)
100 = 16,770,000 colors (RGB)
101 = Not allowed
110 = Not allowed
111 = Not allowed

Bitmap format - Data III

Colors	Bytes/dot	NBG0	NBG1	RBG0	Format
16	1/2	N/H/E	N/H/E	N/H	Pal
256	1	N/H/E	N/H/E	N/H	Pal
2048	2	N/H/E	N/H/E	N/H	Pal
32768	2	N/H/E	N/H/E	N/H	RGB
16,770,000	4	N	-	N	RGB

N/H/E = Normal mode / High-Res mode / Exclusive mode

Available sizes	NBG0	NBG1	RBG0
512 by 256 dots	Yes	Yes	Yes
512 by 512 dots	Yes	Yes	Yes
1024 by 256 dots	Yes	Yes	-
1024 by 512 dots	Yes	Yes	-

VDP2 - Color offset ctrl I

- Two sets of global color offset registers exist known as ‘Color offset A’ and ‘Color offset B’.
- The color offset control registers allow you to specify a signed red, blue and green, offset value which can be added arithmetically to all of the display colors for selected screens.
- Color offset control can be applied to ALL VDP2 display screens in any combination (except for the line color screen) including the BACK and sprite screen.
- When global palette control is applied to any screen, you can choose to place the screen in either the ‘A’ set or the ‘B’ set.

VDP2 - Color offset ctrl II

CLOFEN register (16 bit @ rel addr 0110H)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									Spr Enab	BACK Enab	RBG0 Enab	NBG3 Enab	NBG2 Enab	NBG1 Enab	NBG0 Enab
-	-	-	-	-	-	-	-	-	n	n	n	n	n	n	n

Bits 6/0 Color offset enable control

0 = Color offset control disabled

1 = color offset control enabled

CLOFSL register (16 bit @ rel addr 0112H)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
									Spr A/B	BACK A/B	RBG0 A/B	NBG3 A/B	NBG2 A/B	NBG1 A/B	NBG0 A/B
-	-	-	-	-	-	-	-	-	n	n	n	n	n	n	n

Bits 6/0 color offset control Group select control

0 = If color offset control enabled then place display into color offset group A.

1 = If color offset control enabled then place display into color offset group B.

Presentation aims - Section IV

- Discuss VRAM partitioning/rotation control.
- Discuss cycle pattern access control.

VDP2 VRAM - Partition control

RAMCTL register revisited (Bits 15,13/12 previously discussed)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Data		Mode				VRAM-B/A		Rotation control							
n	-	n	n	-	-	n	n	n	n	n	n	n	n	n	n

Bit 9 VRAM bank B partition control
0 = Do not partition bank
1 = Partition into two banks
Bit 8 VRAM bank A partition control
0 = Do not partition bank
1 = Partition into two banks

Bits 12, 13 and 15 control color RAM;
Bits 0-7 are used for rotation control.

VRAM partitions

VRAM-A	VRAM-A0	000000H	01FFFFH
	VRAM-A1	020000H	03FFFFH
VRAM-B	VRAM-B0	040000H	05FFFFH
	VRAM-B1	060000H	07FFFFH

(VRAM = 4Mbit)

VRAM - Cycle pattern access I

- The VDP2 computes pixels as it displays the screen.
- For each pixel on the screen, the VDP2 can perform up to 8 accesses to each VDP2 VRAM bank.
- In hi-res modes, only 4 accesses per pixel are possible.
- The nature of these accesses is user programmable through VDP2's 'cycle pattern access registers.' One set of these registers exists for each VRAM bank.

VRAM - Cycle pattern access II

- One cycle pattern access value needs to be set for each bank used, for each pattern name table access.
- One cycle pattern access value needs to be set for each bank used, for each 4 bits of pixel display data in character pattern or bitmap tables.
- When scroll reduction processing is in force then the number of cycle pattern accesses required have to be modified.

Colors	16			256		2048	32,768	16,770,000
Reduction setting	1	1/2	1/4	1	1/2	1	1	1
Accesses required during 1 cycle	1	2	4	2	4	4	4	8

VRAM - Cycle Pattern Registers I

CYCA0 @ rel addr 0010H

31							0
A0 - T0	A0 - T1	A0 - T2	A0 - T3	A0 - T4	A0 - T5	A0 - T6	A0 - T7

For VRAM A0
(or VRAM A)

CYCA1 @ rel addr 0014H

31							0
A1 - T0	A1 - T1	A1 - T2	A1 - T3	A1 - T4	A1 - T5	A1 - T6	A1 - T7

For VRAM A1

CYCB0 @ rel addr 0018H

31							0
B0 - T0	B0 - T1	B0 - T2	B0 - T3	B0 - T4	B0 - T5	B0 - T6	B0 - T7

For VRAM B0
(or VRAM B)

CYCB1 @ rel addr 001CH

31							0
B1 - T0	B1 - T1	B1 - T2	B1 - T3	B1 - T4	B1 - T5	B1 - T6	B1 - T7

For VRAM B1

(Range restricted to T0-T4 in High-res mode)

(The values T0 - T7 represent the time slots for the eight VDP2 VRAM cycle accesses available during the time it takes to display a single pixel).

VRAM - Cycle Pattern Registers II

	T0-T7 command values				VRAM access type
0	0	0	0	0	NBG0 pattern name table read
1	0	0	0	1	NBG1 pattern name table read
2	0	0	1	0	NBG2 pattern name table read
3	0	0	1	1	NBG3 pattern name table read
4	0	1	0	0	NBG0 character pattern (bitmap) read
5	0	1	0	1	NBG1 character pattern (bitmap) read
6	0	1	1	0	NBG2 character pattern (bitmap) read
7	0	1	1	1	NBG3 character pattern (bitmap) read
8	1	0	0	0	Setting not allowed
9	1	0	0	1	Setting not allowed
10	1	0	1	0	Setting not allowed
11	1	0	1	1	Setting not allowed
12	1	1	0	0	NBG0 Vertical cell scroll read
13	1	1	0	1	NBG1 Vertical cell scroll read
14	1	1	1	0	CPU read/write
15	1	1	1	1	No access

VRAM - Cycle Pattern Example

Assuming the following requirements :

Screen name	Char cols	Reduction	Pattern name	Char pattern
NBG0	256	X 1/2	A0	B0,B1
NBG1	256	X 1	A0,A1	B0,B1
NBG3	16	X 1	A1	A1,B0

Then the corresponding Cycle Pattern Register settings will be as follows :

T0	T1	T2	T3	T4	T5	T6	T7
0H	1H	0H	EH	EH	EH	EH	FH
3H	FH	1H	EH	EH	EH	EH	7H
4H	4H	5H	5H	EH	4H	4H	7H
4H	4H	5H	5H	EH	4H	4H	FH

NBG0 access 0H and 4H are doubled up to support reduction option

VRAM - Rotation control I

- For each component of a rotation screen defined an entire VRAM bank is occupied.
- If a VRAM bank is occupied by a rotation screen then the corresponding cycle pattern access registers are not applicable and should be set to no access.

VRAM - Rotation control II

RAMCTL register (Bits 15,13/12,9/8 previously discussed)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
								Rotation control							
Data		Mode				VRAM-B/A		VRAM-B1		VRAM-B0		VRAM-A1		VRAM-A0	
n	-	n	n	-	-	n	n	n	n	n	n	n	n	n	n

Bit 1/0 VRAM A0 (or VRAM A)
Bit 3/2 VRAM A1
Bit 5/4 VRAM B0 (or VRAM B)
Bit 7/6 VRAM B1

00 = Not used for RBG0 RAM
01 = RBG0 coefficient table
10 = RBG0 Pattern name table
11 = RBG0 Char pat table (or Bitmap)

When RBG1 exists then the following rules <u>must</u> be applied :

1) RBG1 pattern name table must be stored in VRAM-B1. RBG1 character pattern data must be stored in VRAM-B0.
2) RAMCTL bits 5/4 and 7/6 for banks B0 and B1 must be set to zeroes.

Presentation aims - Section V

- Discuss line and vertical cell scrolling.

VDP2 - Line scroll I

- This function is available only for NBG0/NBG1, and operates independantly of the NBG0/1 screen data format (cell or bitmap). An individual control table is set up for each of the two scrolling screens. .
- It provides the means to display any chosen line of what would be the displayed image for each vertical pixel position on-screen in any order.
- It also provides the means to start displaying any screen line of what would be the displayed image at any horizontal pixel position on screen.
- The above two options can be used either singly or together AND/OR each horizontal display line can be individually scaled by setting an X coordinate increment value.

VDP2 - Line scroll II

- When one or more of the line scroll options available are enabled they have to be placed in a defined order in the appropriate NBG0/1 line scroll control table as follows :

Line scroll table start address

Line 1 horiz screen scroll value	+00H
Line 1 vert screen scroll value	+04H
Line 1 horiz coord inc value	+08H
Line 2 horiz screen scroll value	+0CH
Line 2 vert screen scroll value	+10H
Line 2 horiz coord inc value	+14H

NBG0/NBG1 Line scroll table address register layout @ rel addrs 00A0H/00A4H

31															16
-	-	-	-	-	-	-	-	-	-	-	-	-	18	17	16

15															0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	-

VDP2 - Line scroll III

Horizontal, Vertical screen scroll table entry format

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
						Integer part										
+0H	-	-	-	-	-	n	n	n	n	n	n	n	n	n	n	n
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Fractional part															
+2H	n	n	n	n	n	n	n	n	-	-	-	-	-	-	-	-

Horizontal coordinate increment table entry format

	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
														Int part		
+0H	-	-	-	-	-	-	-	-	-	-	-	-	-	n	n	n
	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	Fractional part															
+2H	n	n	n	n	n	n	n	n	-	-	-	-	-	-	-	-

VDP2 - Line scroll IV

Line (vertical) cell scroll control register @ rel addr 009AH

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
		NBG1 control								NBG0 control					
		Line SI		HCI	Yscl	Xscl	VCSE			Line SI		HCI	Yscl	Xscl	VCSE
-	-	n	n	n	n	n	n	-	-	n	n	n	n	n	n

Bits 13/12, 5/4 Line (SI) Scroll interval

	Non-interlace	SD interlace	DD interlace
00 =	Each line	Every 2 lines	Each line
01 =	Every 2 lines	Every 4 lines	Every 2 lines
10 =	Every 4 lines	Every 8 lines	Every 4 lines
11 =	Every 8 lines	Every 16 lines	Every 8 lines

Bits 11,3 (HCI) Horizontal Coord Inc Enable

0 = Disabled

1 = Enabled

Bits 10,2 (Yscl) Y scroll enable

0 = Disabled

1 = Enabled

Bits 9,1 (Xscl) X scroll enable

0 = Disabled

1 = Enabled

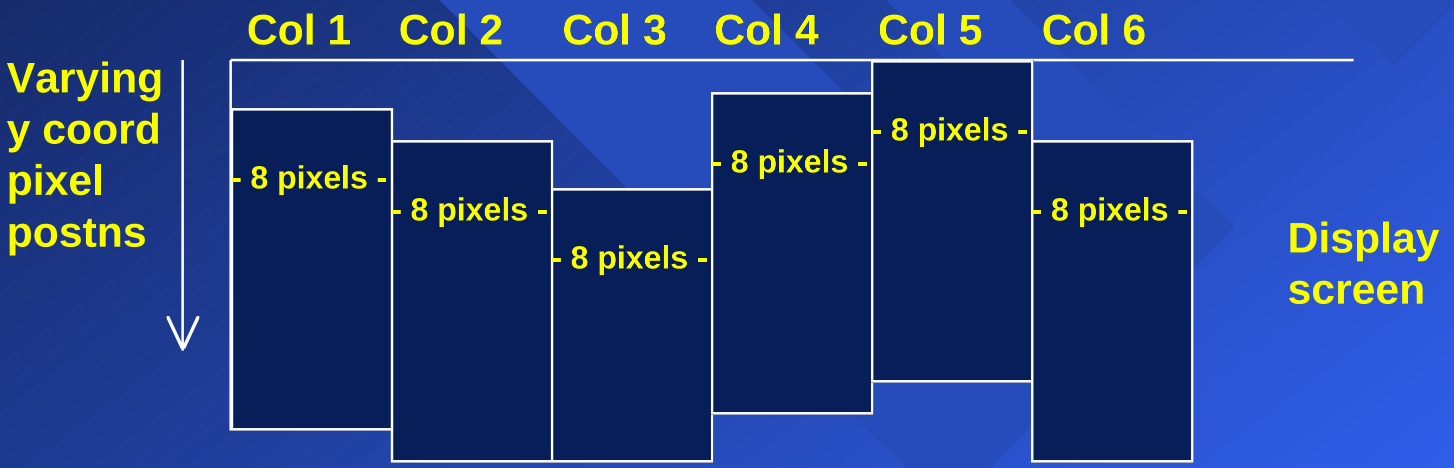
Bits 8,0 (VCSE) Vert Cell Scroll Enable

0 = Disabled

1 = Enabled

VDP2 - Vertical Cell scroll I

- This function is available only for NBG0/NBG1.
- This function is peculiar. What it does it split the screen horizontally into columns of 8 pixel wide cells, and then provides the means to display the columns at different starting Y pixel positions.



VDP2 - Vertical Cell scroll II

- Special settings must be placed in the cycle pattern access registers when this function is selected.
- Only ONE vertical cell scroll table register exists for this function. When both NBG0/1 use vertical cell scrolling then the vertical cell scroll table must contain interleaved NBG0/1 data.

Table entries when only NBG0 is specified

NBG0 Col 1 vert scroll val	+00H
NBG0 Col 2 vert scroll val	+04H
NBG0 Col 3 vert scroll val	+08H

Table entries when only NBG1 is specified

NBG1 Col 1 vert scroll val	+00H
NBG1 Col 2 vert scroll val	+04H
NBG1 Col 3 vert scroll val	+08H

Table entries when both NBG0/1 are specified

NBG0 Col 1 vert scroll val	+00H
NBG1 Col 1 vert scroll val	+04H
NBG0 Col 2 vert scroll val	+08H
NBG1 Col 2 vert scroll val	+0CH
NBG0 Col 3 vert scroll val	+10H
NBG1 Line 3 vert scroll val	+14H

VDP2 - Vertical Cell scroll III

Vertical cell scroll table address register @ rel addr 009CH

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
													Address		
-	-	-	-	-	-	-	-	-	-	-	-	-	18	17	16

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Address															
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	-

Vertical cell scroll table entries format

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
					Integer part										
-	-	-	-	-	n	n	n	n	n	n	n	n	n	n	n

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Fractional part															
n	n	n	n	n	n	n	n	-	-	-	-	-	-	-	-

VDP2 - Mosaic Control I

- Only horizontal mosaic processing is performed for the rotation screen.
- Mosaic control and vertical scrolling are mutually exclusive. Both cannot be done at the same time.

MZCTL register (16 bit @ rel addr 0022H)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Vertical mosaic size				Horizontal mosaic size							RBG0 Enab	NBG3 Enab	NBG2 Enab	NBG1 Enab	NBG0 Enab
3	2	1	0	3	2	1	0	-	-	-	n	n	n	n	n

Bits 15/12 Horizontal mosaic size

0000/1111 = 1 dot/16 dots

Bits 11/8 Vertical mosaic size

0000/1111 = 1 dot/16 dots (non-interlace)

0000/1111 = 2 dots/32 dots (interlace)

Bits 4/0 Mosaic enable control

0 = Mosaic disabled

1 = Mosaic enabled

Presentation aims - Section VI

- Discuss shadowing.

VDP2 - Shadowing I

- Two VDP2 “shadowing” support facilities exist to allow a VDP1 sprite to cast a shadow onto a VDP2 background:
  - **Normal shadow and**
  - **MSB shadow.**
  - **When a sprite shadow is cast then the sprite graphic is not displayed, instead the pixels which would have been obscured by the sprite are displayed at a reduced luminance.**

VDP2 - Shadowing II

- Before either Normal or MSB shadowing can be used then shadowing must be explicitly enabled for the screens onto which shadows are to be cast via the shadow control register as follows :

Shadow control register @ rel addr 00E2H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
							TPS DSL			Back Enab	RBG0 Enab	NBG3 Enab	NBG2 Enab	NBG1 Enab	NBG0 Enab
-	-	-	-	-	-	-	n	-	-	n	n	n	n	n	n

Bit 8 Transparent shadow select
0 = Disable transp shadow sprite
1 = Enable transp shadow sprite
(used by MSB shadowing ONLY)

Bits 5/0 Display enable control
0 = Shadowing disabled
1 = Shadowing enabled

VDP2 - Shadowing III

- The potential to display MSB shadows is created by VDP1 when the MSBON bit in the draw mode word of the VDP1 command table is set to 1. This in turn set the most significant bit of the appropriate pixels in the VDP1 frame buffer. The VDP1 sprite type must be one of types 2 through 7.
- To complete MSB sprite shadow setup then bits 4 and 5 of the Sprite control register (0x25f800E0) must be cleared. Note that this precludes the use of RGB sprites.
- When VDP2 encounters a VDP1 sprite pixel with the MSB set then the following can happen :
  - **If all other sprite pixel bits are zero (pix value = 8000H) then transparent shadow selection processing takes place.**
  - **Else VDP2 interprets the sprite pixel as opaque and the pixel is displayed with its luminance is reduced.**

VDP2 - Shadowing IV

- If transparent shadow selection processing is taking place then the following can happen :
  - **If the transparent shadow select bit of the Shadow select register is clear then MSB shadow are cast onto other sprites but NOT onto any of the backgrounds.**
  - **If the above bit is set then an MSB shadow is cast onto whatever background is immediately behind the transparent shadow pixel.**

VDP2 - Shadowing V

- The potential to display normal shadows is created by VDP1 when all of the sprite dot color bits are set except the least significant bit which is cleared.
- Shadowing on backdrops is then activated by VDP2 when the appropriate background shadow selection bits are set in the Shadow control register.

VDP2 - Shadowing VI

- **Normal sprites have the following advantages as opposed to MSB sprites:**
  - **They can be used in high resolution modes.**
  - **They can be mixed freely with RGB sprites.**
  - **Normal sprites have the following disadvantages as opposed to MSB sprites :**
  - **It is not possible to cast a normal shadow onto another sprite, so . . .**
  - **Any pixels of any sprite that happen to be underneath a normal shadow sprite are obliterated.**

Graphics Hardware

Extras (1)

SoE Internal Product Development

Presentation Aims

- Changing resolution from “A” group to “B” group (and back again).
- Use of normal windows.
- Transparency
- Further work

Changing resolutions from “A” group to “B” group

- 320 / 640 pixel horizontal resolutions are referred to as the “A” group resolutions.
- 352 / 704 pixel horizontal resolutions are referred to as the “B” group resolutions.
- Changing from one group to another involves the MANDATORY use of a reserved SEGA system function call. This is invoked through the SYS_CHGSYSCK() command (covered in the SMPC presentation).
- When a clock change mode is made the SMPC PLL oscillation frequency.changes.

  “A” resolutions - NTSC 26.8741MHz / PAL 26.6875 MHz.

  “B” resolutions - NTSC 28.6364 MHz / PAL 28.4375 MHz.
- This clock change mode affects ALL of VDP1, VDP2 and both SH2 processors.

Code example - Changing res “A” / “B” group

```
#define VDP2_REG_BASE              0x25F80000             /* Sprite IC Register base address  */
#define SPR_SCLREAD_REG(reg)       (*(volatile Uint16*)(VDP2_REG_BASE+(reg)))
#define SPR_R_TVSTAT               0x00000004             /* VDP2 TV status reg */
#define SPR_R_TVSTAT_VBL_BIT       8                      /* VBL status match mask */

            /* Make sure we are not in a VBLANK - Then wait until we know one has just commenced
             * The Tv status bit (3) = 0 during vertical scan / 1 during vertical re-trace */
            while ((SPR_SCLREAD_REG(SPR_R_TVSTAT) & SPR_R_TVSTAT_VBL_BIT) != 0);
            while ((SPR_SCLREAD_REG(SPR_R_TVSTAT) & SPR_R_TVSTAT_VBL_BIT) == 0);

            /* SCL_NORMAL_A = HRES 320 pixels / SCL_NORMAL_B = HRES 352 pixels */
            SCL_SetDisplayMode(SCL_NON_INTER,SCL_224LINE,SCL_NORMAL_B);
```

It is very important to ensure that Group mode changes are done just after the start of VBL.

Changing resolutions from “A” group to “B” group

- The major point about changing from an “A” group resolution such as 320 by 224 pixels to a “B” group resolution of 352 by 240 pixels is that ALL of VDP1, VDP2 and both SH2 processors run approx 6.5% faster.
- Aspect ratio differences from converting from “A” to “B” group can be soaked up to a fair extent by selecting best target resolution eg: 320x224 mode = aspect ratio 1.43. 352x240 mode = aspect ratio 1.47.
- The increased display area in a “B” group resolution screen can be soaked up to produce the appearance of an “A” group display area, by offsetting the base coordinates of any displayed images into the display area, and using a normal window. The only noticable visual difference in the 320/352 case is a slightly larger black border down the left and right hand sides.
- The choice made between “A” / “B” group resolutions is entirely application dependant.

Windowing I

- VDP2 supplies two normal Windows. W0 and W1.
- W0 and W1 can be set up either to function as simple rectangular windows or as line windows.
- The inside or outside of windows can be selected.
- Two active windows can be combined using AND or OR logic.
- Windows can be applied to any combination of VDP2 display screens (excepting the back and line color screens).

Windowing II

- The simple rectangular form of the windows W0 and W1 are setup for each display screen by :
  - Specifying the top left and bottom right screen X/Y coords of the windows.
  - Ensuring that for the selected window W0/W1 the line window format is off.
  - Explicitly enabling windowing.

- The line form of the windows W0 and W1 are setup when the line window format is explicitly switched on.
- A line window table is setup which specifies start and end X screen coordinates for every row of the screen display.
- (Windowing must also be explicitly enabled).

Windowing III

Window coordinate position registers

WPSX0 @ rel addr 00C0H

15															0
						W0 left x coord									
-	-	-	-	-	-	9	8	7	6	5	4	3	2	1	0

WPSY0 @ rel addr 00C2H

15															0
						W0 top y coord									
-	-	-	-	-	-	9	8	7	6	5	4	3	2	1	0

WPEX0 @ rel addr 00C4H

15															0
						W0 right x coord									
-	-	-	-	-	-	9	8	7	6	5	4	3	2	1	0

WPEY0 @ rel addr 00C6H

15															0
						W0 bottom y coord									
-	-	-	-	-	-	9	8	7	6	5	4	3	2	1	0

(A similar set of coordinate registers exists for Normal window W1 @ Rel addrs 00C8H to 00CEH).

