

Chapter 4 - VideoChip

VideoChip is the IE's general-purpose framebuffer chip. It sits at the bottom of the compositor stack (layer 0) and provides three things: a framebuffer in main memory, a programmable raster engine called the **copper list**, and a 2D drawing accelerator called the **blitter**. From BASIC, use POKE32 for raw register programming and the BLIT and COPPER commands for the common accelerated paths. From machine code, ordinary stores to the same MMIO registers do the same work.

4.1 Where the picture lives

VideoChip reads its picture from a region of main memory called **VRAM**. VRAM is a five-megabyte region that begins at byte address \$00100000 (1 MB) and ends at \$005FFFFFF. Any byte in that region is visible to the chip on the next frame.

```
+-----+
| VRAM   |
| $00100000 .. $005FFFFFF |
| 5 MB   |
+-----+
```

Five megabytes is enough to hold several frames at the smaller modes (a 960 × 540 RGBA frame is 2,073,600 bytes, so two such frames fit with room to spare), but **not** enough for one full 1920 × 1080 RGBA frame, which requires 8,294,400 bytes. In the 1920 × 1080 mode, point FB_BASE at a region of ordinary main memory large enough for one frame, instead of into VRAM.

The chip does not own VRAM exclusively. Any CPU can read and write the same bytes. When the compositor builds the next frame, it picks up whatever was last written.

4.2 The register block

Every VideoChip register is a 32-bit word at a fixed address. The block begins at \$000F0000 and ends at \$000F049B. All registers are aligned to 4 bytes.

Address	Name	Purpose
\$F0000	VIDEO_CTRL	Master enable. 0 = off, non-zero = on.
\$F0004	VIDEO_MODE	Output mode (table below).
\$F0008	VIDEO_STATUS	Status bits (read-only).
\$F000C	COPPER_CTRL	Copper enable and reset.
\$F0010	COPPER_PTR	Base address of copper program in main memory.
\$F0014	COPPER_PC	Copper program counter (read-only).
\$F0018	COPPER_STATUS	Copper status bits (read-only).
\$F001C	BLT_CTRL	Blitter start and IRQ enable.
\$F0020	BLT_OP	Blitter operation.
\$F0024	BLT_SRC	Blitter source address.

Address	Name	Purpose
\$F0028	BLT_DST	Blitter destination address.
\$F002C	BLT_WIDTH	Blitter rectangle width in pixels, or byte count for MEMCOPY.
\$F0030	BLT_HEIGHT	Blitter rectangle height in rows.
\$F0034	BLT_SRC_STRIDE	Source stride in bytes per row.
\$F0038	BLT_DST_STRIDE	Destination stride in bytes per row.
\$F003C	BLT_COLOR	Fill colour or line colour.
\$F0040	BLT_MASK	Per-pixel mask for masked copies.
\$F0044	BLT_STATUS	Blitter status bits (read-only).
\$F0048	VIDEO_RASTER_Y	First scanline for the raster band.
\$F004C	VIDEO_RASTER_HEIGHT	Raster band height.
\$F0050	VIDEO_RASTER_COLOR	Raster band fill colour.
\$F0054	VIDEO_RASTER_CTRL	Raster band start (write 1).
\$F0058	BLT_MODE7_U0	Mode 7 texture origin (U).
\$F005C	BLT_MODE7_V0	Mode 7 texture origin (V).
\$F0060	BLT_MODE7_DU_COL	Mode 7 U step per column.
\$F0064	BLT_MODE7_DV_COL	Mode 7 V step per column.
\$F0068	BLT_MODE7_DU_ROW	Mode 7 U step per row.
\$F006C	BLT_MODE7_DV_ROW	Mode 7 V step per row.
\$F0070	BLT_MODE7_TEX_W	Mode 7 U wrap mask (<code>width - 1</code>).
\$F0074	BLT_MODE7_TEX_H	Mode 7 V wrap mask (<code>height - 1</code>).
\$F0078	PAL_INDEX	Auto-incrementing palette write index (0-255).
\$F007C	PAL_DATA	Palette data port. Writes entry, advances PAL_INDEX.
\$F0080	COLOR_MODE	0 = RGBA32 direct, non-zero = CLUT8 indexed.
\$F0084	FB_BASE	Byte address in main memory where the framebuffer begins.
\$F0088-\$F0487	PAL_TABLE	Direct palette window: 256 entries of 4 bytes each.
\$F0488	BLT_FLAGS	Blitter BPP and draw-mode flags.
\$F048C	BLT_FG	Blitter foreground colour for colour-expand.
\$F0490	BLT_BG	Blitter background colour for colour-expand.
\$F0494	BLT_MASK_MOD	Mask-row stride for colour-expand.
\$F0498	BLT_MASK_SRCX	Mask source X offset.