Windowing IV

Window line table (and control) address registers and data format

Non-interlace / Double-density interlace
Line window table start address

1st line horizontal start x	+00H
1st line horizontal end x	+02H
2nd line horizontal start x	+04H
2nd line horizontal end x	+06H

Single density interlace
Line window table start address

1st & 2nd line horizontal start x	+00H
1st & 2nd line horizontal end x	+02H
3rd & 4th line horizontal start x	+04H
3rd & 4th line horizontal end x	+06H

(Each of the start/end x values for either of the above table formats are 9 bit values)

W0 Line window table address register layout @ rel addrs 00D8H/00DAH

31															16
E	-	-	-	-	-	-	-	-	-	-	-	-	18	17	16

15															0
15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	-

E - Line window control bit (0 = Line window disabled, 1 = Line window enabled)

(A similar pair of registers exists for W1 @ rel addrs 00DCH and 00DEH)

Windowing IV

Window logic / select control registers

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NBG1								NBG0							
Log				W1_E	W1_A	W0_E	W0_A	Log				W1_E	W1_A	W0_E	W0_A
n	-	-	-	0	0	0	0	n	-	-	-	0	0	0	0

NBG1/0 control reg @ rel addr 00D0H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
NBG3								NBG2							
Log				W1_E	W1_A	W0_E	W0_A	Log				W1_E	W1_A	W0_E	W0_A
n	-	-	-	0	0	0	0	n	-	-	-	0	0	0	0

NBG3/2 control reg @ rel addr 00D0H

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Sprite								RBG0							
Log				W1_E	W1_A	W0_E	W0_A	Log				W1_E	W1_A	W0_E	W0_A
n	-	-	-	0	0	0	0	n	-	-	-	0	0	0	0

Sprite/Rot control reg @ rel addr 00D0H

Log
0 = Overlaid logic OR
1 = Overlaid logic AND
(within each screen)

W(n)_E
0 = Window disabled
1 = Window enabled

W(n)_A
0 = Window inside area
1 = Window outside area

Transparency of VDP2 Pixels

- Pallette format dots are transparent when their color code is zero.
- RGB format dots are transparent when the most significant bit of the RGB code is zero.
- Programmers can direct VDP2 to recognise transparent dots OR use them as normal color codes for each of the screens NBG0/1/2/3 and RBG0/1.

BGON register (16 bit @ rel addr 0020H)

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
			RBG0 Trns	NBG3 Trns	NBG2 Trns	NBG1 Trns	NBG0 Trns			RBG1 Enab	RBG0 Enab	NBG3 Enab	NBG2 Enab	NBG1 Enab	NBG0 Enab
-	-	-	n	n	n	n	n	-	-	n	n	n	n	n	n

Bits 12/8 Transparency control

0 = Transp. code dots ARE transparent

1 = Transp. code dots displayed as color code values

Bits 5/0 Display enable control

0 = Display disabled

1 = Display enabled

Further Work

- Rotation screens
  - Parameter tables
  - Coefficient tables
  - Pattern over process registers
- Shadowing.
- Gradation.
- Cycle pattern access registers - Comprehensive rules.
- Color calculation (and extended color calculation).

Music
&
Sound

Saturn Custom Sound Processor

(SCSP)

Overview

Richard Jacques

Musician

Hardware Requirements

- Macintosh IIFX or better
  - 16 MB RAM
  - 300 MB Fast Hard Disk
- MIDI Interface
- MIDI Controller

- Saturn Sound Box
  - SCSI Cable
  - InLine SCSI terminator

Software Requirements

- Tone Development tools
  - Waveform editor[1]
  - Tone Editor
  - DSP linker
- Composition Tools
  - MIDI sequencer[2]
- Sound development support system
  - Map tool/Sound Simulator

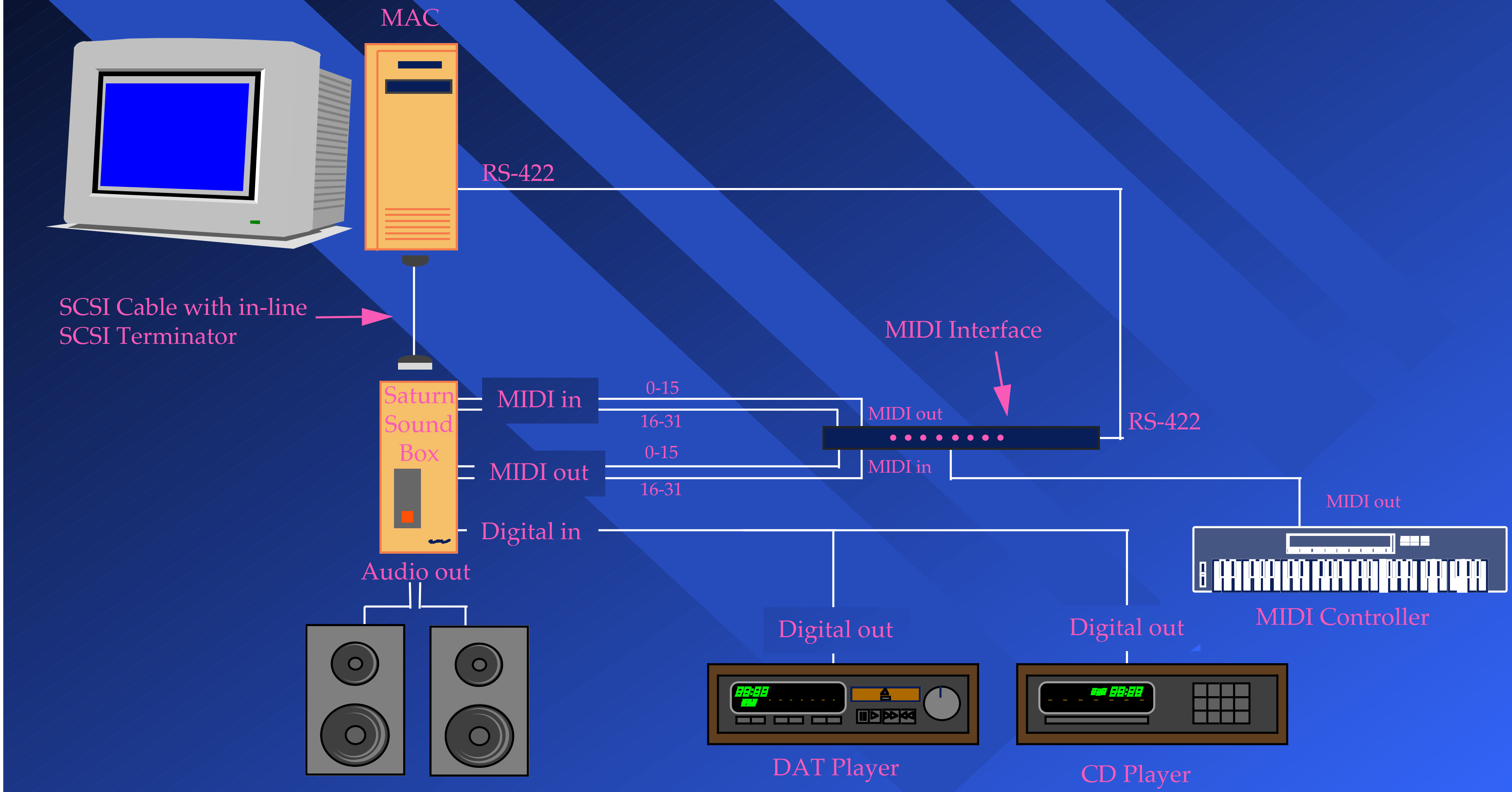
[1] Third party tools can be used instead

[2] Must use third party tools that are able to create a MIDI format 1 file

Functional Overview

- Two CPU interfaces; 68000/data transfer, can be operated in parallel
- Built-in MIDI interface (2 x 16 channels)
- 128-step DSP
- Built-in ring filter buffer for filtering
- Memory control functions allow program in sound memory to operate the CPU
- 16 channel Digital Mixer
- DMA transfer between SCSP Control Registers + Sound Memory (3812 bytes)

Sound Development System

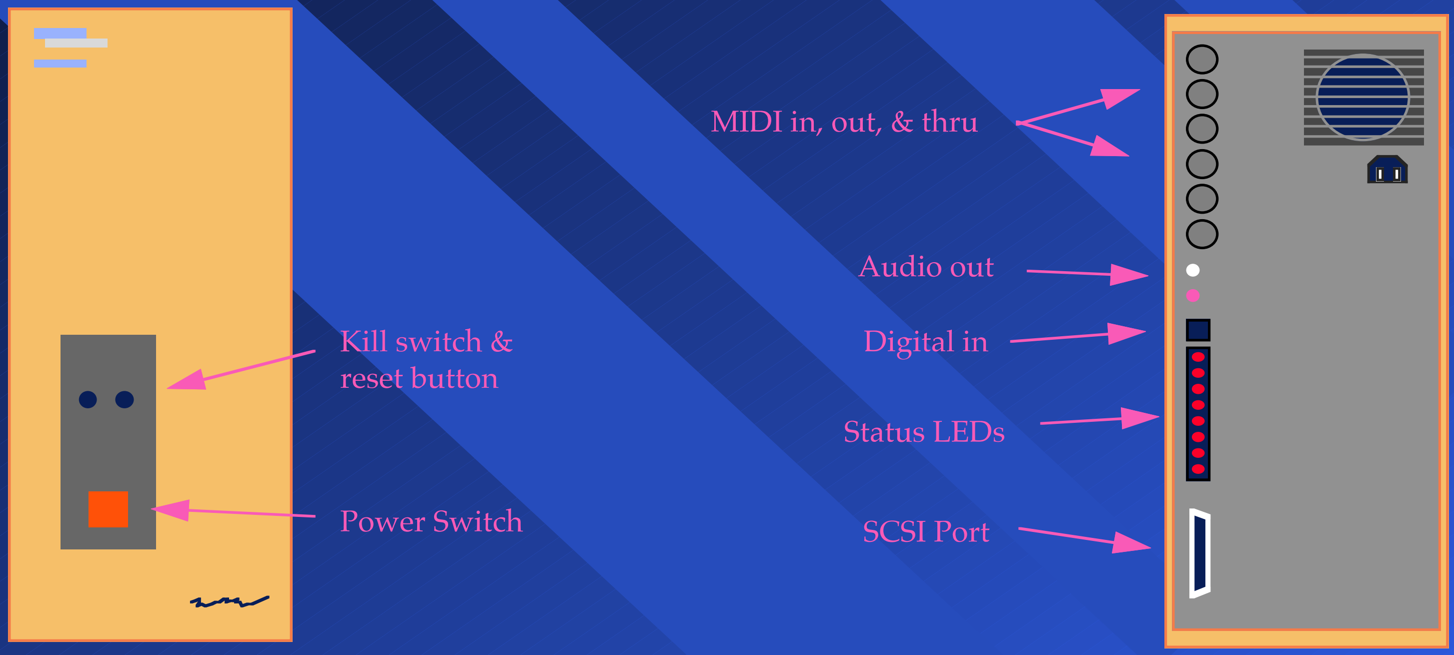


Sega

Workshop 95

Sega Confidential

Saturn Sound Box



MIDI in, out, & thru

Audio out

Digital in

Status LEDs

SCSI Port

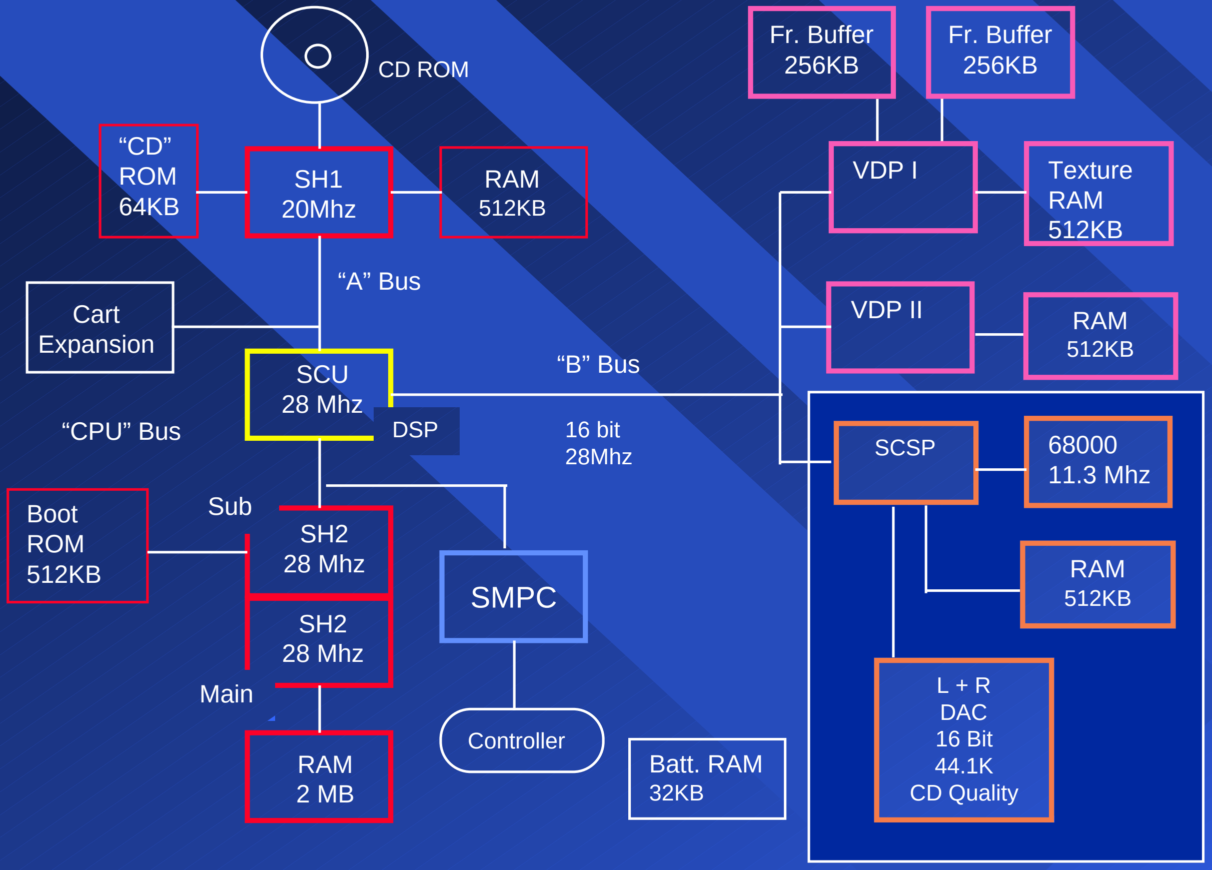
Kill switch & reset button

Power Switch

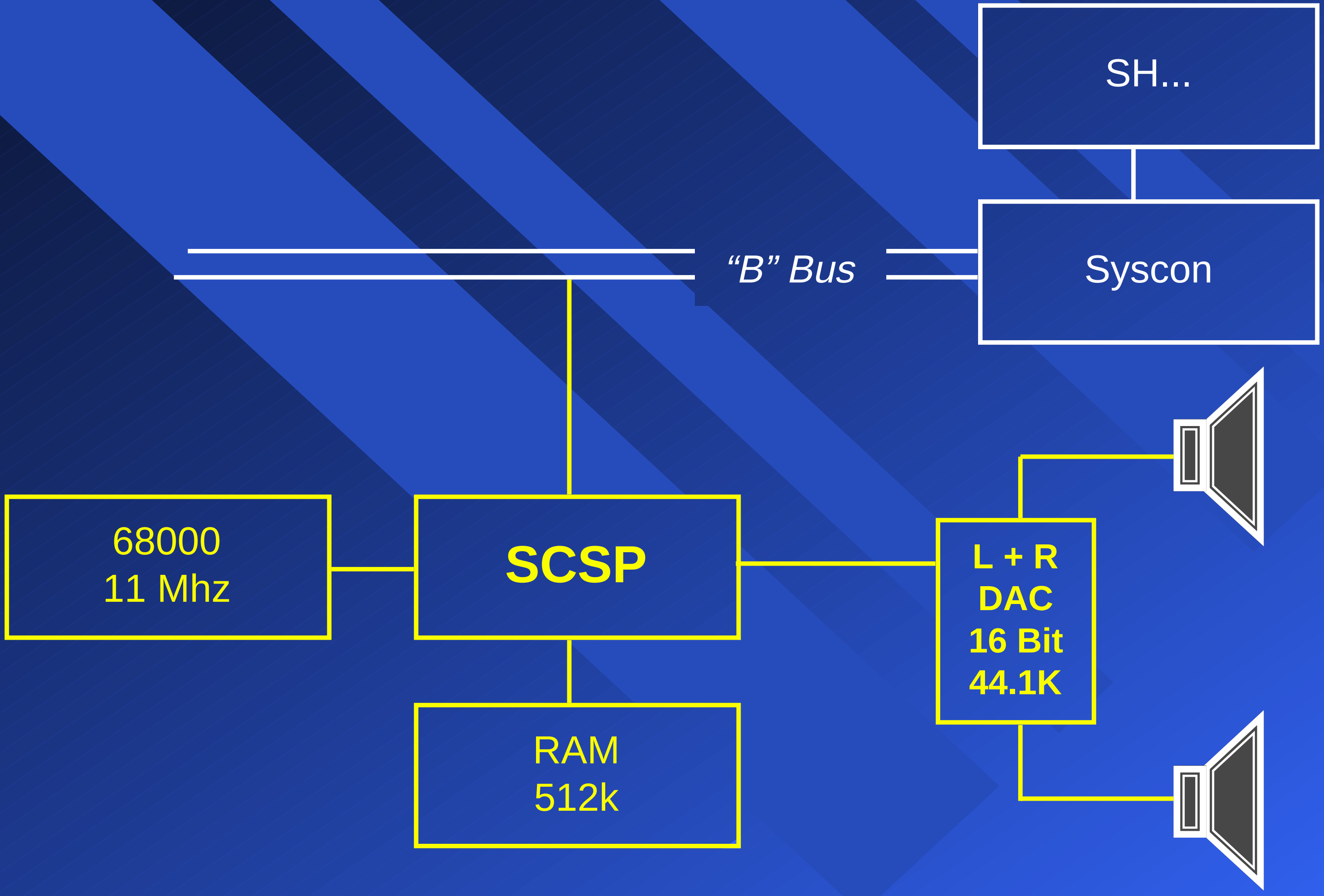
Front Side

Back Side

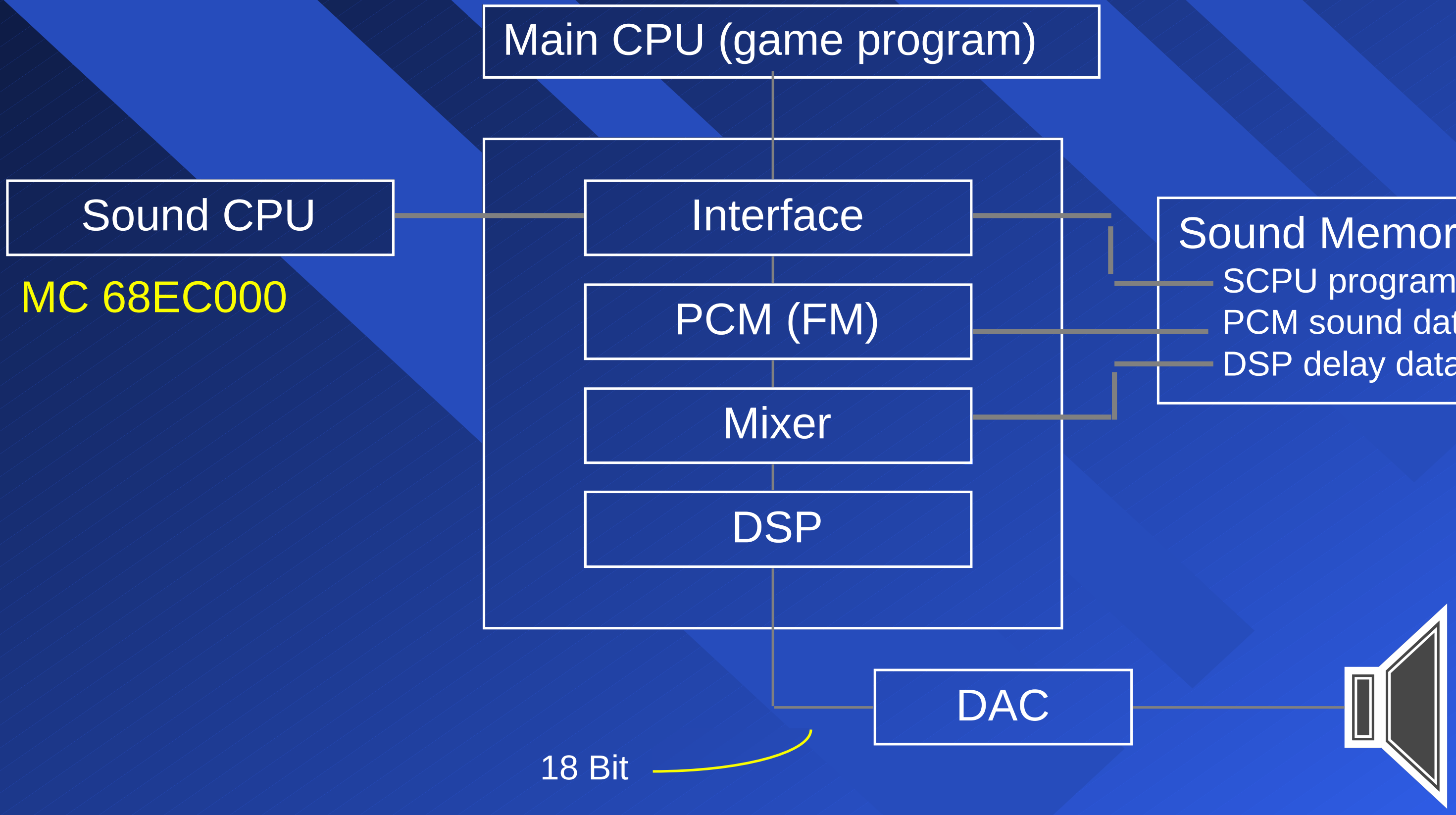
Where the SCSP sits



On the “B” Bus



Sound System Structure



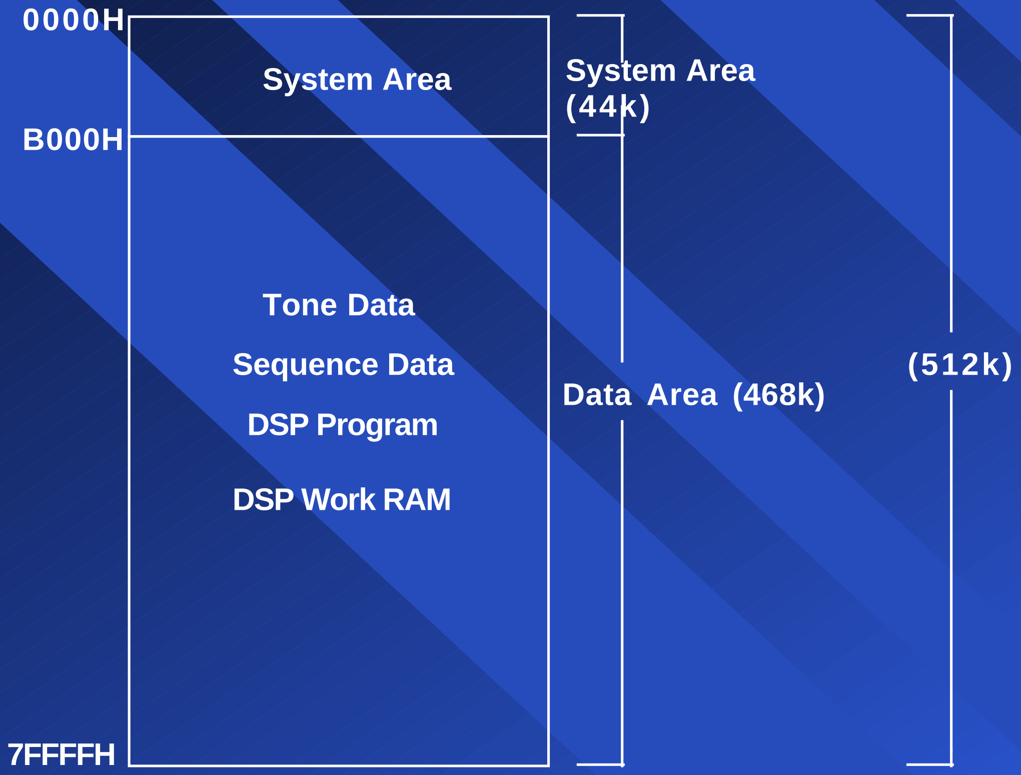
68EC000

- SCSP Controller
  - Direct Connection to SCSP
- 16-bit
- 11.3 MHz (1/2 speed of SCSP)

Microprocessor

- Yamaha FH1 Processor
- 22.58 MHz Overall Frequency
- 44.1 KHz Sampling Frequency
- Built-in DMA

Sound Memory



0000H

System Area

B000H

Tone Data

Sequence Data

DSP Program

DSP Work RAM

7FFFFH

System Area (44k)

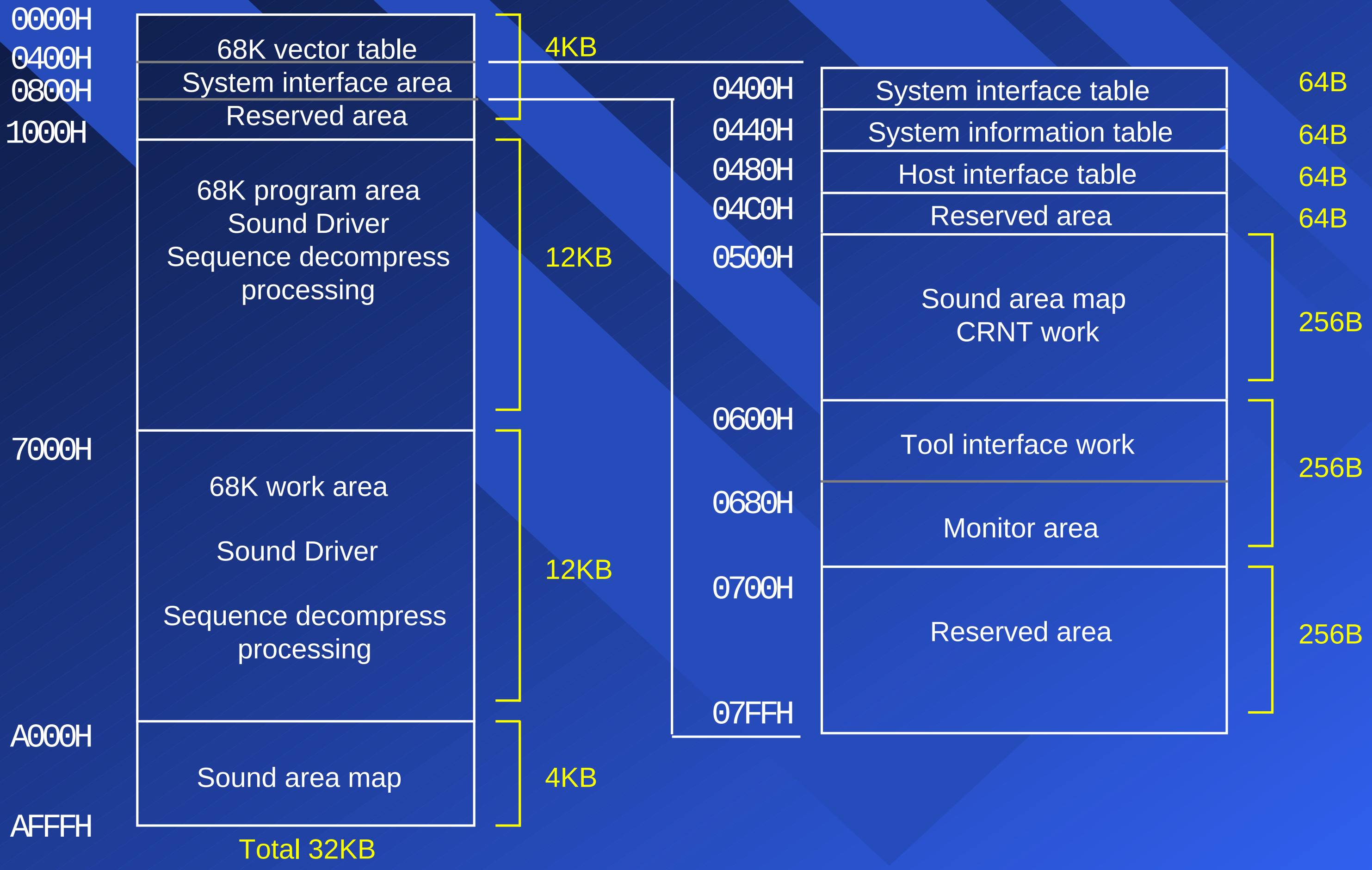
Data Area (468k)

(512k)

1. DSP Work RAM must go on an even boundary.

2. It is advisable to put the DSP work RAM at COOO
   (the lowest address) otherwise RAM area may be lost.

System Area



Address	Area	Size
0000H	68K vector table	4KB
0400H	System interface area	
0800H	Reserved area	
1000H	68K program area Sound Driver Sequence decompress processing	12KB
7000H	68K work area Sound Driver Sequence decompress processing	12KB
A000H	Sound area map	4KB
AFFFH		

Total 32KB

Address	Area	Size
0400H	System interface table	64B
0440H	System information table	64B
0480H	Host interface table	64B
04C0H	Reserved area	64B
0500H	Sound area map CRNT work	256B
0600H	Tool interface work	256B
0680H	Monitor area	
0700H	Reserved area	256B
07FFH		

System Area

- 68K Vector Table
  - The vector table for program interrupt processing by the sound CPU (68000). The size is fixed at 400H and cannot be changed.
- System Interface Area
  - A fixed area for interfacing between sound driver, tone development systems, and the host system for game assembly.
- 68K program
  - This is the program area for the sound CPU and is used to store and execute all programs related to sound.
- 68K work area
  - This is the program work area for the sound CPU and is used as a work area by all sound related programs.

System Area (cont.)

- Sound Area Map
  - Up to 128 area maps can be held in one sound area map (one area map can hold up to 32 pieces of map data
  - Using the Sound Simulator, one sound area map can be made for one game.
  - Since this area is only for storing the entire sound area map, the map data of the currently selected area references sound area map CRNT work of the system interface area.
  - The top address and area size are stored in the system information table in the system interface area.

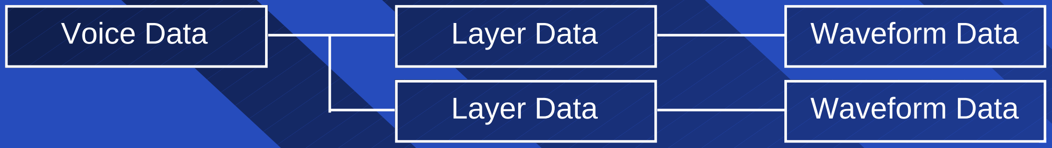
Host Interface Commands

- Song mode
  - Stop
  - Play
  - Fade in
  - Fade out
  - Pause Play
  - Fade in Pause
  - Fade out Pause
- Status mode
  - Normal
  - Error code
  - Timing flag

Sound Generator

- 32 dual-purpose slots for FM and PCM
- PCM data formats are 16-bit and 8-bit linear
- 32 Low Frequency Oscillators (LFO) that are completely independent for each slot
- 32 four-segment envelope generators (EG)
- A variety of loop controls can be implemented

Voice Architecture



Voice Data

Layer Data

Layer Data

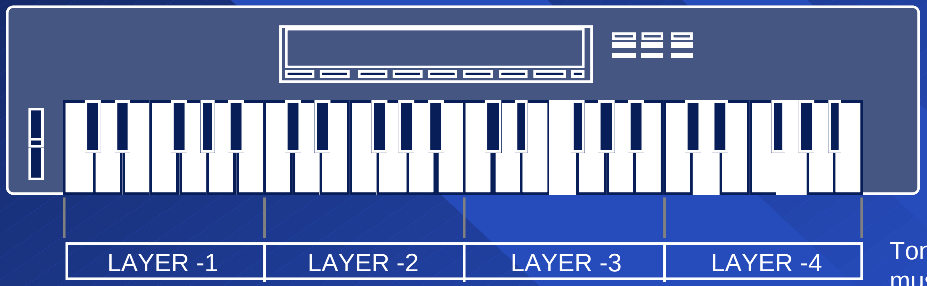
Waveform Data

Waveform Data

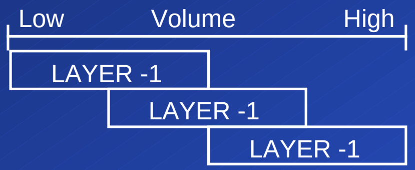
Voice Data

Layer Data

Waveform Data



Tone is changed by
musical interval (key split)



Tone is changed by volume
of sound (velocity switch)

A Voice Consists of...

- VoiceName
- Bend RangeWidth H - BendRangeWidth L
- Play Mode
- Number of Layers - LayerNumber
- Portamento
- VolBias

A Layer Consists of....

- Layer Name
- StartNote and EndNote
- Direct level
- Effect Send Level
- Effect select
- Direct Pan
- SLOT

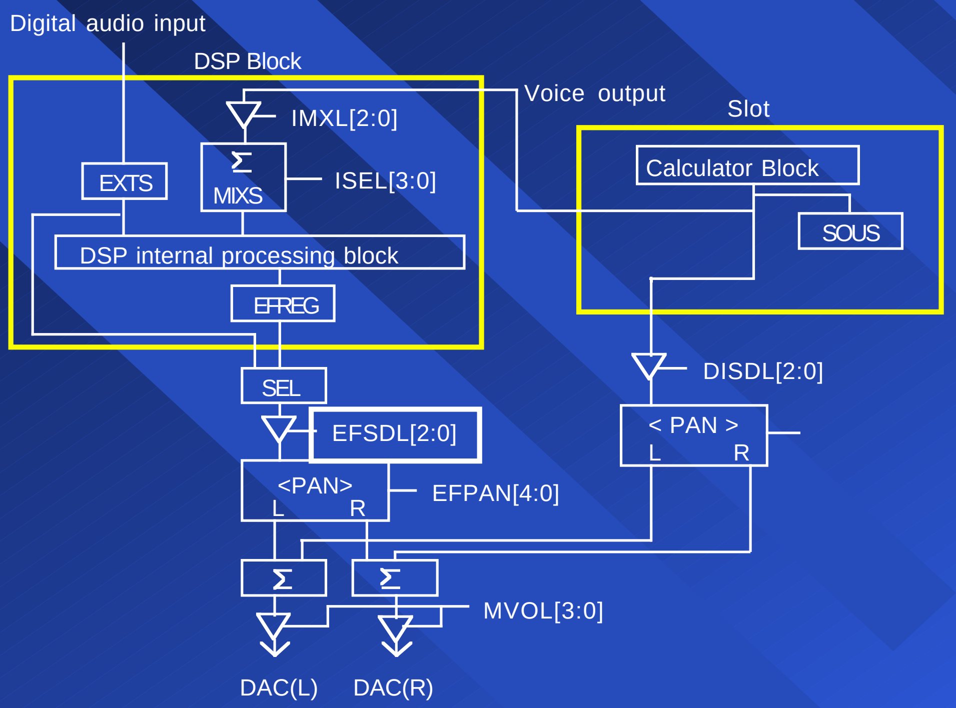
A Slot Consists of...

- PEG
- PLFO
- AIFF Sample Format
- ALFO
- EG
- Total Level
- Module set-up for FM

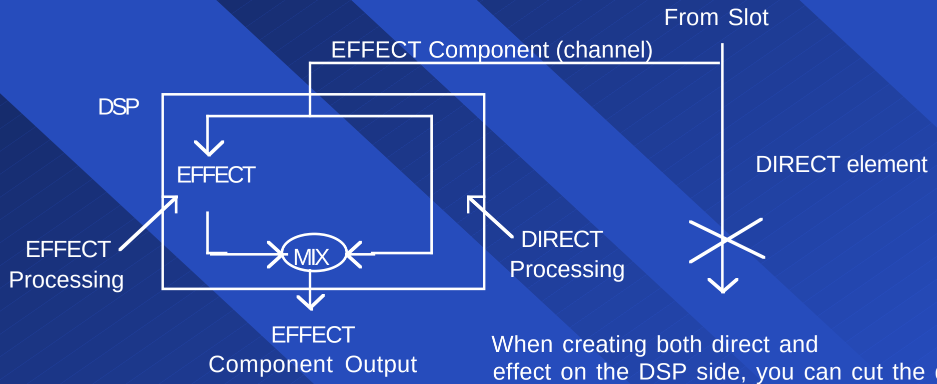
Mixer

- 16 channel digital mixer
- Flexible effects and bypass processing
- Routes through DSP modules

Mixer Block Diagram



Direct and Effect Components



When creating both direct and effect on the DSP side, you can cut the direct mixing circuit by setting DISDL=0.

DSP Modules

- Effects
  - Reverb
  - Early Reflection
  - Echo (Delay)
  - Pitch Shifter
  - Chorus
  - Flanger
  - Symphonic
  - Q Sound
  - Surround
  - Voice Cancel
  - Auto Pan
  - Phaser
  - Distortion
  - Filter
  - Parametric EQ
  - Yamaha 3D

SCSP Overview
Q&A

Sound & Music

Saturn Sound Tools

Sega

Workshop 95

Sega Confidential

SCSP Tools

- Sound Simulator
  - MAP Editor
- Tone Editor
- Waveform Editor
  - or optional 3rd party editor (e.g. Sound Designer)
- Linker
  - DSP Editor
  - SCSP (68000) Assembler
  - SSBug
- Third Party Sequence Editor
  - MIDI File Format 1

Hardware Requirements

- Macintosh IIFX or better
  - System 7.0 or better
  - 16 MB RAM
  - 300 MB Fast Hard Disk
- MIDI interface
- MIDI Controller
- Saturn Sound Box
  - SCSI Cable
  - InLine SCSI terminator

Sound Development Procedure

1. Start up sound board with Sound Simulator
2. Create Wave or AIFF sample data
3. Create Tone Data using the following editors:
   Waveform, Tone, FM, Layer, Voice, DSP, and Mixer Editor
4. Compose music using 3rd party MIDI sequencers - Compress MIDI data to download.
5. Modify 68000 programs using SDSS*.
6. Use the Sound Simulator to collect Tone Banks, Sequences, etc.. for game to download.
7. Download to target using the Sound Simulator.
8. Use SSBug to debug program*.

***optional**

How to start up the Sound Box

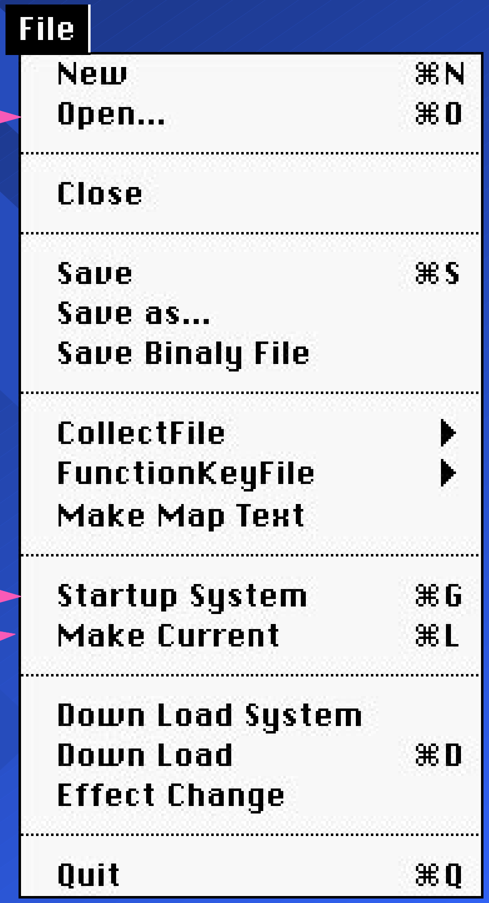
Sound Simulator 1.28

1. Start the system
2. Open Demo Map
3. Make Current

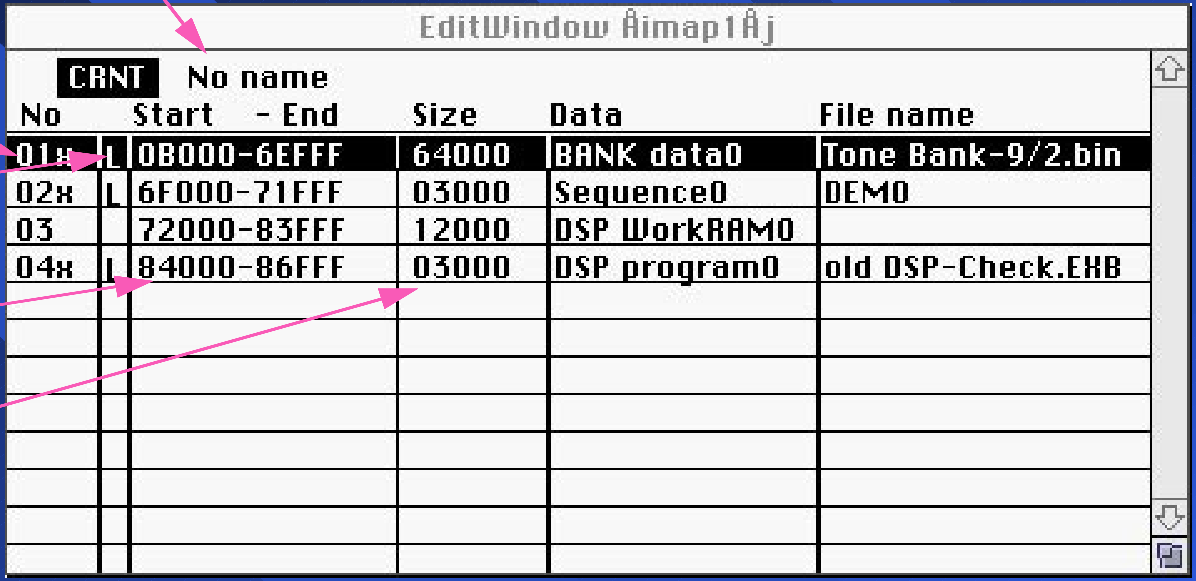
2nd

1st

3rd



MAP Editor



EditWindow Åimap1Åj

CRNT No name

No		Start - End	Size	Data	File name
01x	L	0B000-6EFFF	64000	BANK data0	Tone Bank-9/2.bin
02x	L	6F000-71FFF	03000	Sequence0	DEMO
03		72000-83FFF	12000	DSP WorkRAM0	
04x	L	84000-86FFF	03000	DSP program0	old DSP-Check.EXB

Map Name

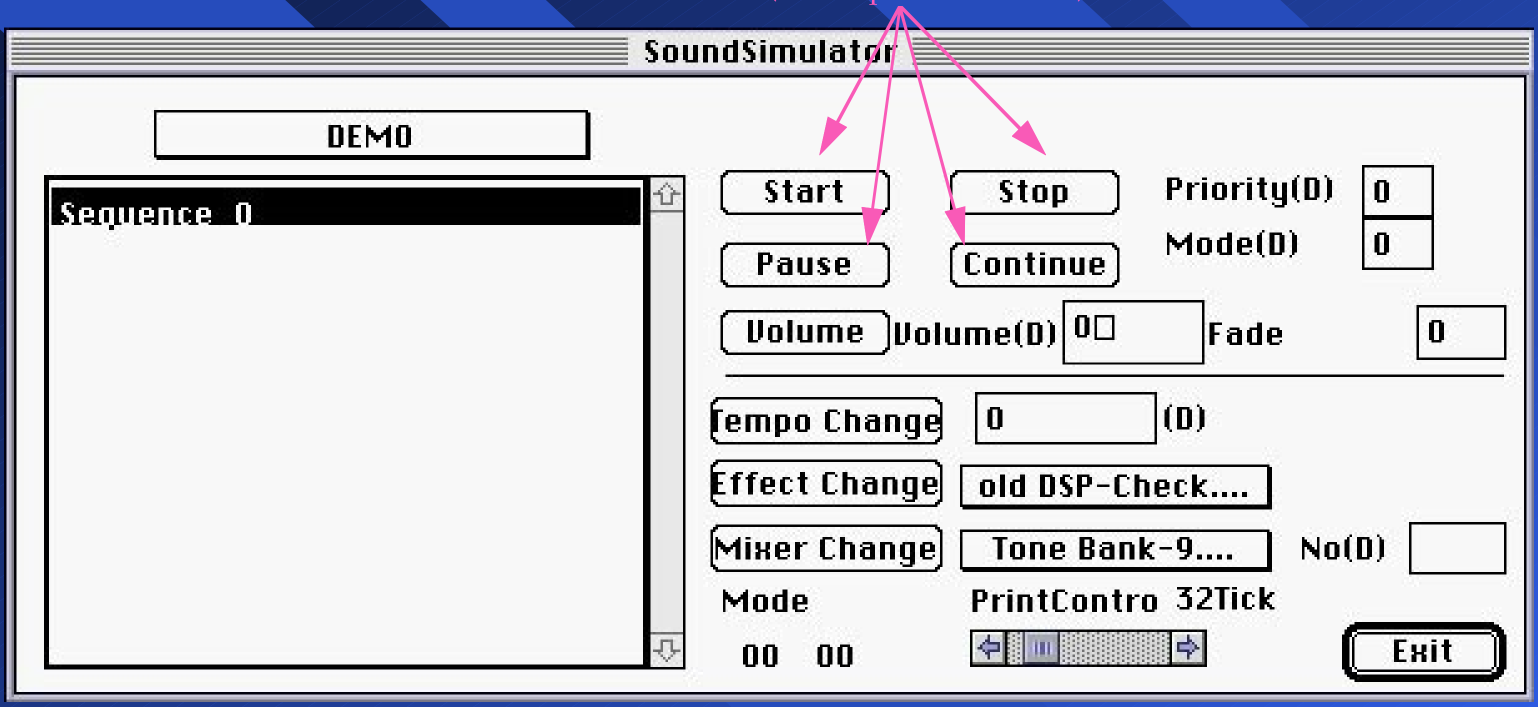
Map Data bank number

Loaded into map automatically

Memory addresses (in hex)

Memory size (in hex)

The Sound Simulator



Start, Stop, pause, and continue
(one sequence at a time)