Most setup registers read back the value last staged by the program. Status registers report live state. BLT_STATUS is the exception to the read-only rule: writing bit 2 clears a pending blitter interrupt.

4.3 Enabling the chip and choosing a mode

VideoChip starts disabled. To turn it on, write a non-zero value to VIDEO_CTRL. To turn it off, write 0.

The chip supports eight output modes. Each mode selects a frame width and height. The default mode at power-on is \$07 (960 × 540).

Value	Mode
\$00	640 × 480
\$01	800 × 600
\$02	1024 × 768
\$03	1280 × 960
\$04	320 × 200
\$05	320 × 240
\$06	1920 × 1080
\$07	960 × 540

Write the desired value into VIDEO_MODE. The chip resizes its internal framebuffer and notifies the compositor. Writing a value outside the table is ignored.

Below is a BASIC fragment that turns the chip on at 960 × 540 and sets the framebuffer base at the start of VRAM:

```
10 POKE32 &H000F0004, &H07           : REM VIDEO_MODE = 960 BY 540
20 POKE32 &H000F0084, &H00100000     : REM FB_BASE = VRAM start
30 POKE32 &H000F0000, 1              : REM VIDEO_CTRL = on
```

POKE32 writes the four bytes of a 32-bit word at the given aligned address (see Chapter 2).

4.4 The framebuffer

FB_BASE is the byte address in main memory where the chip reads the first pixel of the picture. The chip reads $W \times H$ pixels in left-to-right, top-to-bottom order, where W and H are the width and height of the current mode.

There are two pixel formats, selected by COLOR_MODE:

COLOR_MODE	Pixel format	Bytes per pixel
0	RGBA32 direct colour	4
non-zero	CLUT8 indexed colour	1

In **RGBA32** mode every pixel is four bytes, little-endian, in the order red, green, blue, alpha. The compositor uses the alpha byte for mask blending as described in Chapter 3.

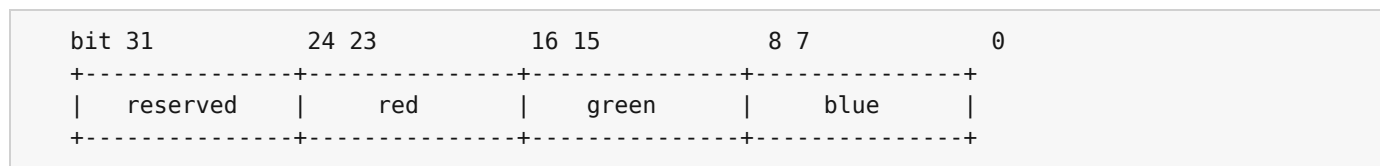
When you write an RGBA32 pixel with BASIC POKE32, remember that POKE32 writes a 32-bit little-endian word: alpha in the high byte, then blue, green, and red in the low byte. BASIC examples often use values below &H01000000, such as &H000000FF for red, because those integers are exact in BASIC's numeric format. The compositor still treats any non-black direct-colour pixel as opaque.

In **CLUT8** mode every pixel is one byte, which is an index into the 256-entry CLUT palette. The chip expands each index into an RGBA32 pixel on the fly. The alpha byte produced by the expansion is always \$FF (fully opaque).

FB_BASE may be set anywhere in main memory. There is no requirement to point it at the VRAM window; any address that holds $W \times H \times \text{bytes-per-pixel}$ bytes of valid data works. Pointing it into VRAM means writes through ordinary memory stores update the picture immediately.

4.5 The palette

The palette has 256 entries. Each entry is a 24-bit RGB triple packed in the low three bytes of a 32-bit word:



There are two ways to set palette entries.