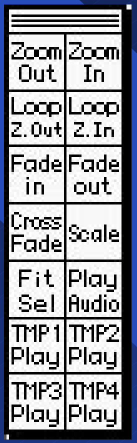
Sequence 0

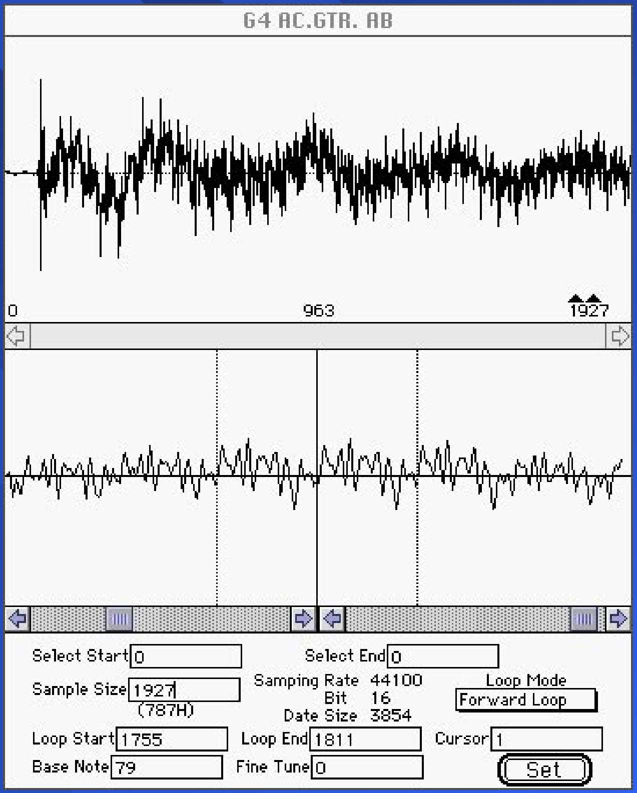
Priority(D) 0
Mode(D) 0
Volume(D) 0□
Fade 0
0 (D)
old DSP-Check....
Tone Bank-9....
No(D)
Mode
00 00
PrintContro 32Tick

Building from Scratch

- Collect Samples
  - Sample Editing
- Create a Tone Bank
  - Patch construction
  - Mixer configuration
- Compose the Music
  - Convert the MIDI file
- Add the DSP
- Pull it all together in a Map
- Make the files for the game

Waveform Editor





Select Start 0

Select End 0

Sample Size 1927
(787H)

Samping Rate 44100
Bit 16
Date Size 3854

Loop Mode
Forward Loop

Loop Start 1755

Loop End 1811

Cursor 1

Base Note 79

Fine Tune 0

Waveform Editor Functions

- Capture Sound
- Loop
- Resample
- Pitch Shift
- Size Shift
- Scale
- Filter
- Compression
- Noise Gate
- Cross Fade
- Fade In
- Fade Out

The Tone Editor

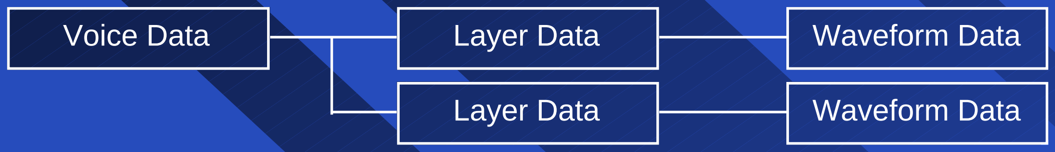
SCSP

ToneEditor

Ver 1.2

Sega

Voice Architecture



Voice Data

Layer Data

Layer Data

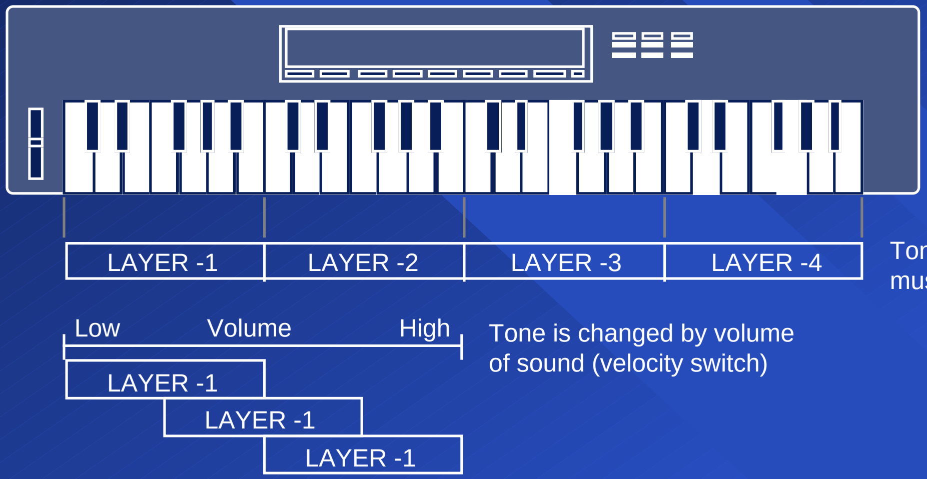
Waveform Data

Waveform Data

Voice Data

Layer Data

Waveform Data



LAYER -1

LAYER -2

LAYER -3

LAYER -4

Tone is changed by
musical interval (key split)

Low

Volume

High

LAYER -1

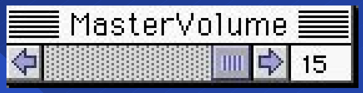
LAYER -1

LAYER -1

Tone is changed by volume
of sound (velocity switch)

Voice Window

Over all Volume
(indicates that program
is up and running)

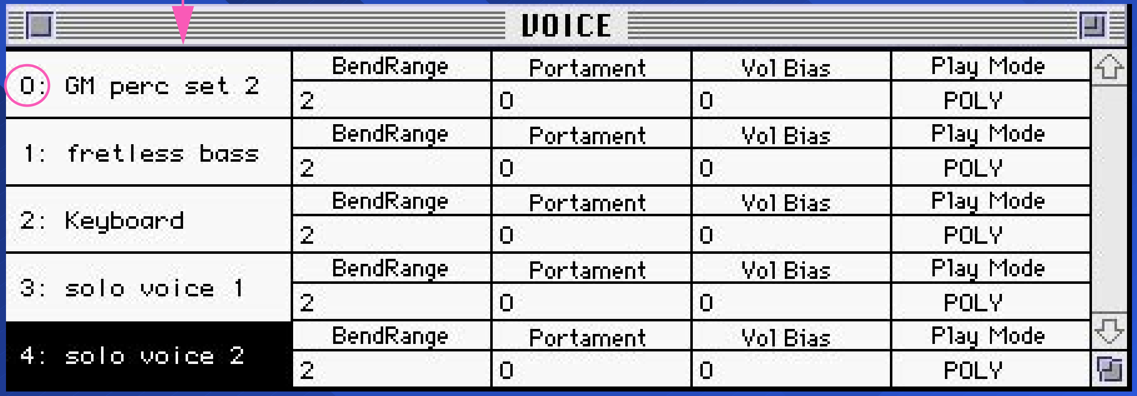


MasterVolume

15

Patch Name

Patch number
(double click
here to access
the Layers)



VOICE

0: GM perc set 2	BendRange	Portament	Vol Bias	Play Mode
	2	0	0	POLY
1: fretless bass	BendRange	Portament	Vol Bias	Play Mode
	2	0	0	POLY
2: Keyboard	BendRange	Portament	Vol Bias	Play Mode
	2	0	0	POLY
3: solo voice 1	BendRange	Portament	Vol Bias	Play Mode
	2	0	0	POLY
4: solo voice 2	BendRange	Portament	Vol Bias	Play Mode
	2	0	0	POLY

Layer Window

Select this button to make it an FM modulator

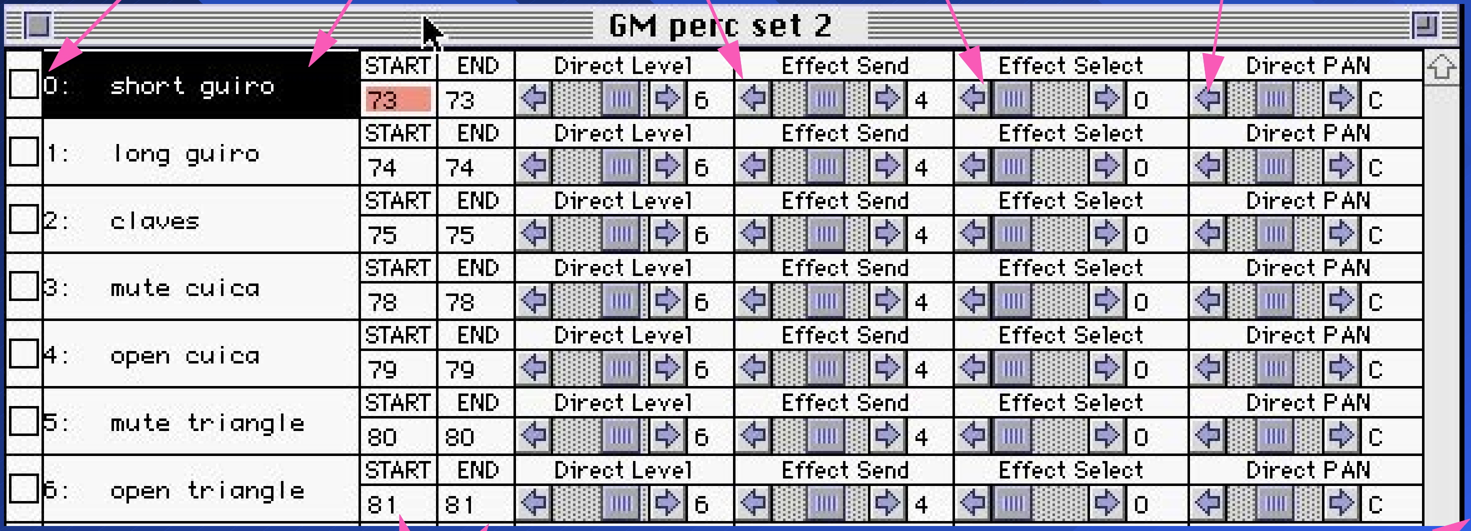
Layer number (double click here to access the Slot Window)

Layer Name (double click here to edit layer name)

The amount "DRY" volume

The amount "wet" volume sent

The number that corresponsed to the effect module



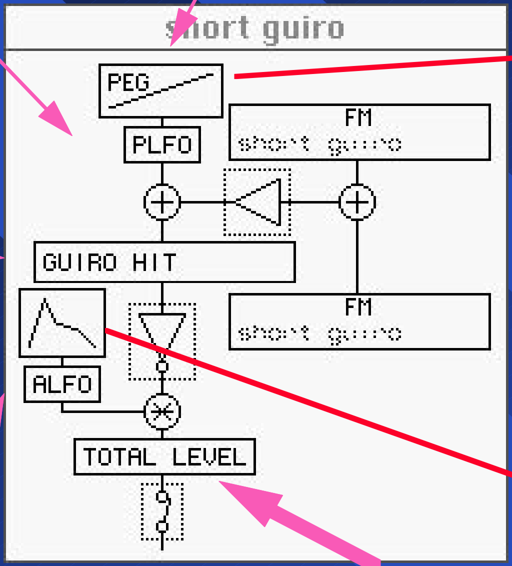
GM perc set 2

	START	END	Direct Level	Effect Send	Effect Select	Direct PAN
□ 0: short guiro	73	73	6	4	0	C
□ 1: long guiro	74	74	6	4	0	C
□ 2: claves	75	75	6	4	0	C
□ 3: mute cuica	78	78	6	4	0	C
□ 4: open cuica	79	79	6	4	0	C
□ 5: mute triangle	80	80	6	4	0	C
□ 6: open triangle	81	81	6	4	0	C

MIDI note number

The amount "DRY" signal panning

Slot Window



Pitch Low Frequency Oscillator

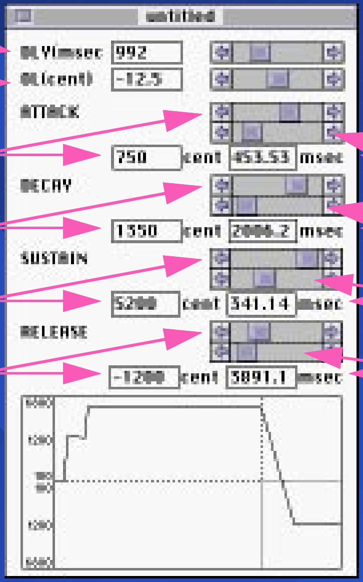
Pitch Envelope Generator

Sample name (double click to load in Sample)

Amplitude Envelope

Amplitude Low Frequency Oscillator

Defaults to 0



DLY(msec) 992

OL(cent) -12.5

ATTACK
750 cent 453.53 msec

DECAY
1350 cent 2006.2 msec

SUSTAIN
5200 cent 341.14 msec

RELEASE
-1200 cent 5891.1 msec

Delay in Milliseconds

Offset Level

Attack Level

Decay Level

Sustain Level

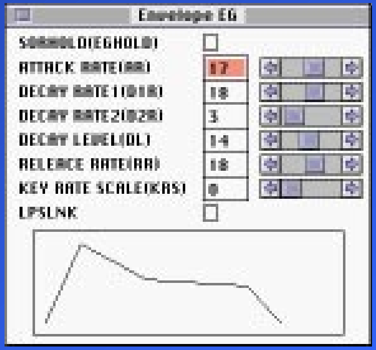
Release Level

Attack Time

Decay Time

Sustain Time

Release Time



SAMHOLD(EGHOLD) □

ATTACK RATE(AR) 17

DECAY RATE1(D1R) 18

DECAY RATE2(D2R) 3

DECAY LEVEL(DL) 14

RELEASE RATE(RR) 18

KEY RATE SCALE(KRS) 0

LPSLNK □

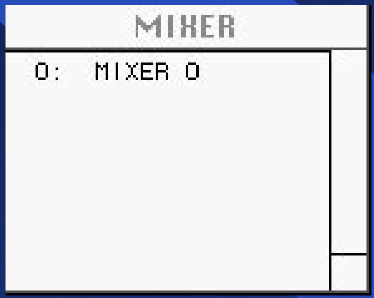
FM

- up to 32 oscillators
  - one oscillator - voice
  - no more than 2 modulators per carrier
  - multi feed back
- multiple wave types
  - sine
  - cosine
  - triangle
  - saw

Mixer

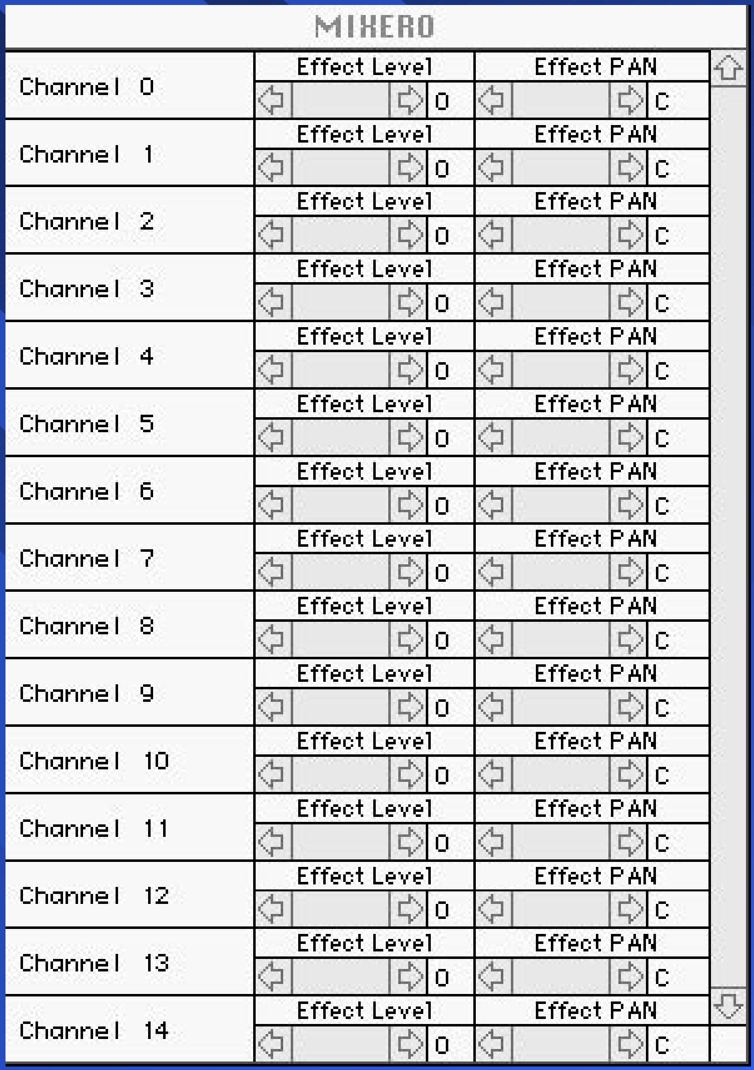
- 16 channel digital mixer
- Flexible effects and bypass processing
- Routes slot sound data through DSP module

Mixer



MIXER

0: MIXER 0



MIXER0

	Effect Level	Effect PAN
Channel 0	0	C
Channel 1	0	C
Channel 2	0	C
Channel 3	0	C
Channel 4	0	C
Channel 5	0	C
Channel 6	0	C
Channel 7	0	C
Channel 8	0	C
Channel 9	0	C
Channel 10	0	C
Channel 11	0	C
Channel 12	0	C
Channel 13	0	C
Channel 14	0	C

Third Party Sequence Editor

- Third party sequencer of your choice (Cubase, Logic, Vision etc.)
- Control voices and sound effects with MIDI file format type 1
- Compress MIDI files using Sound Simulator

MIDI file Converter Specifications

Can

Note On/Off

Poly-Key Pressure

Control Change

Channel Pressure

Pitch Wheel Change

Can NOT

Meta Event

System Exclusive Messages

Start

Stop

Song Position

MIDI Channels and Voices

- Up to 32 simultaneous channels
  - 2 banks of 16
- 32 instruments simultaneously
  - 0-31 priority 0 being the highest
- 8 sound control numbers
  - one sequence per sound control number
  - 127 direct MIDI calls per sound control numbers

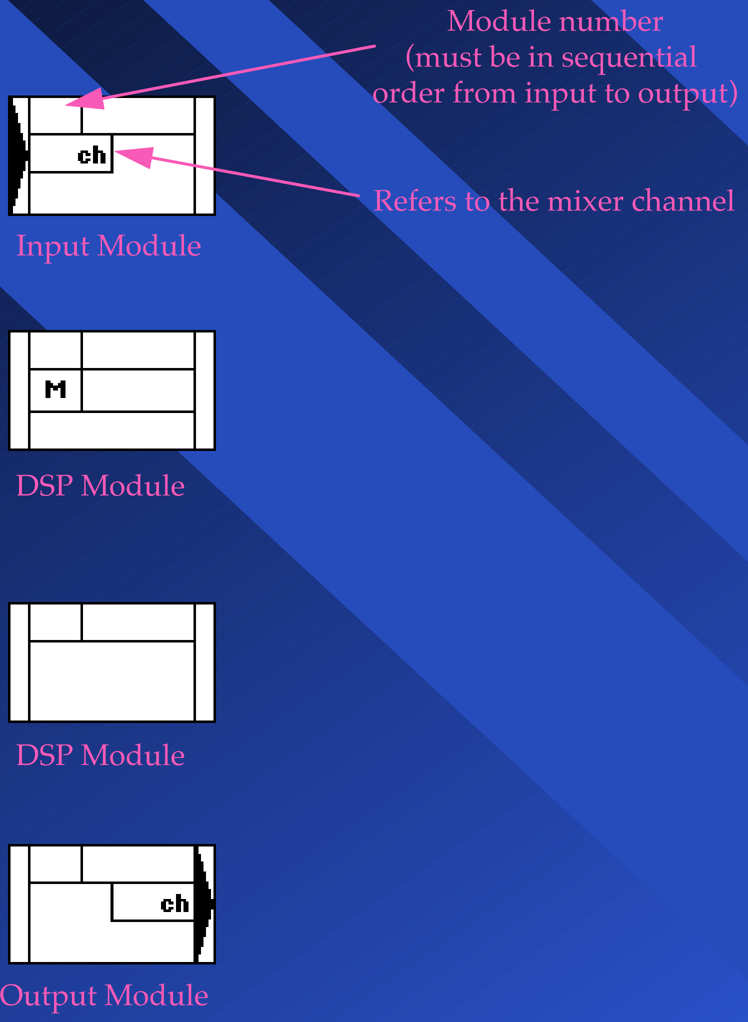
The DSP Linker

- Configure Input and Output Module
- Select Effect Module
- Set up Mixer Configuration in the Tone Editor
- Play music from sequencer
- Adjust DSP configuration
- Link and Download
- Save DSP files
- Put files into map
- Use effect change (command 83h) to select DSP setting
- Use mixer change (command 87h) to select mixer setting

DSP Modules

- Effects
  - Reverb
  - Early Reflection
  - Echo (Delay)
  - Pitch Shifter
  - Chorus
  - Flanger
  - Symphonic
  - Yamaha 3D
  - Surround
  - Voice Cancel
  - Auto Pan
  - Phaser
  - Distortion
  - Filter
  - Parametric EQ
  - QSound

DSP Editor



Module number (must be in sequential order from input to output)

Refers to the mixer channel

Input Module

DSP Module

DSP Module

Output Module

QSound on the Saturn

- Sound spatialization processing for Saturn
- Produces 3D type effects
- Uses the eLinker (Enhanced Linker)
- 4 Channel and 8 Channel versions available
- Must use sound driver 1.31
- Use control command 12h for QSound control

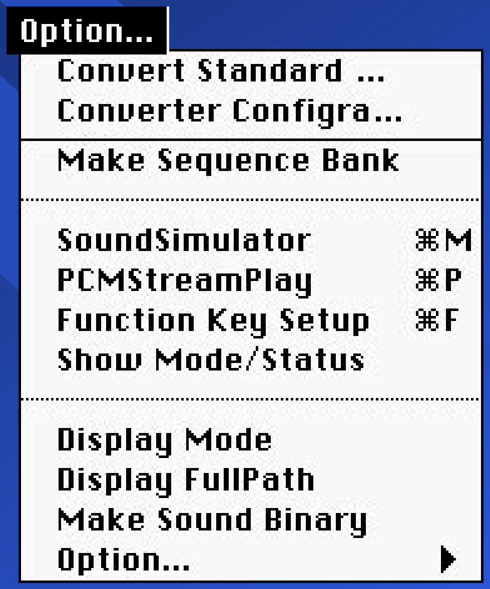
Yamaha 3D Sound on the Saturn

- True 3D sound processing for Saturn
- Full 360 degree spatialization
- Best results on headphones
- Yamaha 3D module included in the eLinker
- Must use sound driver version 1.31
- 1 and 2 channel versions will be available
- Use control command 11h for 3D sound control

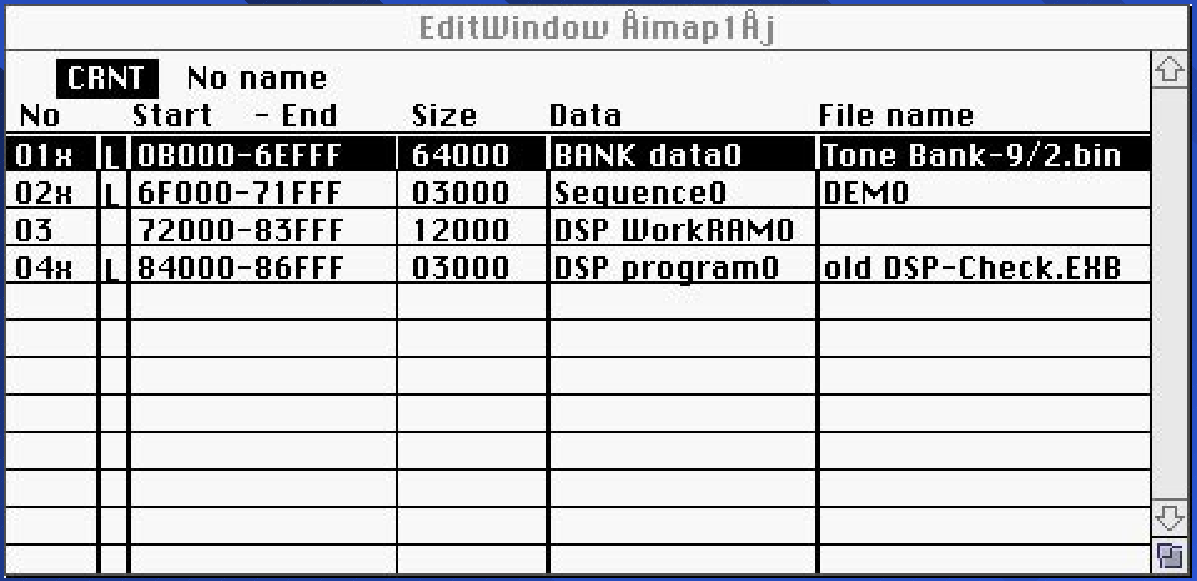
Displaying Link Results

- 1. Number of program steps.
- 2. Amount of coefficient RAM used.
- 3. Amount of address constant RAM used.
- 4. Amount of DRAM used for the ring buffer.
- 5. Amount of DRAM used for the coefficient tables.
- 6. Size of DRAM area that must be allocated for the DSP, the beginning of the area in 4 to the end of 5.

MIDI Converter



Back to the Sound Simulator

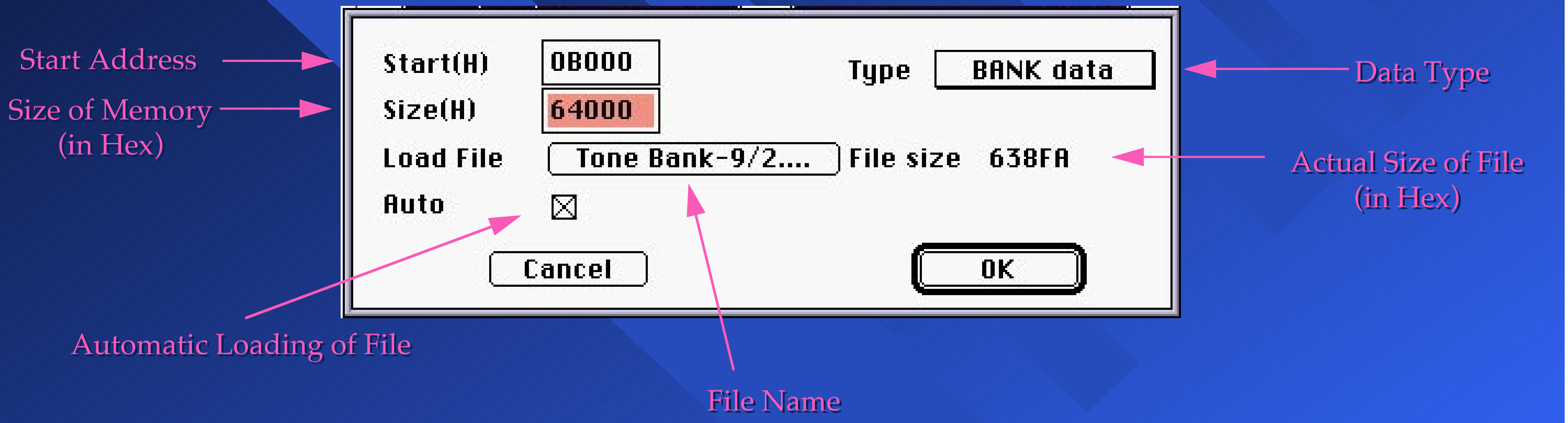


EditWindow Åimap1Åj

CRNT No name

No		Start - End	Size	Data	File name
01x	L	0B000-6EFFF	64000	BANK data0	Tone Bank-9/2.bin
02x	L	6F000-71FFF	03000	Sequence0	DEMO
03		72000-83FFF	12000	DSP WorkRAM0	
04x	L	84000-86FFF	03000	DSP program0	old DSP-Check.EXB

Data Edit Window



Start(H) 0B000

Size(H) 64000

Type BANK data

Load File Tone Bank-9/2....

File size 638FA

Auto ☒

Start Address

Size of Memory (in Hex)

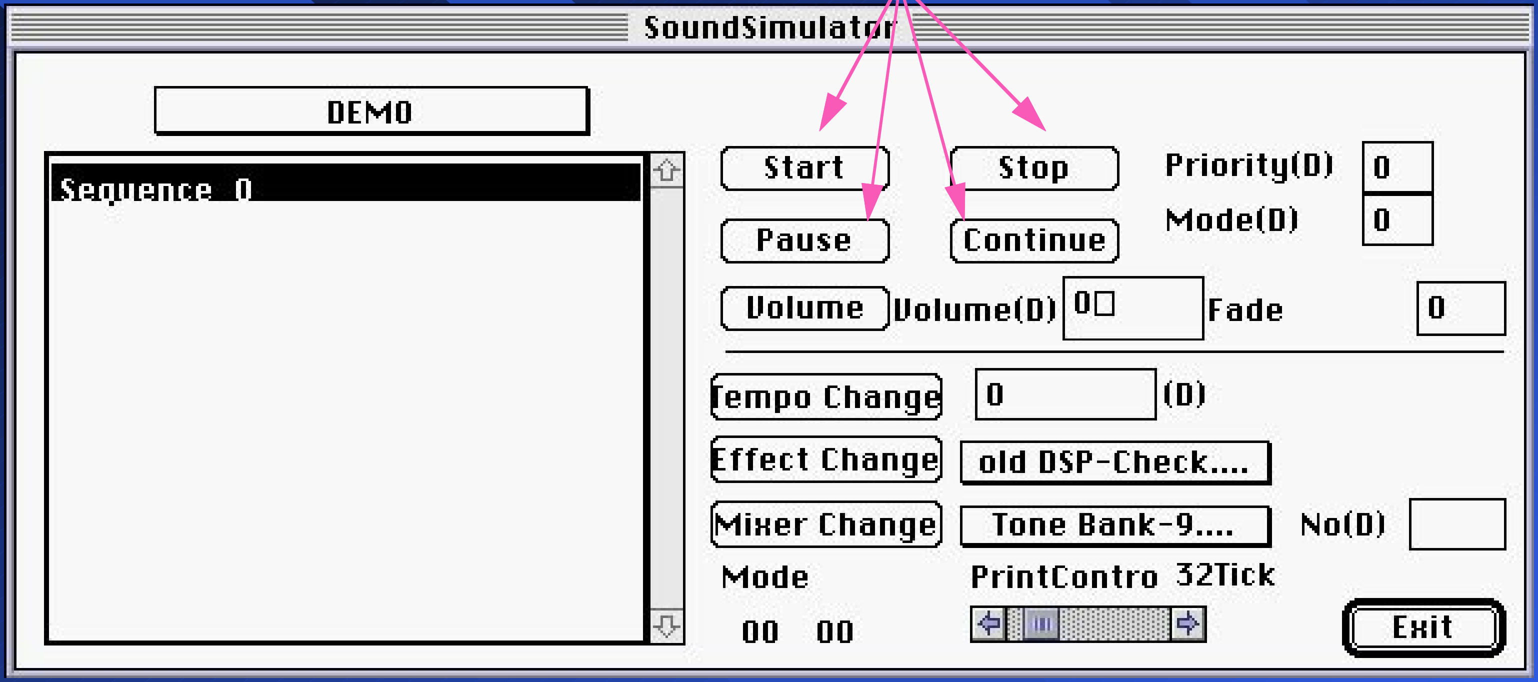
Data Type

Actual Size of File (in Hex)

Automatic Loading of File

File Name

The Sound Simulator



Sequence 0

Priority(D) 0

Mode(D) 0

Volume(D) 0□

Fade 0

Tempo Change 0 (D)

old DSP-Check....

Tone Bank-9.... No(D)

Mode PrintContro 32Tick

00 00

Start, Stop, pause, and continue
(one sequence at a time)

The Function Key Set-up

- Use this screen to test Music and Sound Effects
- Provides access to Sound Control Commands
- Test mixing of Music and Sound Effects

Saturn Sound Tools Q&A

SCSP Sound Driver

Richard Jacques
Musician
Simon Golding
Technical Consultant

Files Needed for Game Program

- Map.bin
- Map.text
- DSP.EXB
- Tone Bank.bin
- Sequence Bank.cnv

SCSP/Sound Driver

- Integrating Sound Tool Data
- Driver set-up & initialisation
- Sound Driver interface calls
- Resource allocation, including: Map layout and usage, sound control usage
- Sound memory Access

Driver Set-up & Initialisation

```
INCDATA.S

                .global         _scsp_sound_driver
                .global         _scsp_sound_driver_end
                .global         _scsp_map
                .global         _scsp_map_end


                .align          2
_scsp_sound_driver:
                .include "sddrvs.125"
_scsp_sound_driver_end:

                .align          2
_scsp_map:
                .include "example.s"
_scsp_map_end:
```

```
/****************************************************************************
 *
 *  Load and initialise sound driver, music, and sound effects data.
 *
 ****************************************************************************/

void scspInit(void)
{

 SndIniDt       snd_init;        /* address data for driver and map */
                SndSeqNum  seq_no;
                SndSeqPri seq_pri;

                handle                          fileHdl;



 SND_INI_PRG_ADR(snd_init) = (Uint16 *)&scsp_sound_driver;
 SND_INI_PRG_SZ(snd_init) = (Uint16 )0x6000;
 SND_INI_ARA_ADR(snd_init) = (Uint16 *)&scsp_map;
 SND_INI_ARA_SZ(snd_init) = (Uint16) 0x001c;
 SND_Init(&snd_init);

                SND_DspClear();
                SND_AllOff();

                SND_ChgMap(0);
```

Move Sound Tool Data

```
fileHdl = LoadFile("LEV1INST.BIN", 0);
SND_MoveData((Uint16 *) *fileHdl, STMemHandleSize(fileHdl), SND_KD_TONE, 0);
STMemFree(fileHdl);

fileHdl = LoadFile("LEV1SFX.BIN", 0);
SND_MoveData((Uint16 *) *fileHdl, STMemHandleSize(fileHdl), SND_KD_TONE, 1);
STMemFree(fileHdl);

fileHdl = LoadFile("LEV1MUSS.BIN", 0);
SND_MoveData((Uint16 *) *fileHdl, STMemHandleSize(fileHdl), SND_KD_SEQ, 0);
STMemFree(fileHdl);

fileHdl = LoadFile("LEV1SFXS.BIN", 0);
SND_MoveData((Uint16 *) *fileHdl, STMemHandleSize(fileHdl), SND_KD_SEQ, 1);
STMemFree(fileHdl);

fileHdl = LoadFile("LEV1DSP.BIN", 0);
SND_MoveData((Uint16 *) *fileHdl, STMemHandleSize(fileHdl), SND_KD_DSP_PRG, 0);
STMemFree(fileHdl);
```

Triggering SFX

```
/*-----------------03-08-95 02:41pm-----------------
                This shows how to play 1-shot sound effects
                First do a Bank Select to setup program change
---------------------------------------------------*/

                SND_CtrlDirMidi(SNDCTRL1,SCSP_PRI1,MIDICTRLCHG,MIDICHAN1,32,1);

/*-----------------03-08-95 02:42pm-----------------
                Now do a patch select
---------------------------------------------------*/
                SND_CtrlDirMidi(SNDCTRL1,SCSP_PRI1,MIDIPRGCHG,MIDICHAN1,SFX_PATCH,0);

/*-----------------03-08-95 02:43pm-----------------
                Now fire off some 1-shot sound effects
---------------------------------------------------*/
                SND_CtrlDirMidi(SNDCTRL1,SCSP_PRI1,MIDINOTEON,MIDICHAN1,Splatfire,127);
                SND_CtrlDirMidi(SNDCTRL1,SCSP_PRI1,MIDINOTEON,MIDICHAN1,EngineRev,127);
                SND_CtrlDirMidi(SNDCTRL1,SCSP_PRI1,MIDINOTEON,MIDICHAN1,SmallExplode,127);
```

Triggers Music and SFX by sequences

```
/*-----------------03-08-95 02:46pm-----------------
 Start a music sequence
---------------------------------------------------*/
#define MusicBank 0
#define LevelTune1 1

            seq_pri = 15;

            SND_StartSeq(SNDCTRL0, MusicBank, LevelTune1, seq_pri);

#define SFXBank 0
#define ComplexSFX1 1

            seq_pri = 1;
            SND_StartSeq(SNDCTRL1, SFXBank, ComplexSFX1, seq_pri);
```

Preparing Red Book Audio Data

- Compose Music
- Prepare DAT master of 16 bit 44.1kHz format
- Prepare audio file (for example using Sound Designer)
- It is often recommended to use “normalise” functions
- Save audio file in RAW format (i.e.. NOT AIFF)
- Swap bytes around using “swap.exe” program
- Add file to disk image
- Stream Red Book using sound driver commands

PCM Streaming

```
bnk_dest_addr = 2;   /* map buffers 2 and 3 used for streaming */
gfs1 = StreamFile("SURFIN.BIN");      /* one pcm channel */
intr(gfs1);                      /* prime buffers **    */
intr(gfs1);                      /* to fill BOTH BUFFERS */

SND_PRM_MODE(pcm_start) = SND_MD_STEREO | SND_MD_16;
SND_PRM_SADR(pcm_start) = 0x1000;
SND_PRM_SIZE(pcm_start) = 0x2000;    /* per channel buffer size */
SND_PRM_OFSET(pcm_start) = 0;
SND_PRM_NUM(pcm_chg) = 2;
SND_PRM_LEV(pcm_chg) = 7;
SND_PRM_PAN(pcm_chg) = 0;
SND_PRM_PICH(pcm_chg) = 0;
SND_L_EFCT_IN(pcm_chg) = 0;
SND_L_EFCT_LEV(pcm_chg) = 7;
SND_R_EFCT_IN(pcm_chg) = 0;
SND_R_EFCT_LEV(pcm_chg) =  7;

        SND_StartPcm(&pcm_start, &pcm_chg);             /* start up 1 voice playing PCM */
```

PCM Streaming cont.

```
void intr(GfsHn gfs1)
{
        extern Uint32 buf1a[];
        extern Uint32 buf1b[];
        extern Uint32 *buf1[];
        Sint32 stat,nbytes;
        static Uint8 i=0;
        i=!i;
        do
        {
                GFS_NwFread(gfs1, RD_UNIT, buf1[i], RD_UNIT * SECT_SIZE);
                GFS_NwExecOne(gfs1);
                GFS_NwGetStat(gfs1,&stat,&nbytes);
        } while (nbytes < RD_UNIT * SECT_SIZE);

                SND_MoveData((Uint16 *)buf1[!i],            /* one channel */
                        (Uint32)0x2000,                             /* size */
                        SND_KD_TONE,
                        bnk_dest_addr);
  bnk_dest_addr = ~bnk_dest_addr & 1;
}
```

The System Control Unit

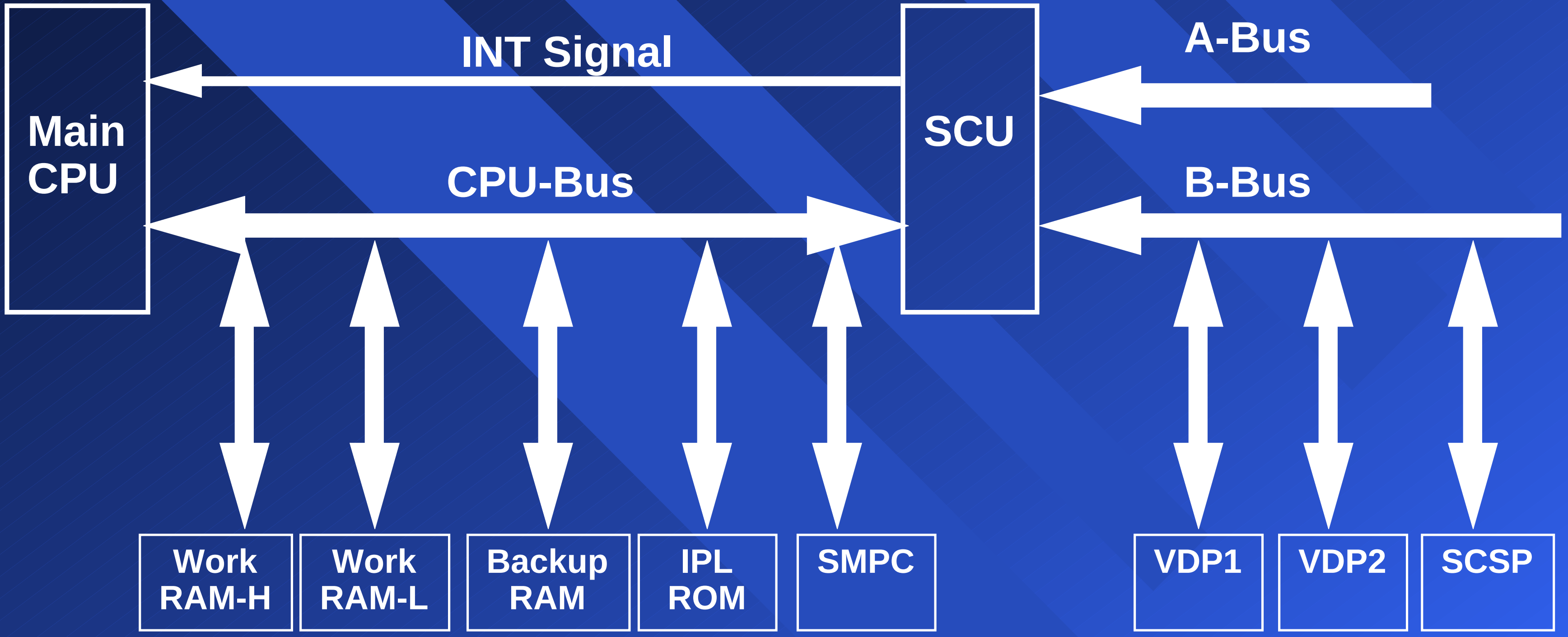
Richard Parr
Advanced Technology Group
SoE Internal Product Development

Topics

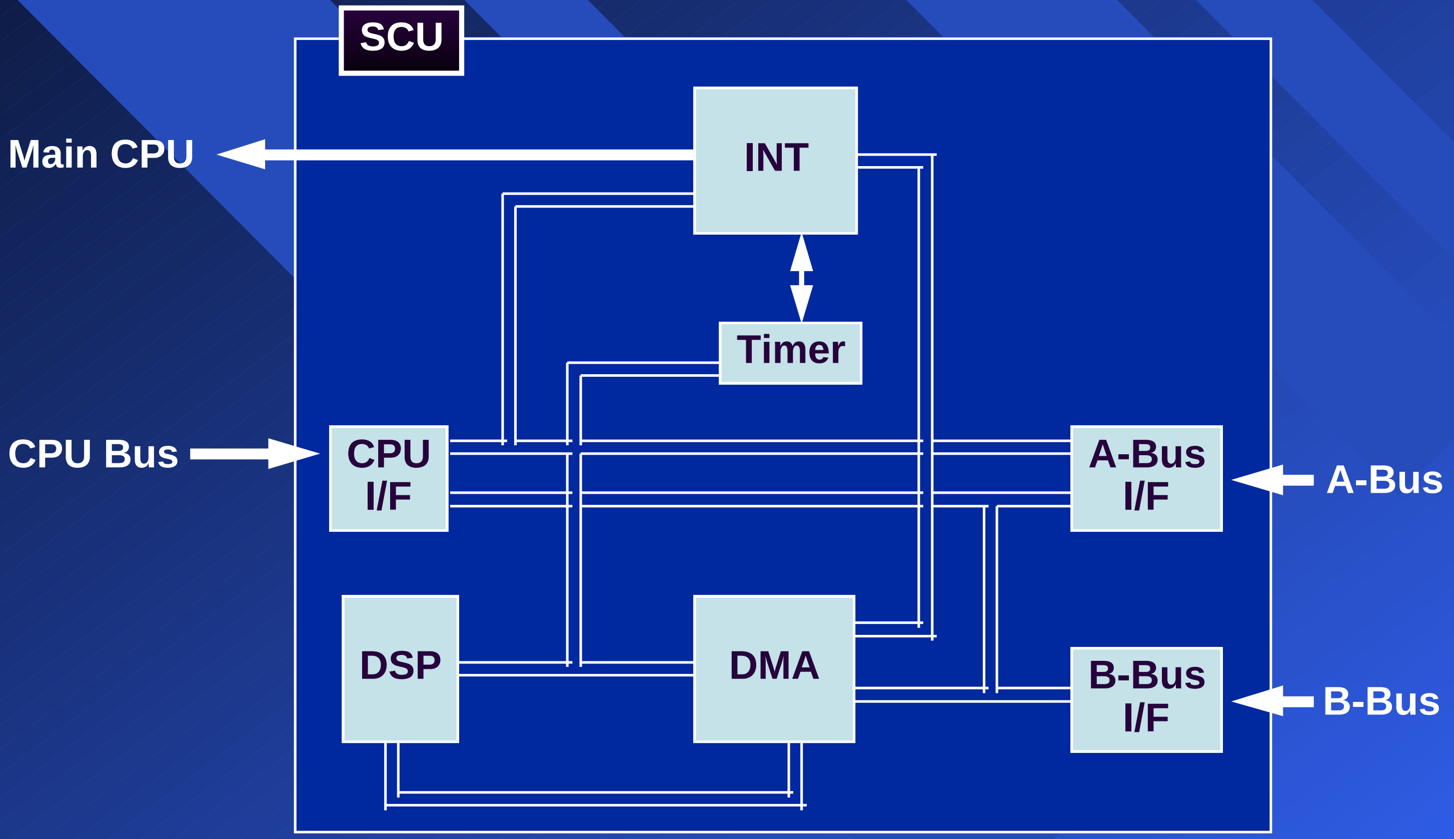
- Overview of the SCU
  - Address Translation
  - DMA
  - Timers
  - Interrupts
  - Q & A
- Digital Signal Processor Programming
  - DSP architecture
  - Instruction format
  - SH2 / DSP communication
  - Programming example
  - Q & A