Indexed port. Write the entry number (0-255) to PAL_INDEX. Then write each entry's 24-bit value to PAL_DATA. The chip stores the value, increments PAL_INDEX, and wraps at 256. This is the shortest way to load a whole palette:

```
100 POKE32 &H000F0078, 0      : REM PAL_INDEX = 0
110 POKE32 &H000F007C, &HFF0000 : REM entry 0 = red
120 POKE32 &H000F007C, &H00FF00 : REM entry 1 = green
130 POKE32 &H000F007C, &H0000FF : REM entry 2 = blue
```

Direct window. Each entry is also visible at PAL_TABLE + index*4. Writing the word at that address updates entry index and leaves PAL_INDEX unchanged. Use this when you want to set a single entry without disturbing a write cursor.

The palette is shared between CLUT8 picture display and the blitter's colour-expand operation.

4.6 The blitter

The framebuffer and palette are enough for pixels you place yourself. The blitter is for work you would rather not do one pixel at a time: moving rectangles, building sprites from masks, expanding 1-bit templates, scaling pictures, drawing lines, and treating a texture as a rotating floor. Think of this section as the VideoChip's workbench. The tables give the exact knobs; each example shows the visible job those knobs do.

The blitter is VideoChip's 2D drawing engine. It can copy memory, fill rectangles, draw lines, apply raster operations, expand 1-bit templates, alpha-blend sprites, scale bitmaps, and run the Mode 7 affine texture unit. BASIC exposes the common copy, fill, line, memcpy, wait, and Mode 7 paths through BLIT. The other operations are still native to the machine: enter them with POKE32 to the blitter registers.

The blitter is synchronous. Starting it with BLT_CTRL .START runs the operation before the POKE32 or machine-code store returns. A program may still poll BLT_STATUS or use BLIT WAIT; in normal BASIC use the operation has already finished.

4.6.1 Blitter registers and start sequence

Every blitter operation uses the same staged-register model. Writes to BLT_SRC, BLT_DST, BLT_WIDTH, and the other setup registers prepare the next operation. Writing 1 to BLT_CTRL latches those values and starts the transfer.

Register	Used for
BLT_CTRL	Bit 0 starts the operation. Bit 2 enables blitter IRQs. Reads return BUSY in bit 1 and IRQ-enable in bit 2.
BLT_OP	Operation number, 0-8.

Register	Used for
BLT_SRC	Source address, or packed start coordinate for legacy line drawing.
BLT_DST	Destination address, or packed end coordinate for legacy line drawing.
BLT_WIDTH	Width in pixels for rectangles, source width for scale, or packed end coordinate for extended line drawing.
BLT_HEIGHT	Height in pixels for rectangles, source height for scale.
BLT_SRC_STRIDE	Source bytes per row. 0 means a default stride.
BLT_DST_STRIDE	Destination bytes per row. 0 means a default stride.
BLT_COLOR	Fill/line colour, or packed scale destination size.
BLT_MASK	Mask/template address.
BLT_FLAGS	Pixel size, draw mode, and colour-expand flags.
BLT_FG, BLT_BG	Foreground and background colours for colour-expand.
BLT_MASK_MOD	Template bytes to advance after each colour-expand row.
BLT_MASK_SRCX	First template bit X offset for colour-expand.

General raw-register sequence:

1. Write BLT_OP.
2. Write operation-specific setup registers.
3. Write BLT_FLAGS if the operation needs CLUT8, draw modes, or colour-expand flags. Leave it at 0 for the default RGBA32 copy mode.
4. Write 1 to BLT_CTRL.
5. Read BLT_STATUS if you need to check DONE, ERR, or IRQ_PENDING.

BLT_SRC_STRIDE and BLT_DST_STRIDE are bytes, not pixels. If a stride register is 0, the blitter chooses a default. For VRAM destinations the default is the current framebuffer row width (mode width * bytes-per-pixel). For non-VRAM bitmaps the default is the rectangle width times bytes per pixel.