SCU Overview

- The SCU “glues together” the principal components of the Saturn



SCU Block Diagram

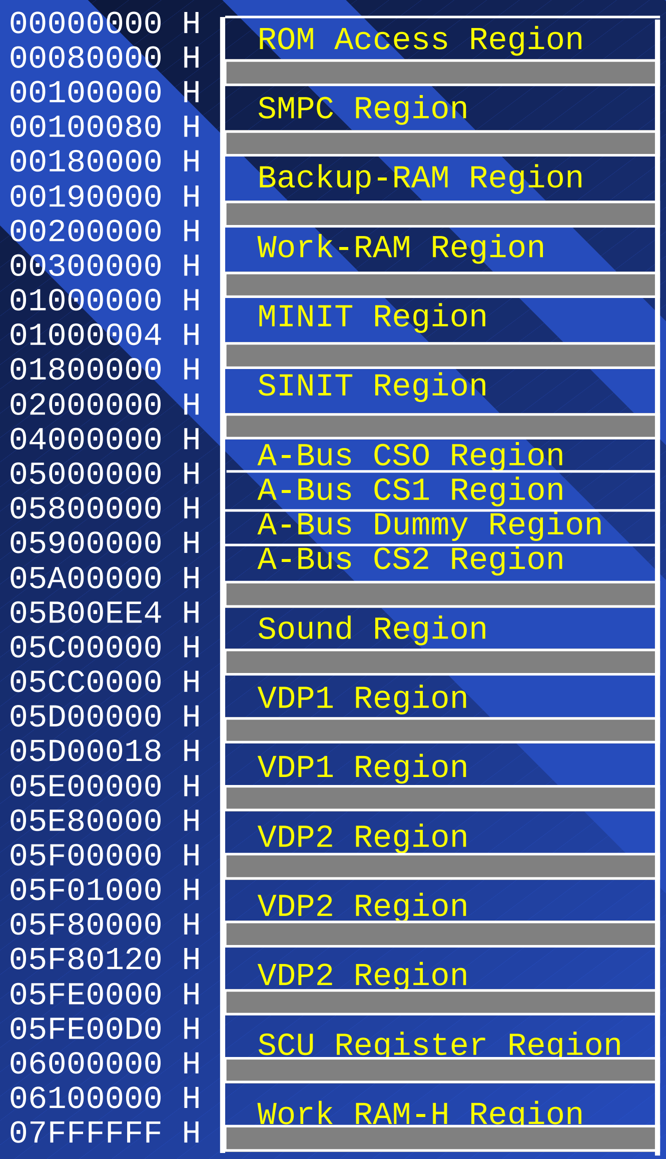


SCU Features

SCU Functions

- Processor Interface / Address Translation
- DMA Controller
- Timers
- Interrupt Controller
- Digital Signal Processor

Address Translation



00000000 H
00080000 H
00100000 H
00100080 H
00180000 H
00190000 H
00200000 H
00300000 H
01000000 H
01000004 H
01800000 H
02000000 H
04000000 H
05000000 H
05800000 H
05900000 H
05A00000 H
05B00EE4 H
05C00000 H
05CC0000 H
05D00000 H
05D00018 H
05E00000 H
05E80000 H
05F00000 H
05F01000 H
05F80000 H
05F80120 H
05FE0000 H
05FE00D0 H
06000000 H
06100000 H
07FFFFFF H

ROM Access Region
SMPC Region
Backup-RAM Region
Work-RAM Region
MINIT Region
SINIT Region
A-Bus CS0 Region
A-Bus CS1 Region
A-Bus Dummy Region
A-Bus CS2 Region
Sound Region
VDP1 Region
VDP1 Region
VDP2 Region
VDP2 Region
VDP2 Region
SCU Register Region
Work RAM-H Region

▨ indicates areas that can't be accessed

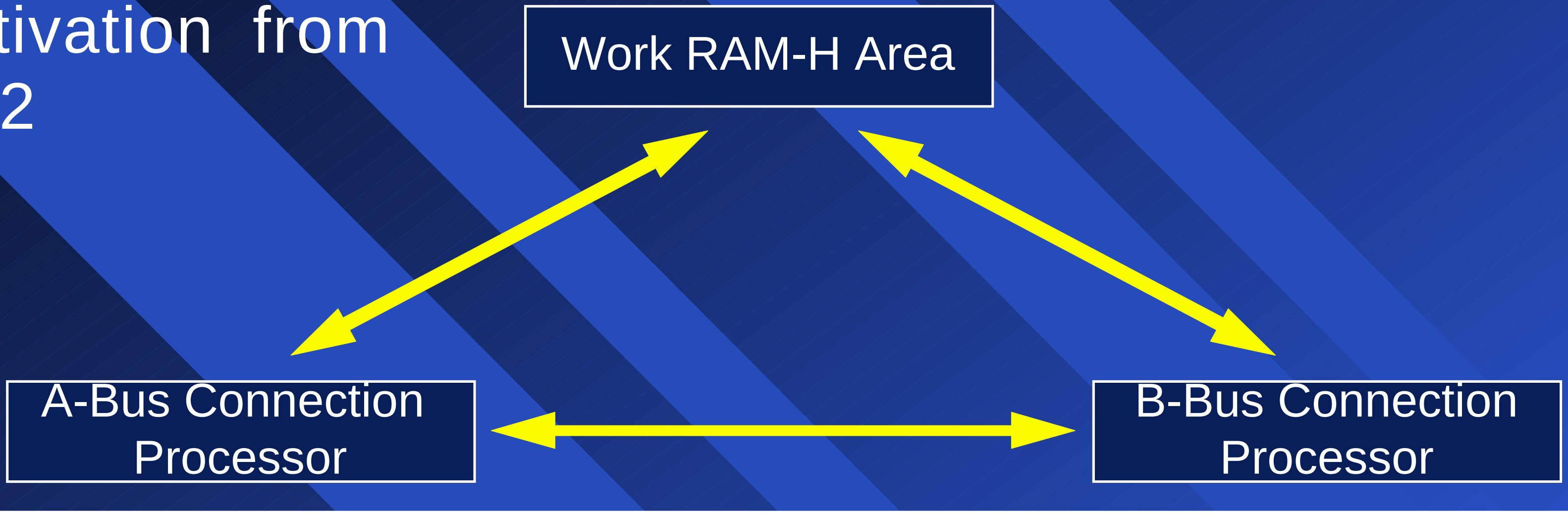
- Maps peripheral components into SH2 address space

SCU DMA

- 3 priority levels
  - Level 0: Upto 1 megabyte , lowest priority
  - Level 1: Upto 4 kilobytes
  - Level 2: Upto 4 kilobytes, highest priority
- 2 DMA channels active concurrently
- Activated by CPU or DSP
- Direct or Indirect modes
- DMA triggered manually or on signal (VBLANK IN/OUT, HBLANK-IN, timers, sound, sprite draw end)
- Interrupts triggered on end of transfer

SCU DMA

Activation from SH2

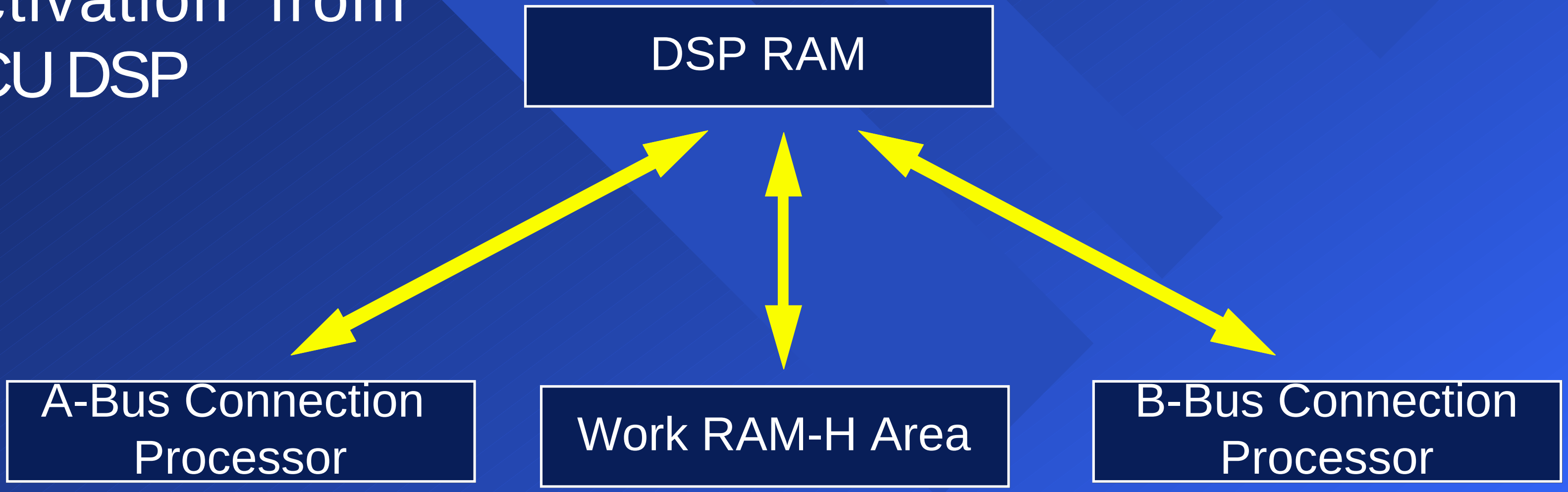


Work RAM-H Area

A-Bus Connection Processor

B-Bus Connection Processor

Activation from SCU DSP



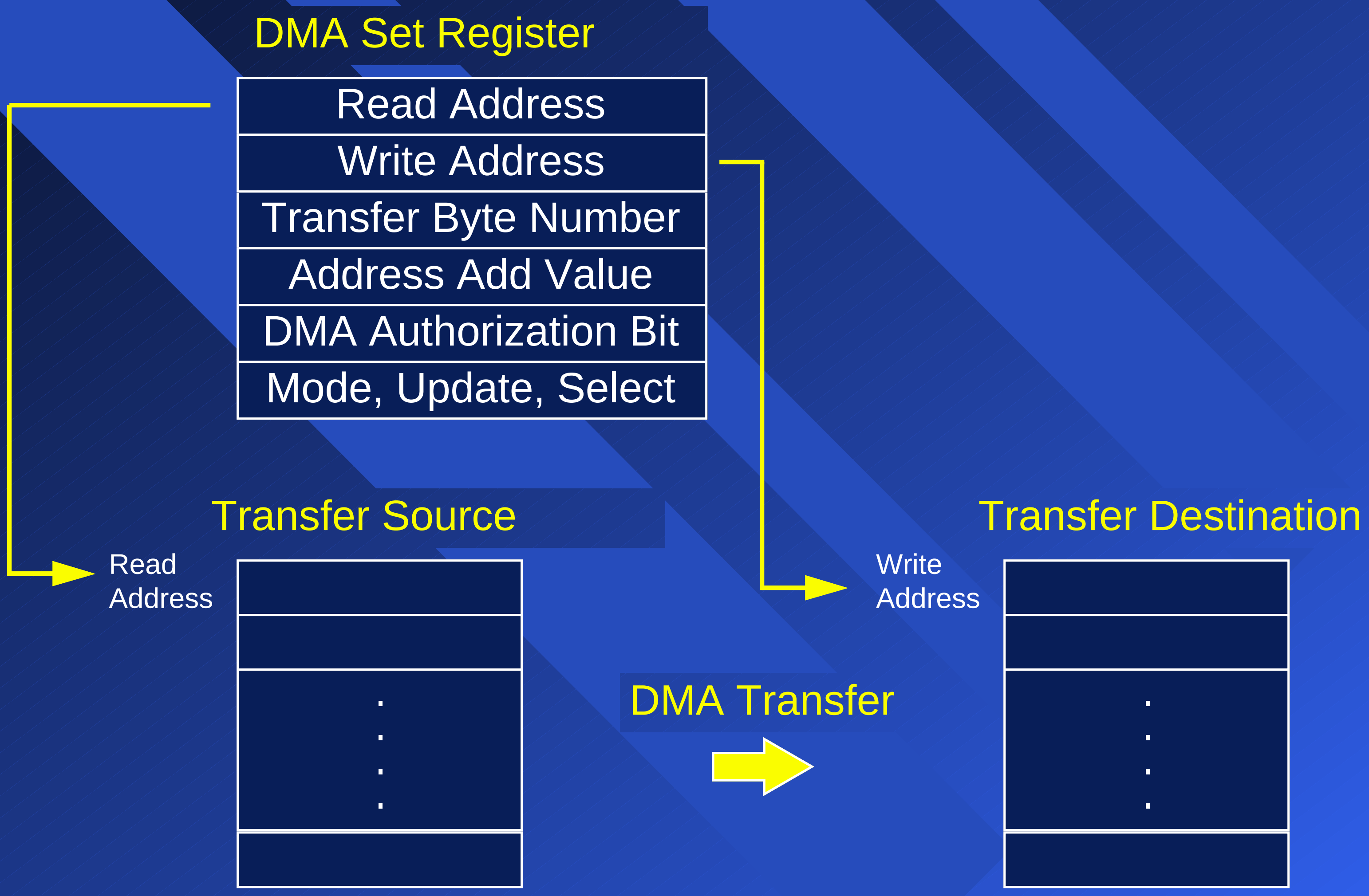
DSP RAM

A-Bus Connection Processor

Work RAM-H Area

B-Bus Connection Processor

SCU DMA - Direct Mode



DMA Set Register

- Read Address
- Write Address
- Transfer Byte Number
- Address Add Value
- DMA Authorization Bit
- Mode, Update, Select

Transfer Source

Read Address

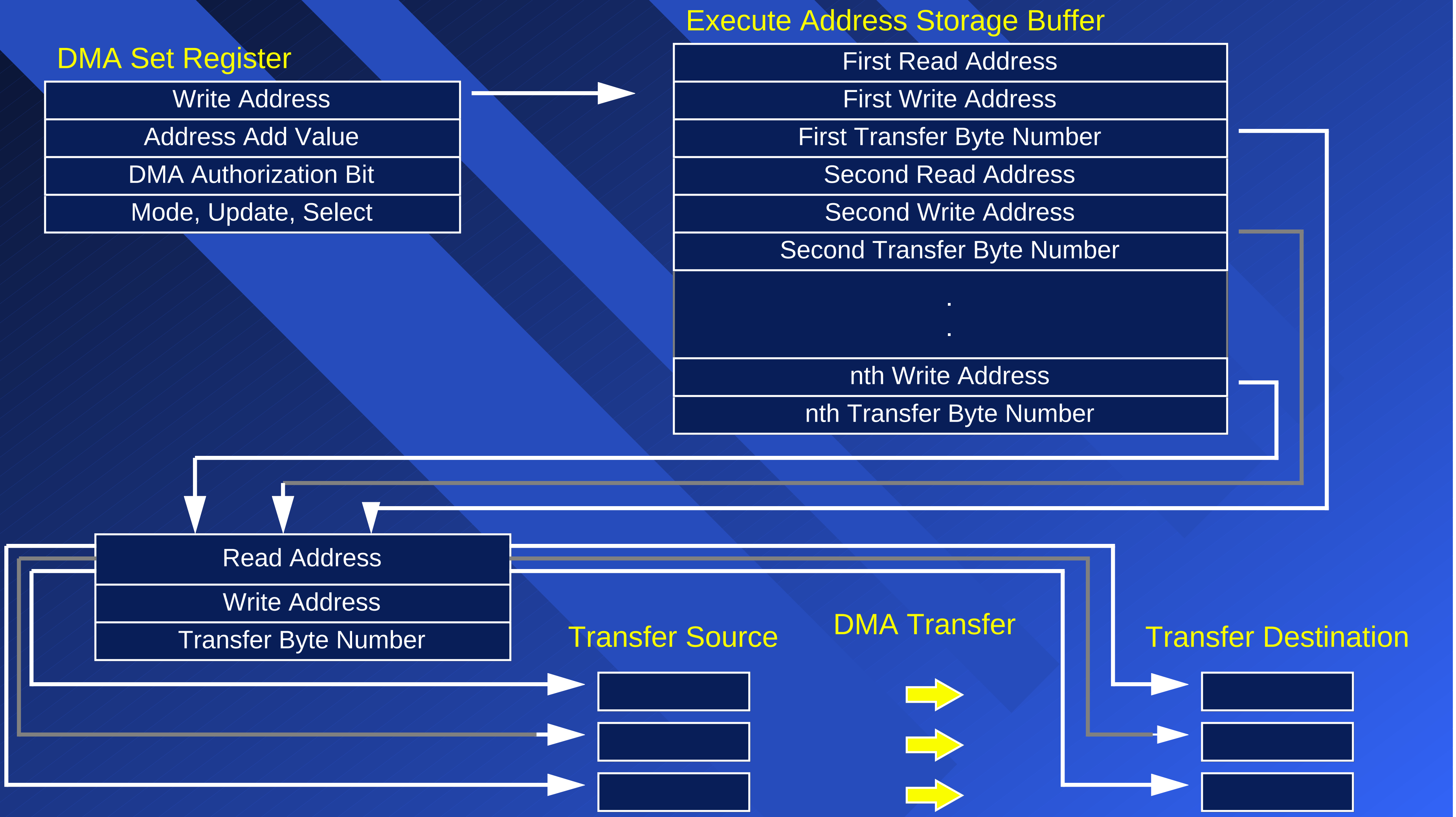
Transfer Destination

Write Address

DMA Transfer

ST-97, pg. 18

SCU DMA - Indirect Mode



DMA Set Register

- Write Address
- Address Add Value
- DMA Authorization Bit
- Mode, Update, Select

Execute Address Storage Buffer

- First Read Address
- First Write Address
- First Transfer Byte Number
- Second Read Address
- Second Write Address
- Second Transfer Byte Number
- .
- .
- nth Write Address
- nth Transfer Byte Number

- Read Address
- Write Address
- Transfer Byte Number

Transfer Source

DMA Transfer

Transfer Destination

SCU Timers

- Timer 0
  - Reset in sync with V-BLANK-OUT interrupt
  - Increments after each horizontal line drawn on screen
  - Triggers interrupt when value equals that in Timer 0 Compare Register
- Timer 1
  - Resets in sync with H-BLANK-IN interrupt to value in Timer 1 Data Register
  - Counts down as pixels plotted in the current horizontal line
  - Triggers interrupt on every line or only at line indicated by Timer 0 (Timer 1 Mode Register)

SCU Interrupt Control

- Interrupt Mask Register
  - 1 bit per interrupt
  - Place '1' in relevant bit to DISABLE interrupt
- Interrupt Status Register
  - Read - 1 indicates interrupt will occur, 0 that it will not
  - *THIS IS A READ ONLY REGISTER*

The DSP (Digital Signal Processor)

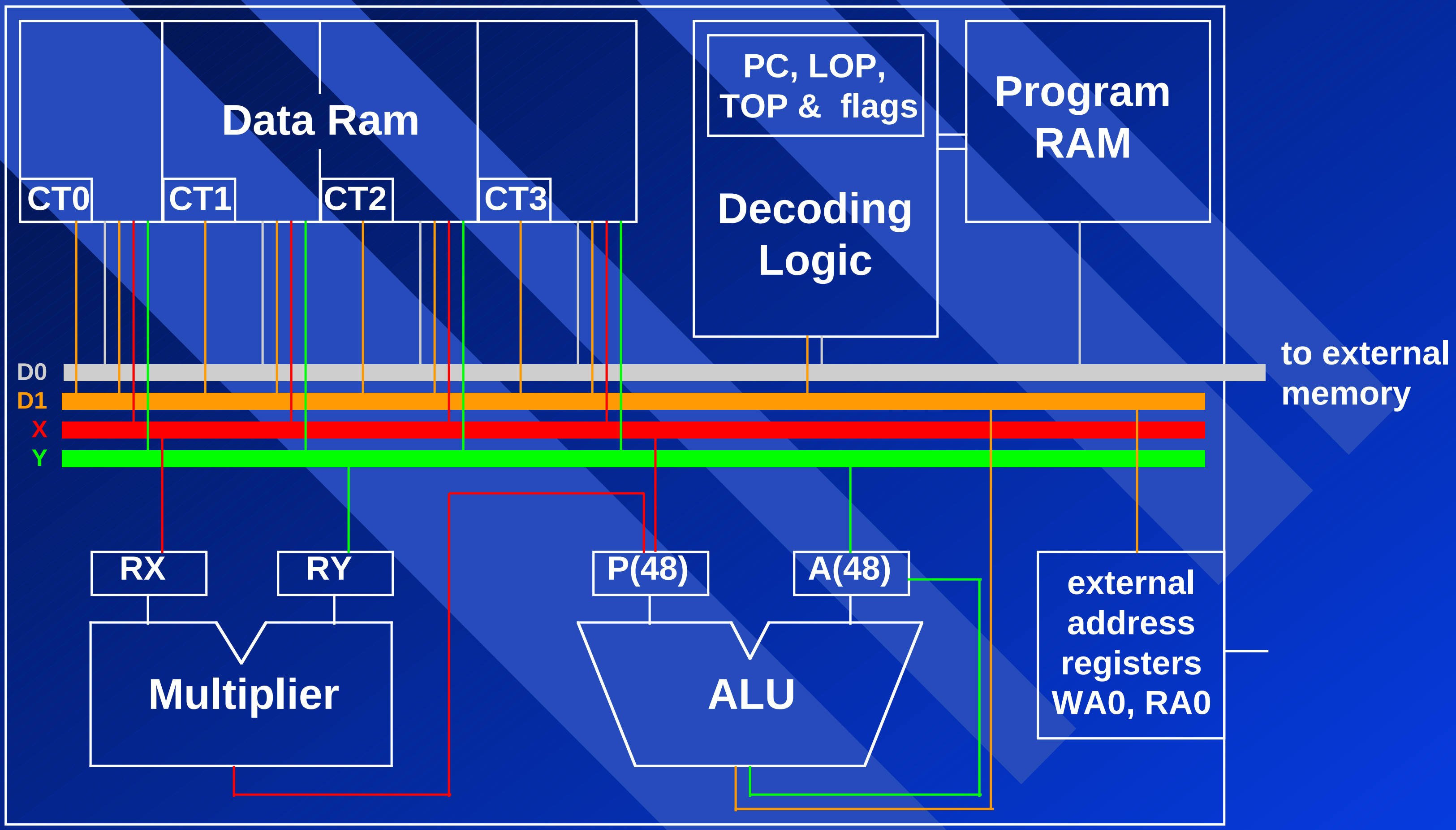
- What is a DSP?
  - Specialised microprocessor
  - Optimised for multiplication of streams of data

- What can I use one for?
  - Matrix operations
  - Image processing
  - Audio processing

SCU DSP Features

- 14 MHz operation
- Single instruction may comprise many commands
- Dedicated single-cycle multiplier
- On-chip data and program RAM
- Instructions for control of SCU DMA channels

Architecture



Memory

- Data RAM
  - –4 banks (MD0, MD1, MD2, MD3)
  - –64 x 32 bit words per bank
  - –Access via index registers (CT0, CT1, CT2, CT3) or DMA

- Program RAM
  - –256 x 32 bit words
  - –1 instruction per word (many commands per instruction)

Multiplier

- Single cycle multiplication unit
- Input registers (RX, RY) 32 bits
- Output (MUL) 48 bits

Programming

- Logical Operations (AND, OR, XOR, SR, RR, SL, RL, RL8)
- Arithmetic Operations (ADD, AD2, SUB)
- Input registers (A, P) 48 bits
- Output register (ALU) 48 bits
- *Result of ALU operation should be moved out of ALU in same instruction*

Buses

- X Bus
  - Connects data RAM banks to multiplier input (RX)
  - Connects data RAM banks to ALU input (P)
  - Connects multiplier output (MUL) to ALU input (P)
- Y Bus
  - Connects data RAM banks to multiplier input (RY)
  - Connects data RAM banks to ALU input (A)
  - Connects ALU output to ALU input (A)

Buses

- D1 Bus
- Moves data from
  - Data RAM banks
  - ALU
  - Immediate 8 bit signed data

- Moves data to
  - Data RAM banks
  - Registers (RX, PL, RA0, WA0, LOP, TOP, CT0, CT1, CT2, CT3)

Buses

- DO Bus
- Moves data from
  - Data RAM banks
  - External memory
  - Program Counter
  - Immediate data

- Moves data to
  - Data RAM banks
  - Program RAM
  - External memory

DSP Registers

- Data RAM index registers (CT0, CT1, CT2, CT3)
- Program Counter (8 bits)
- Jump / call address (TOP)
- Loop counter (LOP)
- Multiplier input registers (RX, RY)
- ALU input registers
  - P (PL, PH) - 48 bits
  - A (AL, AH) - 48 bits
- External address registers
  - RA0 - address for external to DSP DMA transfer.
  - WA0 - address for DSP to external DMA transfer
  - Hold 32 bit, word unit addresses (address / 4)

DSP Instructions

- 32 bit instruction format
- Each instruction comprised on 1 or more “commands”
- 5 classes of command
  - ALU control
  - X Bus
  - Y Bus
  - D1 Bus
  - Other

DSP Instructions

- 2 instruction formats

31	29	25	19	13 Bit 0
0 1	ALU control	X Bus Control	Y Bus Control	D1 Bus Control

31 Bit 0
Other command

- Up to 6 commands per instruction (X Bus 1, X Bus 2, Y Bus 1, Y Bus 2, D1 Bus, ALU)

DSP Instructions

- X Bus
- X Bus 1:
  - MOV [s], X
- X Bus 2:
  - MOV MUL, P
  - MOV [s], P
- [s] (a data RAM source) must be the same for both groups

- Y Bus
- Y Bus 1:
  - MOV [s], Y
- Y Bus 2:
  - MOV [s], A
  - CLR A
  - MOV ALU, A
- [s] must be the same for both groups

DSP Instructions

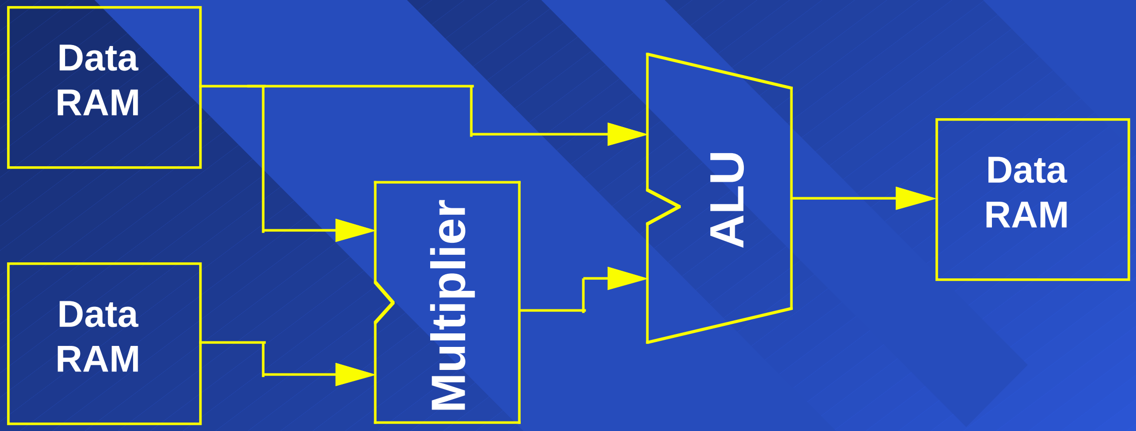
- "Other" commands
  - Load immediate (MVI)
  - DMA
  - JUMP (JMP)
  - LOOP (BTM, LPS)
  - Termination (END, ENDI)

- "Other" commands not parallelisable

- In one instruction, the DSP can
  - Load both sides of the multiplier from data RAM
  - Move the last multiplication result to the ALU
  - Move the last ALU result to its own input register (for MAC operation)
  - Move the last ALU result to data RAM

Data Path

- Typical data path for a program involves two data RAM banks feeding the multiplier, the multiplier feeding the ALU and the ALU result being returned to another data RAM bank



Looping

- 2 types of looping
  - single instruction loops
  - multiple instruction loops

Single instruction loops

- LPS command repeats following instruction LOP + 1 times

```
mov #5, LOP    ; Load the loop counter.
lps            ; Do the next instruction...
mov MC0, MC1   ; ...6 times.
```

Multiple instruction loops

- BTM command jumps to the address held in TOP and decrements LOP, until LOP is zero.

```
       mov #5, LOP           ; Load the loop counter.
       mov loopTop, TOP
loopTop:
       mov M0, MC2           ; looped instruction - done 6 times
       btm                   ; Loop back if LOP > 0, after...
       mov MC0, MC1          ; ...the branch delay slot.
```

Subroutines

- Store 1 in LOP to ensure subsequent BTM works.
- Store address of subroutine in PC to call it
- Branch delay instruction executed once before call and again on return
- Terminate subroutines with BTM
- *BTM has a branch delay slot*

Subroutines

```
        [...]

        mov #1, LOP           ; To ensure BTM returns
        mvi foobar, PC        ; Call the subroutine after storing the PC in TOP
        mvi #10, MC0          ; ...and after branch delay / return instruction.

        [...rest of program...]


;
; A subroutine.
;

foobar:
        mov #11, MC0          ; Body of subroutine.
        BTM                   ; Return...
        mov #12, MC0          ; ...after delay slot is executed
```

Red instruction executed twice, blue instructions form subroutine.

SH2 / DSP Communication

- 2 methods of SH2 / DSP comms.
  - Register based - allows SH2 to directly read/write SCU DSP RAM banks
  - DMA - allows the DSP to transfer data / programs to / from external RAM areas

SCU DSP Registers

- DSP Program Control Port
  - bit 0 ~ bit 7 set program transfer address
  - other bits contain control/status flags
- DSP Program RAM Data Port
  - transfers data to program RAM at above address
  - auto-increments transfer address
- DSP Data RAM Address Port
  - bit 0 ~ bit 7 set address for transfer of data to DSP data RAM.
- DSP Data RAM Data Port
  - Allows access to the data RAM address set above.
  - Auto-increments the transfer address after access

DSP DMA

- Transfers data between DSP memory and external addresses (A Bus, B Bus or Work RAM-H)
- External address set in registers RA0 & WA0
- Transfer specified to / from single RAM bank
- Transfer completion indicated by “T0” flag

DSP DMA

```
; Program to do a little DMA.
; Source: WORK RAM-H.
; Target: DSP Data RAM 0

address = 0x06000000 >> 2    ; Long word address of source.
length  = 32                 ; 32 long words.

       mvi address, RA0      ; Set the read address.
       mov #0, CT0           ; Set the write address.

       dma d0,mc0,length     ; Start the DMA.

dma_wait:
       jmp t0, dma_wait      ; Wait for the DMA to complete.
```

Programming Example

- Simple program to multiply 2 sets of numbers together
- 2 versions - “serial” and “parallel”

Multiplication program - serial

```
; Program to multiply two sets of integers.  Input sets are assumed
; to be in data RAM 0 and RAM 1, results will be put into data RAM 2.

num_integers = 9                    ; Number of multiplication's to do - 1.

       mov #0,CT0                   ; Set index registers to 0.
       mov #0,CT1
       mov #0,CT2
       clr A                        ; Clear the ALU input A.
       mov #num_integers, LOP       ; LOP is the loop counter.
       mov #loopTop, TOP            ; Store top address of loop

loopTop:
       mov MC0, X                   ; Data RAM 0 -> RX, CT0++.
       mov MC1, Y                   ; Data RAM 1 -> RY, CT1++.
       mov MUL, P                   ; Result of multiplication -> P
       btm                          ; Bottom of loop - after delay slot
       or           mov ALL, MC2    ; ( A | P ) -> data RAM 2, CT2++

       end                          ; The End.
```

This takes 57 cycles to exe

Multiplication program - instruction types

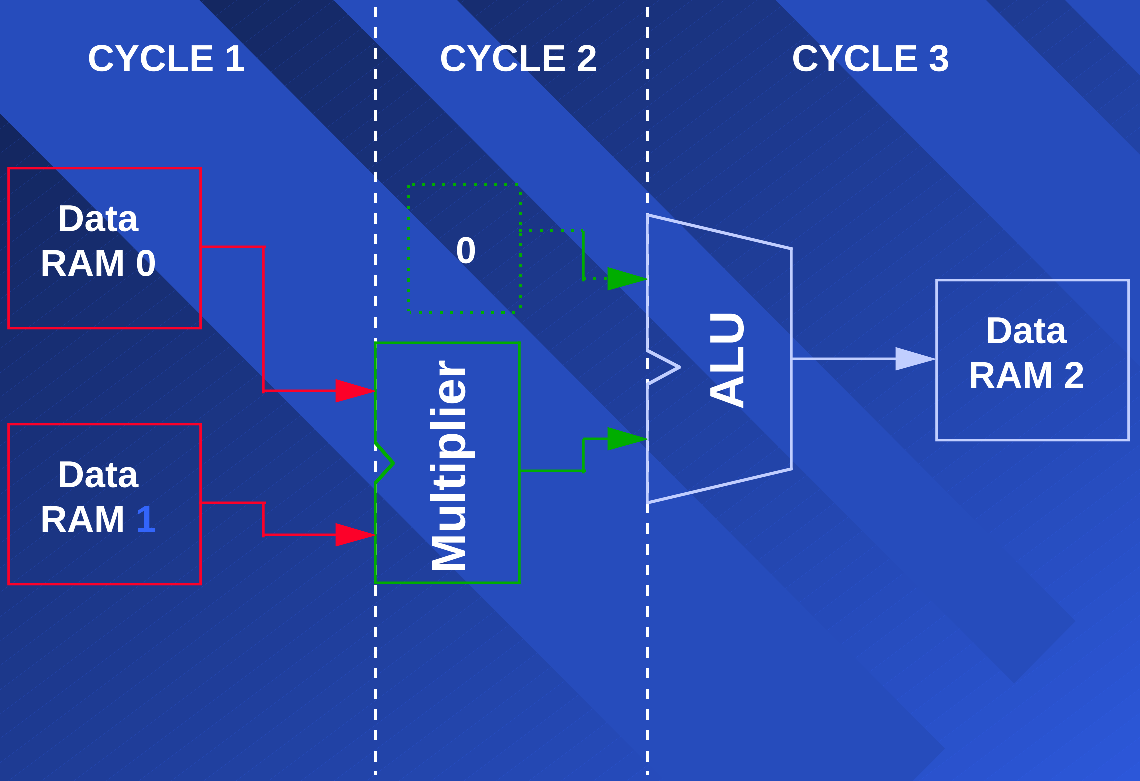
ALU	X Bus	Y Bus	D1 Bus	Other
			mov #0,CT0	
			mov #0,CT1	
			mov #0,CT2	
		clr A		
			mov #9, LOP	
			mov #6, TOP	
	mov MC0, X			
		mov MC1, Y		
	mov MUL, P			
				btm
or			mov ALL,MC2	
				end

Multiplication program parallel

ALU	X Bus	Y Bus	D1 Bus	Other
			mov #0,CT0	
			mov #0,CT1	
		clr A	mov #0,CT2	
			mov #9, LOP	
			mov #5, TOP	
				btm
or	mov MC0,X mov MUL,P	mov MC1, Y	mov ALL,MC2	
				end

This won’t quite work!!!

Multiplication program - data flow



Multiplication program parallel

ALU	X Bus	Y Bus	D1 Bus	Other
			mov #0,CT0	
			mov #0,CT1	
		clr A	mov #0,CT2	
			mov #9, LOP	
	mov MC0, X	mov MC1, Y		
	mov MC0,X mov MUL,P	mov MC1, Y		
				lps
or	mov MC0,X mov MUL,P	mov MC1, Y	mov ALL,MC2	
				end

This takes 18 cycles to execute

Summary

- A powerful, specialised processor
- Performance optimised through parallelism
- DMA provides easy access to external data

The Included Demo

- This sets up two versions of a polygon object and defines the rotating scroll plane.
- In the game loop, it takes joypad info, then creates the camera based on the joypad.
- It then draws the two polygon objects (two pyramids).
- Finally uses the rotating scroll plane as the ground.

(Note the use of slScrMatConv() which allows the demo to manipulate the rotating scroll plane as another polygon object).

Top SGL question (1)

- If you will have polygon objects that move offscreen, you will often find that they show up again in the form of lines across the screen sometimes while off-screen.
- The way to fix this is to add the "UseClip" option in the "option" field of the polygon attributes for the object. This causes the vertices that are offscreen to be clipped, thus no longer drawn.

Top SGL question (2)

- If you are having problems with individual polygons in an object being clipped when they get near the viewscreen, there are two solutions.
- The first is to redefine the view-volume, using slPerspective, slZdspLevel, and slWindow.
- The other is to increase the size of the polygon object by some factor, and then also increase the viewing distances by the same factor. That way, the image on screen is the same, but the object is actually farther away, and thus does not get clipped by the forward viewscreen. In actuality, a combination of these two solutions may provide the best answer.

Top SGL question (3)

- Error in SGL manual: slCheckOnScreen

```
should be   FIXED slCheckOnScreen(pos, size)
                 FIXED pos[XYZ];
                 FIXED size;
```

- To find if an area is on screen, define the position (X,Y,Z) of the location, and then use the SIZE parameter to define the diameter of the volume to check.

Top SGL question (4)

- As of yet, there is no official art cutting tool for converting art files into SGL formats, either for textures or backgrounds. For now, you will have to create your own.

Hitachi SH2 for the GNU 'C' compiler

SoE Internal Product Development

Presentation aims

- To discuss SH2 code and optimisation in context with the GNU ‘C’ compiler as seen from the ‘C’ programmers point of view.
- To introduce some pitfalls that ‘C’ programmers may encounter if the need (or interest) arises to code for the SH2.
- For Saturn ‘C’ programmers SH2 awareness can have significant impact on improving code efficiency.

SH2 Overview I

- 32 bit RISC Processor
- Sixteen 32-bit General Purpose Registers (Rn)
- Three 32-bit Control Registers (SR) (GBR) (VBR)
- Four 32-bit System Registers (PC) (PR) (MACH/MACL)
- Five-stage pipeline
- Instruction and data cache
- Designed for ‘C’ language

Instruction Set

Data:	MOV, MOVA, SWAP, XTRCT
Math:	ADD(C/V), CMP/cond, DIVxx, DT, EXTx
Multiply:	MULS/U, MUL, DMULS/U, MAC
Logic:	NOT, AND, OR, XOR, TST, TAS*
Shifts:	ROT(C)L/R, SHAR/L, SHLR/L 1,2,8,16 bits
Branch:	BF, BT, BF/S, BT/S, BRA(F), BSR(F),JMP, JSR, RTS
System:	RTE, LDC, STC, LDS, STS, CLRMAC, NOP

Address Mode

Direct; Indirect:	Rn; @Rn
Post-inc.; Pre-dec.:	@Rn+; @-Rn
Indirect plus disp.:	@(disp:4,Rn)
Indirect indexed:	@(R0,Rn)
Indirect GBR plus dis.:	@(disp:8,GBR)
Indirect indexed GBR:	@(R0,GBR)
PC relative plus disp.:	@(disp:8,PC)
PC relative:	disp:8; disp:12
Immediate:	#data:8 (extended)

- Displacements multiplied by size of access

Function Interface Optimisation

- ‘GNU “C” parameters use registers R4 - R7 and then stack frame’.
  - Routine can use R0 - R3 as scratch
  - Routine should preserve R8 - R15, GBR, VBR, MACL, MACH
- Carefully select the number of parameters for ‘C’ functions so that register parameters are used where possible, subdividing functions if nesc. The compiler will try to continue to use these registers rather than access the stack.
- If the number of parameters exceeds 4, then an alternative is to group the data components together in a structure and pass a single pointer.

Computation Instruction Reduction

- Instructions required are reduced.

'C' source	Using multiply	Using shift
a*4; or a<<2;	mov.l #_a,r1 mov.b @r1,r1 mov #4,r2 muls r1,r2 sts macl,r0	mov.l #_a,r0 mov.b @r0,r0 shll2 r0

'C' source	Using divide	Using shift
b/2; or b>>1;	mov.l #_b,r0 mov.l @r0,r0 mov #2,r1 mov #_divi,r2 jsr @r2 nop	mov.l #_b,r0 mov.l @r0,r0 shlr r0

Code Optimisation Creating Static Data in C

- ‘C’ source generally compiles into small object, thanks to the SH’s 16-bit instructions (i.e. smaller than x86, 680x0, MIPS, ARM)
- Use mostly local variables and few global variables, because of SH2 addressing modes.
- When using global or static data and variables, try to avoid many independent objects or items. Instead :
  - Group static data into sets of small structures, with less than 16 members
  - Pass data around by pointers
- This will generally result in smaller, faster code.

Code Optimisation/Creating Static Data in C

- Each time the SH2 accesses a global variable,
  a 32-bit absolute address must be loaded into a register (from a literal pool). Only then can the memory location (variable) be accessed.

```
– mov.l    @(disp,PC), r1   ; get addr from literal pool
– mov.l    @r1, r2          ; get global variable in r2
– add      #4, r2           ; global=global+4
– mov.l    r2, @r1          ; store global
```

Optimising/Parameters & Local Variables - C

- C compilers map procedure parameters and local variables to registers.
- When the number of parameters and local variables exceeds the number of allotted registers, some of the variables spill onto the stack.
- At a procedure invocation, register variables of the calling procedure are stacked.
- Within a procedure, operations on local variables are efficient :

- Try to subdivide the algorithm into appropriately sized procedures - small functions are generally better.
- Look at the assembly listing of compiled object from time to time.

Local variables

- Always try to declare tempory variables and loop counters as local variables.

Corresponding ‘C’ source

```
int val;

afunc()
{
    . . .
    val = 1;
    . . .
}
```

Non local

```
mov     #_val,r0
mov     #1,r1
mov     r1,@r0
```

Corresponding ‘C’ source

```
afunc()
{
    int val;

    . . .
    val = 1;
    . . .
}
```

Local (non reg)

```
mov     #1,r0
mov     r0,@r15
```

Local (reg)

```
mov     #1,r8
```

Dealing with Global Variables (Method A)

C Source Code

```
int glob, glob2;

int sub( int x)
{
    glob = x;
    glob2 = x+2;
}
```

Compiler generates the following :

```
        .global _sub
_sub:
        mov.l   L2,r1
        mov.l   r4,@r1
        mov.l   L3,r1
        add     #2,r4
        rts
        mov.l   r4,@rl
        .align 2
L2:
        .long   _glob
L3:
        .long   _glob2
        .comm   _glob, 4
        .comm   _glob2, 4
```

Dealing with Global Variables (Method B)

C Source Code

```
struct Gtype
{
int glob, glob2;
}

sub (struct Gtype *G,
     int x);
{
    G->glob = x;
    G->glob2 = x+2;
}
```

Compiler generates the following :

```
    .global _sub
_sub:
    mov.l       r5,@r4
    add         #2,r5
    rts
    mov.l       r5,@(4,r4)
```

Delay Branch Optimisation

- Utilise delay slots.

Corresponding ‘C’ source	**Before optimisation (With NOP pad)**	**After optimisation (NOP reduction)**
`a = 1;`	`mov #1,r0`	`bsr _dofunc`
`dofunc();`	`bsr _dofunc`	`mov #1,r0`
	`nop`	

Other delay branch instructions are JMP, JSR, BRA, RTS and RTE.

Pipelined Instructions

- Five Stage Pipeline
  - Instruction Fetch (IF)
  - Instruction Decode (ID)
  - Execute (EX)
  - Memory Access (MA)
  - Write Back (WB)

IF	ID	EX	MA	WB				
	IF	ID	EX	MA	WB			
		IF	ID	EX	MA	WB		
			IF	ID	EX	MA	WB	
				IF	ID	EX	MA	WB

Pipeline Stall Example and Optimisation

Corresponding ‘C’ source	Before optimisation (With pipeline wait)	After optimisation (No pipeline wait)
x = a;	mov.l #_a,r0	mov.l #_a,r0
x++;	mov.l @r0,r8	mov.l @r0,r8
y = 2;	add #1,r8	mov #2,r9
	mov #2,r9	add #1,r8

mov.l #_a,r0	IF	ID	EX	MA	WB		
mov.l @r0,r8		IF	ID	EX	MA	WB	
add #1,r8			IF	ID	--	EX	
mov #2,r9							

Instruction Fetch (IF) Instruction Decode (ID)Execute (EX) Memory Access (MA) Write Back (WB)

Formed code improvement - Tail recursion

- Used to eliminate redundant BSR/RTS combinations.

Corresponding 'C' source

```
afunc()
{
    if (x==1)
    {
        func1();
    }
    else
    {
        func2();
    }
```

Before optimisation

```
    cmp/eq    #1,r0
    bf        lab1
    bsr       _func1
    nop
    rts
    nop
lab1:
    bsr       _func2
    nop
    rts
    nop
```

After optimisation

```
cmp/eq    #1,r0
    bf        lab1
    bra       _func1
    nop
lab1:
    bra       _func2
    nop
```

Commonly used Compiler Options

- gcc -v. This will display the compiler version number. You will probably be asked for this during tech support queries.
- -fomit-frame-pointer. Instructs the compiler not to keep the frame pointer in a register for functions that don’t need one. This avoids instructions to save, set-up and restore frame pointers.
- -finline-functions. Instructs the compiler to integrate all simple functions into their callers.

Summary

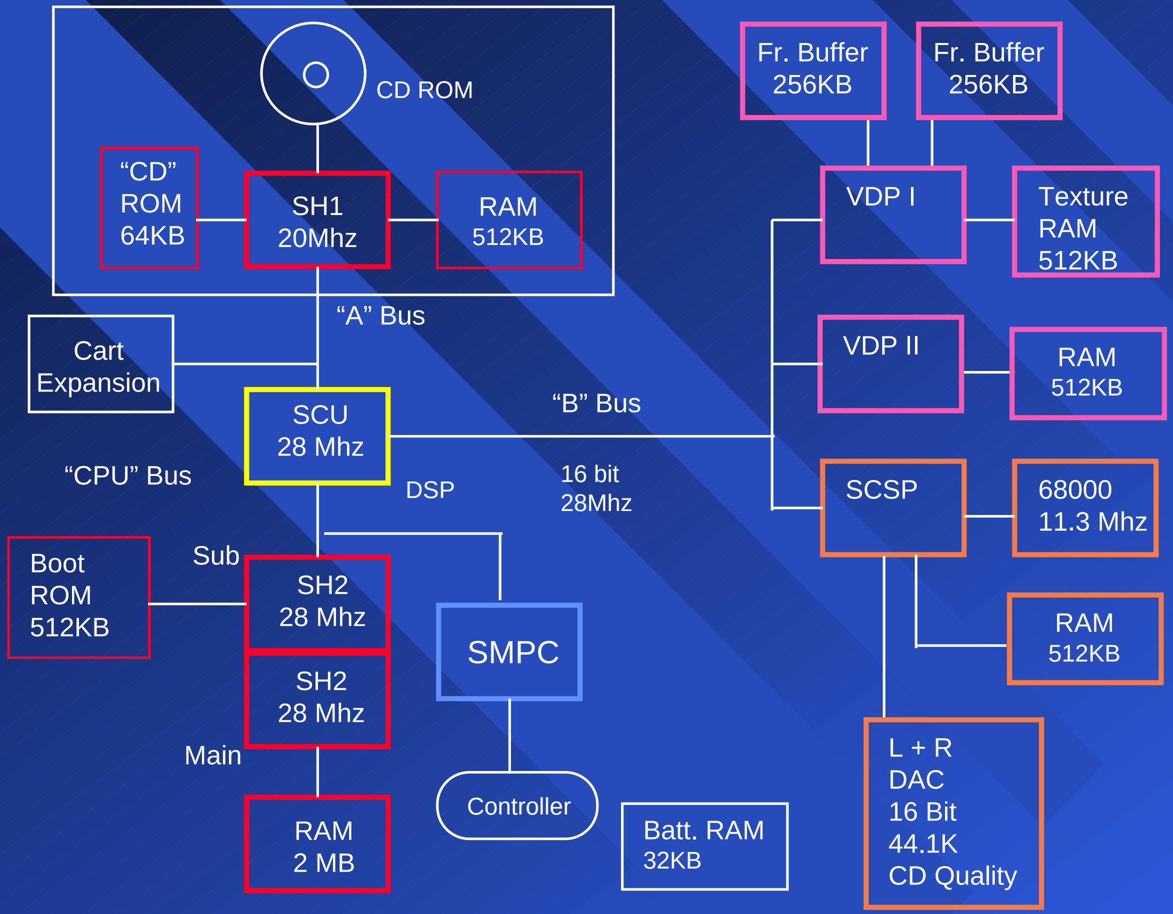
- Use the -S option on the compiler command line to generate SH2 code output from time to time.
- Experiment with the various compiler optimisation options and study the SH2 code result.