4.6.2 Operations

BLT_OP	Operation	Reader path	Summary
0	COPY	BLIT_COPY or POKE32	Copy a rectangle.
1	FILL	BLIT_FILL or POKE32	Fill a rectangle with BLT_COLOR.
2	LINE	BLIT_LINE or POKE32	Draw a Bresenham line.
3	MASKED_COPY	POKE32	Copy RGBA32 source pixels where a 1-bit mask is set.
4	ALPHA_COPY	POKE32	Copy or blend RGBA32 source pixels using source alpha.
5	MODE7	BLIT_MODE7 or POKE32	Affine RGBA32 texture mapper.
6	COLOR_EXPAND	POKE32	Expand a 1-bit template into CLUT8 or RGBA32 pixels.
7	SCALE	POKE32	Nearest-neighbour scale from source rectangle to destination rectangle.
8	MEMCOPY	BLIT_MEMCOPY or BLIT_M	Copy a byte-counted linear memory span.

4.6.3 BLT_FLAGS

BLT_FLAGS controls bytes per pixel and raster operation for the copy, fill, scale, colour-expand, and extended line paths.

Bits	Field	Meaning
1-0	BPP	0 = RGBA32, 1 = CLUT8. Other values currently behave as RGBA32.
7-4	Draw mode	Raster operation number, table below.
8	JAM1	Colour-expand: clear template bits leave destination unchanged.
9	Invert template	Colour-expand: flip each template bit before use.
10	Invert mode	Colour-expand: set template bits XOR the destination with all ones.
11	Mask MSB-first	Masked-copy: sample mask bits MSB-first (Amiga PLANEPTR order) instead of the default LSB-first.
12	Alpha template	Alpha-copy: treat the source as an 8bpp alpha plane blended over the destination with BLT_FG.

If BLT_FLAGS is 0, the draw mode is treated as COPY (3) for compatibility. Otherwise, the draw-mode field is used exactly. Note the consequence: because 0 is the RGBA32-Copy sentinel, the Clear draw mode (0) cannot be expressed in RGBA32 with no other flags set; fill with colour 0 instead.

RGBA32 raster-op caveat. The draw modes operate on the full 32-bit pixel, *including the alpha byte*. In RGBA32 only Copy (3), Clear (0), and Set (15) yield colours that round-trip cleanly; the bitwise modes (And/Or/Xor/...) mangle alpha and can produce off-palette values. Use the non-Copy raster ops in CLUT8 mode, where operands are 8-bit indices.

Big-endian colour contract. When the chip runs in big-endian mode (M68K guests), BLT_COLOR/BLT_FG/BLT_BG are still consumed and stored little-endian, whereas a CPU-direct store lands big-endian. To make a blitter-rendered RGBA32 pixel match a CPU-rendered one, byte-reverse the colour before writing it (0xRRGGBBAA → 0xAABBGGRR). CLUT8 indices occupy the low byte and are never swapped.

Mode	Name	Result
0	Clear	0
1	And	source AND destination
2	AndReverse	source AND NOT destination
3	Copy	source
4	AndInverted	NOT source AND destination
5	NoOp	destination
6	Xor	source XOR destination
7	Or	source OR destination
8	Nor	NOT (source OR destination)
9	Equiv	NOT (source XOR destination)
10	Invert	NOT destination
11	OrReverse	source OR NOT destination
12	CopyInverted	NOT source
13	OrInverted	NOT source OR destination
14	Nand	NOT (source AND destination)

Mode	Name	Result
15	Set	all bits set

To build a flags word by hand:

```
flags = bpp + drawmode * 16
```

For example, RGBA32 XOR is $0 + 6 * 16 = 96$. CLUT8 copy is $1 + 3 * 16 = 49$.

4.6.4 Status, errors, and IRQs

BLT_STATUS carries three bits:

Bit	Name	Meaning
0	ERR	Last operation failed or detected an invalid setup.
1	DONE	Last operation completed.
2	IRQ_PENDING	Blitter interrupt is pending.

Starting a blit clears ERR and DONE, then sets DONE after the operation finishes. Invalid operation numbers set ERR. Scale sets ERR for zero sizes, out-of-range rectangles, and visible RGBA32 VRAM writes that are not 4-byte aligned. Mode 7 sets ERR for invalid texture masks or out-of-range texture reads. Rectangle copy and fill also set ERR when a visible RGBA32 VRAM destination is misaligned or beyond the displayed framebuffer. Any blitter operation that would write outside backed writable guest memory sets ERR instead of wrapping into another address range.