The CD SubSystem

Simon Golding

Technical Consultant

Where the CD lives



CD Preparation Software

CD Software Overview

- SEGACDW.EXE
  - Used to burn a W.O.R.M (Write Once Read Many).
- VCDBUILD.EXE [/p]
  - Reads script file, creates an .RTI file used by VCDEMU.EXE.
- VCDEMU.EXE (*JVC VCD system)*
  - Performs CD Emulation on disk image or via the DOS files.
- VCDBUILD.EXE
  - Used to build disk images for Emulation and CD burns.
- VCDUTIL.EXE
  - Allows in-situ replacement of files in a disk image.
- VCDMKTOC.EXE
  - Creates the table of contents (necessary for the CD burn).
- SWAP.EXE
  - Converts a BIG Endian file into LITTLE Endian REDBOOK file.
- TOCCHK.EXE
  - Validates .TOC files for errors.

Creating CD Information Files

- VCDBUILD.EXE [/p]
  - VCDBUILD.EXE [/p] inputs a script file, calculates the correspondence to MS-DOS files and CD access position and creates correspondence tables and other files.
  - Files generated by VCDBUILD.EXE [/p] are read and used by the VCD emulator.
- VCD “Script” files (.scr)
- Specifies how MS-DOS files, i.e. the data and programs will be arranged on the CD.

Example Script File

```
Disc TEST.DSK           ;Name of disk image
Catalog No 0
Session CDROM           ;Type of image

LeadIn MODE1
EndLeadIn

SystemArea IP.BIN       ;File in System Area

Track MODE1

        Volume ISO9660 TEST.PVD
            PrimaryVolume 0:2:16 ;1st file
            EndPrimaryVolume
        EndVolume

;IP.BIN will load the first file alphabetically

    File 1.PRG ;ensures that this is the 1st file
        FileSource c:\demo\main.bin
        EndFileSource
    EndFile

;Place all your DATA files here onwards

    File GRAPHICS.BIN
        FileSource c:\demo\graphics.bin
        EndFileSource
    EndFile
```

```
;-- Setup a file to be placed in a Directory --;

 Directory SOUND        ;Setup Directory name

   File MUSIC.BIN
       FileSource c:\demo\music.bin
       EndFileSource
   EndFile

EndDirectory

   PostGap 150
Endtrack   ;End of the ISO9660 track

;---------- redbook audio tracks -------------;

Track CDDA
  Pause 150

  File CDDA1
      Attributes HIDDEN
      FileSource c:\demo\redbook.red
      EndFileSource

EndFile
EndTrack
LeadOut CDDA
    Empty 300
EndLeadOut
EndSession


EndDisc
```

Creating the CD image file

- VCDBUILD.EXE
  - Inputs a script file
  - CD image and TOC information are created in a file.
  - Files generated by VCDBUILD.EXE are read and used by the VCD emulator and the write-once CD writer.

Building An Actual CD

- These are the steps you need to follow in order to prepare a disk image that can subsequently be used to burn a CD-ROM:
  - » Type “VCDBUILD TEST”
    - This will parse the “TEST.SCR” file and create “TEST.DSK” and “TEST.RTI” where “TEST.DSK” is the disk image file.
  - » Type “VCDMKTOC TEST”
    - This will create “TEST.TOC” from the image file. “TEST.TOC” is used by the ROM burning software.
  - » Type “TOCCHK TEST.TOC”
    - This will check the .TOC file for errors, such as incorrect LeadIns / LeadOuts and lack of a CD-DA track.

Updating the CD Image

- Performed by `VCDUTIL.EXE`
- Real Time Emulation
  - The image file is updated with the new file
  - The file must be the same size or smaller than the original
- CD Image Partial Update
  - Creates a patch file for disk images
  - Saves time over recreating the entire image

VCD
&
Production Saturns

Powering up the VCD

- For those who are using the production Saturn that has been modified for use with the VCD, you will need to do the following whenever you power up the Saturn (You will not need to do this after each subsequent reset):
  - a) Set the VCD selector switch to "CD".
  - b) Power up the Saturn
  - c) Insert the Saturn System-Disc (SegaBrand / 3rd Party)
  - d) Wait for the CD+G ball to become an "S" (CD-ROM mode)
  - e) Select the CD-ROM ball and press A with the joypad
  - f) Wait for the SEGA logo to come up and the word COMPLETED to appear
  - g) Switch the VCD selector to "VCD"
- At this time the user interface will re-appear and the VCD system has control over the CD Block.

Debugging with CartDev & VCD

- Modify script file to include a corrupt `IP.BIN` file in the system area.
  - To corrupt the `IP.BIN` insert random data into the file.
  - A corrupt `IP.BIN` is not needed for a Programming Box.
- Start up the target with the emulator.
- Run SNBUGSAT on the host.
- Download your main module to the host and then run and debug as normal.

A
Saturn
CD

CD Overview

- Standards:
  - ISO9660
  - ISO111702 (MPEG)
- Formats:
  - CD-ROM
  - CD-ROM XA
  - CD-DA
- Capacity:
  - 635 MB
- Time:
  - 63 mins (*this is after Outer-Edge security*)

SEGA Approved CD's

- Manufacturers:
  - – YAMAHA YOD-201
  - – MEGA CD-R
  - – SEGA SATURN CD-R
  - – TDK CD- R71
- Line Speed:
  - – 1.25m/sec instead of 1.20m/sec

(Allows for greater mechanical precision.)

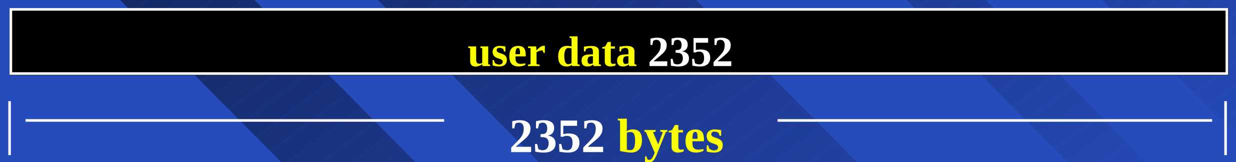
CD Data Layout

- Program and Data files are created as a collection of MS-DOS files
- MODE1 file format: (CD-ROM) (TRACK 1)
  - 2352 bytes per sector
  - 2048 for data
- MODE2 Form 1 file format: (CD-ROM XA) (TRACK 2 - 98)
  - 2352 bytes per sector
  - 2048 for data
- MODE2 Form 2 file format: (CD-ROM XA) (TRACK 2 - 98)
  - 2352 bytes per sector
  - 2324 for data
- CD-DA file format: (CD-DA) (TRACK 2 - 99)
  - Redbook audio tracks (one file per track)
  - MS-DOS file format binary file
  - Intel format (Little endian)
  - Motorola format (Big endian) swap must be implemented in advance

Sector Layout (1)

CD-DA

Data Layout in a CD-Audio Sector (1/75 second)

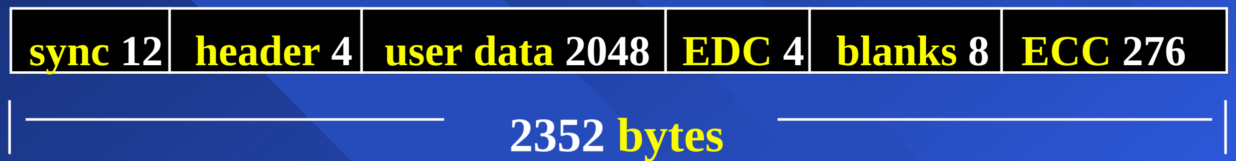


user data 2352

2352 bytes

CD-ROM

2352 byte Layout for CD-ROM Mode 1



sync 12	header 4	user data 2048	EDC 4	blanks 8	ECC 276

2352 bytes

Sector Layout (2)

CD-ROM/XA

2352 byte Layout for CD-ROM Mode 2 / XA Form 1

sync 12	header 4	sub-head 8	user data 2048	EDC 4	ECC 276

2352 bytes

2352 byte Layout for CD-ROM Mode 2 / XA Form 2

sync 12	header 4	sub-head 8	user data 2324	EDC 4

2352 bytes

CD Security

CD Security Overview

- The Outer-Edge Security Code is burnt on to the CD-ROM during the manufacturing process.

- A System Disc must be acquired through SEGA for either SEGABRAND or 3RD PARTY before any Saturn CD-ROM data will run on a production model.

- SEGABRAND <Red Label System Disc>
  - Will only activate SegaBrand titles.

- 3RD PARTY <Black Labe System Disc>
  - Will only activate 3rd Party titles.

IP.BIN

System ID Stencil

SYSTEM ID

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00h	Hardware Indentifier															
10h	Marker ID															
20h	Product Number										Version					
30h	Release Date								Device Information							
40h	Compatible Area Symbols										Space					
50h	Compatible Peripherals															
60h	Game Title															
70h																
80h																
90h																
A0h																
B0h																
C0h																
D0h	Reserved															
E0h	IP size				Reserved				Stack - M				Stack - S			
F0h	1st Read Addr.				1st Read Size				Reserved				Reserved			

System ID (SegaBrand) Example

SYSTEM ID

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00h	S	E	G	A		S	E	G	A	S	A	T	U	R	N	
10h	S	E	G	A		E	N	T	E	R	P	R	I	S	E	S
20h	G	S	-	9	0	9	9				V	1	.	0	0	0
30h	1	9	9	5	0	1	1	8	C	D	-	1	/	1		
40h	J										Space					
50h	J															
60h	S	E	G	A		G	A	M	E							
70h																
80h																
90h																
A0h																
B0h																
C0h																
D0h	Reserved															
E0h	00001000h				Reserved				06001FFCh				06000FFCh			
F0h	06010000h				IGNORED				Reserved				Reserved			

System ID (3rd Party) Example

SYSTEM ID

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00h	S	E	G	A		S	E	G	A	S	A	T	U	R	N	
10h	S	E	G	A		T	P		K	A	I	S	H	A	-	A
20h	T	-	9	9	9	0	1	G			V	1	.	0	0	0
30h	1	9	9	5	0	1	1	8	C	D	-	1	/	1		
40h	E	U									Space					
50h	J	M														
60h	S	E	G	A		G	A	M	E							
70h																
80h																
90h																
A0h																
B0h																
C0h																
D0h	Reserved															
E0h	00001000h				Reserved				06001FFCh				06000FFCh			
F0h	06010000h				IGNORED				Reserved				Reserved			

Territory / Peripheral Codes

Territory Codes

"J"	JAPAN
"T"	ASIA -NTSC- (TAIWAN, PHILIPPENES)
"U"	NORTH AMERICA (USA, CANADA)
"B"	SOUTH AMERICA -NTSC- (BRAZIL)
"K"	KOREA
"A"	EAST ASIA -PAL- (CHINA, MIDDLE EAST)
"E"	EUROPE -PAL-
"L"	SOUTH AMERICA -PAL-

Peripheral Codes

"J"	CONTROL PAD
"A"	ANALOG CONTROLLER
"M"	MOUSE
"K"	KEYBOARD
"S"	STEERING WHEEL
"T"	MULTI-TAP

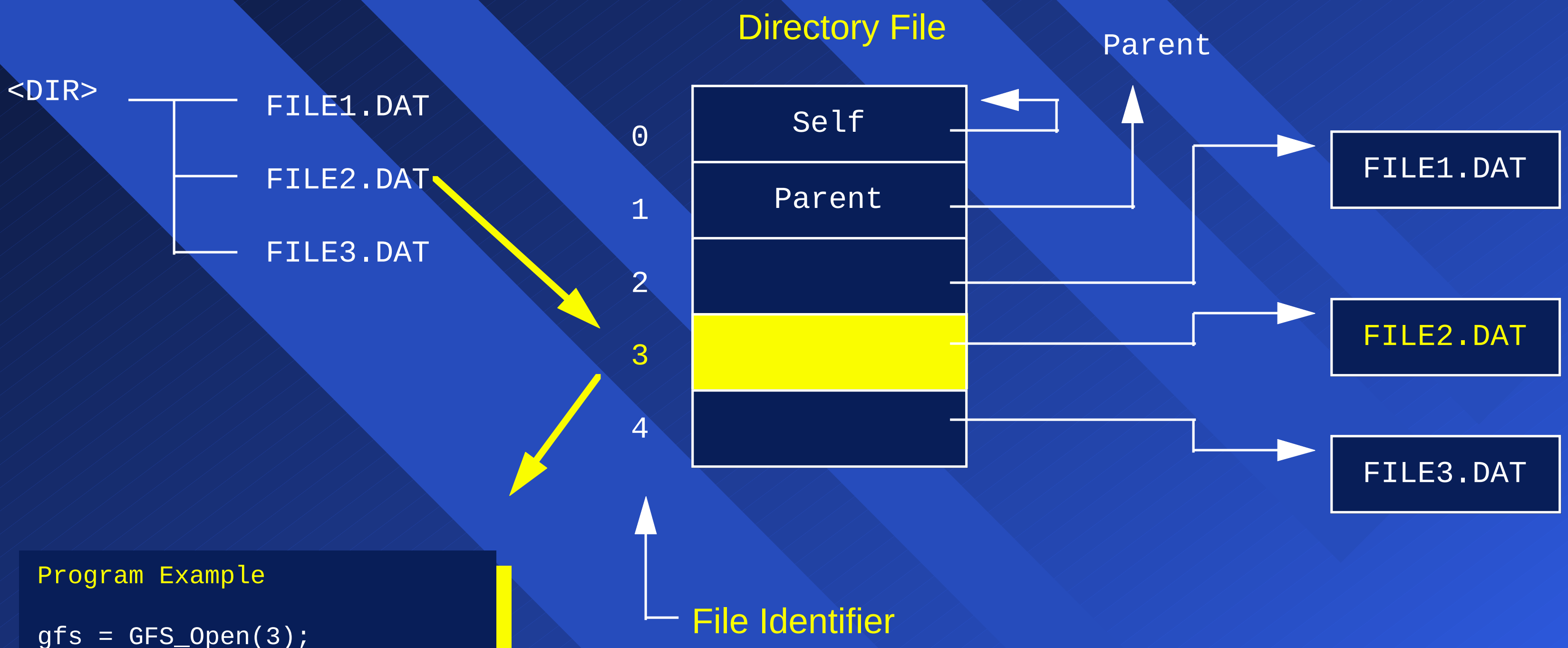
(Number of characters will be increased as new peripherals are introduced.)

CD Libraries

File System Library Overview

- File Access Format
  - Supports ISO9660 Level File Access.
- Data Buffering
  - Accessible if CD block buffer control mechanism is present.
  - Besides reading files, the buffer can be used for pre-reading sectors.
- File Identifiers
  - Access is based on the file identifier (sequence number within the directory).
  - Faster directory searches
  - File name -> file identifier conversion function is supported, but can be removed after the file configuration on the CD is confirmed.

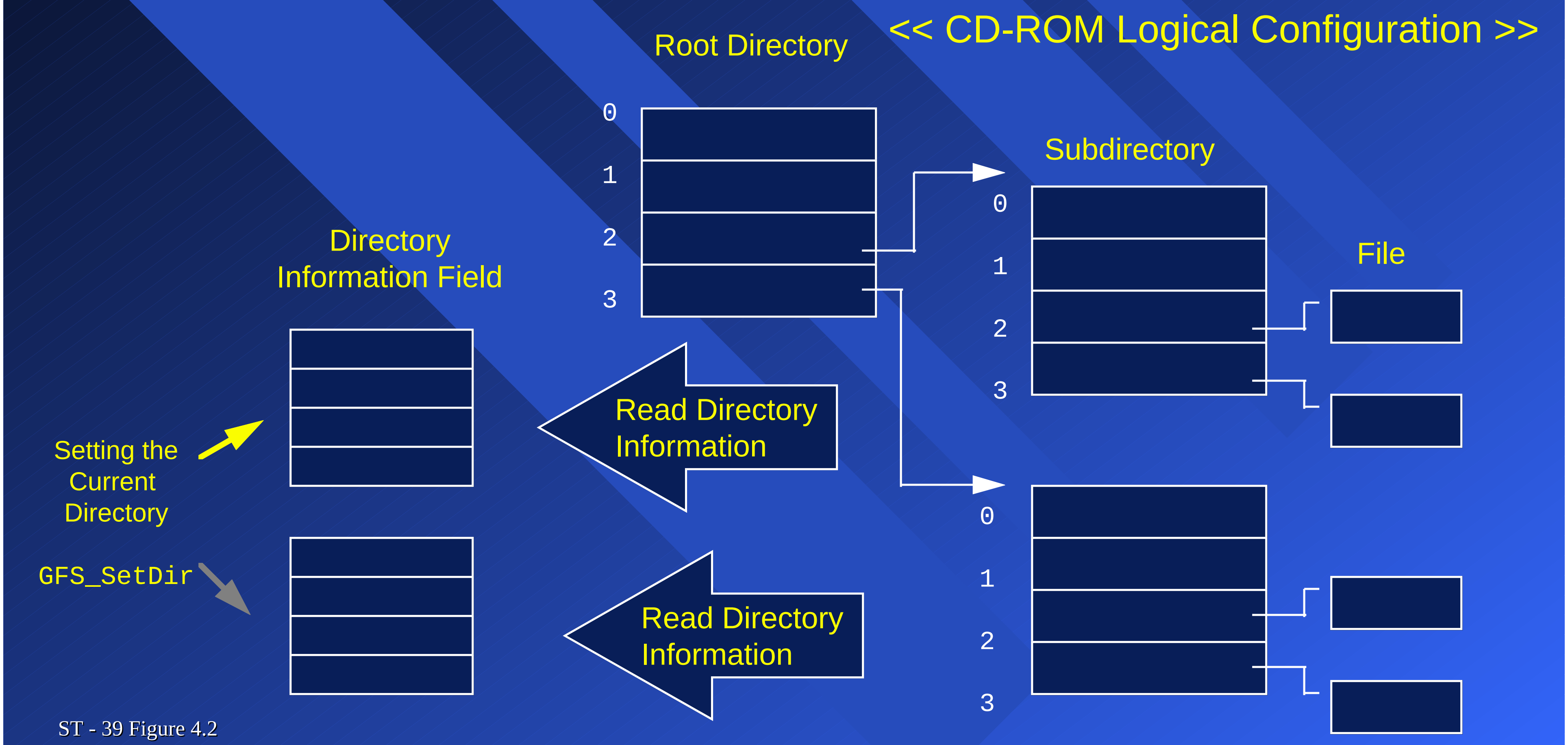
File Access Overview



Program Example

```
gfs = GFS_Open(3);
/*
*  File access is done here
*/
GFS_Close(gfs);
```

Setting the Current Directory (GFS_SetDir)



ST - 39 Figure 4.2

File Access Procedure

- Read the directory information
  - This can be done once at the beginning of the Application.
- Set the current directory
  - Load in the directory structure the Application wishes to access.
- Open the file
  - Open the file either by ID or by NAME.
- Access the file
  - Read in the data.
- Close the file
  - End.

Reading Batch Files - GFS_Load

```
Sint32 fid;                /* file identifier */
Uint8  buf[BUF_SIZE];      /* read area */

/* Load upto size of buffer from file at sector 1 */
fsize = GFS_Load(fid, 0, buf, BUF_SIZE);

/* Load entire file */
fsize = GFS_Load(fid, 0, adr, GFS_BUFSIZ_INF );
```

offset

- Data is read from the file fid sector 0 to `buf[BUF_SIZE]`.
- When this function is finished, `fsize` byte data can be read in `buf[]`.

Example: Accessing by Name

```
GfsDirTbl dirtbl:                       /* Area containing directory information */
GfsDirName dirname[MAX_DIR];            /* Area containing directory information */
GfsHn gfs;                              /* File handle of the access file */
Sint32 fid;                             /* The access file identifier goes here *
Sint8 *fname[] = “level4.bin”;

GFS_DIRTBL_TYPE(&dirtbl) = GFS_DIR_NAME; /* Tell GFS system that we are using names */
GFS_DIRTBL_NDIR(&dirtbl) = MAX_DIR;
GFS_DIRTBL_DIRNAME(&dirtbl) = dirname;   /* Tell GFS where the name table is */

fid = GFS_NameToId( (Sint8 *)”LEVEL4” );
GFS_LoadDir(fid, &dirtbl);               /* Read directory information */

GFS_SetDir(&dirtbl);                     /* Set current directory */

/* Set the identifier of the file to access fid */
gfs = openFileByName(fname);             /* Uses GFS_NameToId to get id and opens */

* File is accessed here
*/
GFS_Close(gfs);
```

High Level Access

- Completion Return Access
  - Functions return only after data has been read from CD
  - Useful for bulk file loads
  - Simple DOS-like file commands

- Immediate Return Access
  - Functions return control to program to allow simultaneous reading / processing
  - Server-like in nature - multiple files may be open

Completion Return Access

```
GfsHn gfs;                                 /* File Handle */
GfsFid fid;                                /* File Identifier */
Sint32 nsct;                               /* Read sector count */
Uint8 buf[BUF_SIZE];                       /* Read Area */

gfs = GFS_Open(fid);                       /* Open file */

GFS_Fread(gfs, nsct, buf, BUF_SIZE);       /* read nsct sector to buf */

GFS_Close(gfs);                            /* close file */
```

Immediate Return Access

```
gfs = GFS_Open(fid);                       /* open file */
GFS_NwFread(gfs, nsct, buf, BUF_SIZE);     /* read nsct sector to buf */
                                          /* return immediately */

for(;;)
{
  stat = GFS_NwExecOne(gfs);                /* execute reading */
  if(stat == GFS_EXEC_COMPLETED)            /* end reading  ? */
  {
    break;
  }
  user();                                  /* optional user process */
}
GFS_Close(gfs);                            /* close file */
```

Multiple File Access

- An “access server” is provided for application processing of files while multiple files are being read.
- Access operations are executed one after another by periodically transferring control to the server.

Pre-reading Multiple Files to the CD Buffer

- Multiple files can be read to the CD buffer simultaneous with other processes.
- By reading in advance, data can be retrieved when specifically needed.