To request an interrupt, write $BLT_CTRL = 5$ (START + IRQ_ENABLE) instead of 1. The operation still completes synchronously, and BLT_STATUS.IRQ_PENDING is set. Clear the pending bit by writing 4 to BLT_STATUS.

4.6.5 COPY and MEMCOPY

COPY transfers a rectangle from BLT_SRC to BLT_DST.

Register	Meaning
BLT_SRC	Source base address.
BLT_DST	Destination base address.
BLT_WIDTH	Width in pixels for RGBA32/CLUT8 rectangles.
BLT_HEIGHT	Height in rows.
BLT_SRC_STRIDE	Source bytes per row.
BLT_DST_STRIDE	Destination bytes per row.
BLT_FLAGS	Pixel size and draw mode.

BLIT COPY src, dst, w, h uses operation 0. Add both strides when the source or destination is a small off-screen bitmap rather than a full framebuffer row:

```
BLIT COPY src, dst, w, h, srcStride, dstStride
```

BLIT MEMCOPY src, dst, len and its shorthand BLIT M src, dst, len select operation 8. This is not a rectangle copy. It copies len bytes from src to dst as one linear span. BLT_SRC, BLT_DST, and BLT_WIDTH are the registers that matter.

BLT_HEIGHT, BLT_SRC_STRIDE, BLT_DST_STRIDE, and BLT_FLAGS are ignored by the hardware operation, although the BASIC keyword stores 1 in BLT_HEIGHT and 1 in both stride registers before it starts the blitter.

Use COPY when the source is a rectangle with a row stride. Use MEMCOPY when the source is a flat block, such as a saved RGBA screen, a work buffer, or a full back buffer. The length is always bytes. A 640 by 480 RGBA32 screen is therefore 640*480*4, or 1228800 bytes.

MEMCOPY back buffer

This listing fills an off-screen buffer, copies it into the visible framebuffer, then prints the blitter status:

```
10 FB=&H00100000:BB=&H00230000:ST=640*4
20 POKE32 &H000F0004,0:POKE32 &H000F0080,0:POKE32 &H000F0000,1
30 BLIT FILL BB,640,480,&H00002040,ST
40 BLIT MEMCOPY BB,FB,640*480*4
50 PRINT PEEK32(&H000F0044)
```

The screen shows the filled buffer after line 40. The status print is 2, meaning DONE is set and ERR is clear.

Overlapping RAM and VRAM copies preserve the original source bytes in the normal memory paths. For portable programs, still prefer non-overlapping source and destination spans unless you are deliberately moving a buffer in place.

4.6.6 FILL

FILL writes BLT_COLOR over a destination rectangle. In RGBA32 mode the whole word is written. In CLUT8 mode only the low byte is used.

Register	Meaning
BLT_DST	Destination base address.
BLT_WIDTH	Width in pixels.
BLT_HEIGHT	Height in rows.
BLT_DST_STRIDE	Destination bytes per row.
BLT_COLOR	Fill value.
BLT_FLAGS	Pixel size and draw mode.

BLIT FILL dst,w,h,colour[,stride] is the BASIC form. If the draw mode is not COPY, the fill value is combined with the destination through the selected raster operation.

4.6.7 LINE

LINE draws a Bresenham line.

The BASIC form is:

```
BLIT LINE x0, y0, x1, y1, colour
```

This is the legacy line path. BLT_SRC packs the start coordinate as $x0 + y0 * 65536$; BLT_DST packs the end coordinate the same way. The destination base is fixed at VRAM start, and the line is clipped to the current VideoChip mode.

For a custom bitmap base, CLUT8 lines, or draw modes, use extended line mode by setting BLT_FLAGS non-zero:

Register	Meaning in extended line mode
BLT_SRC	$x0 + y0 * 65536$.
BLT_WIDTH	$x1 + y1 * 65536$.
BLT_DST	Destination bitmap base address.
BLT_DST_STRIDE	Destination row stride in bytes.
BLT_COLOR	Line colour or CLUT8 index.
BLT_FLAGS	BPP and draw mode. Must be non-zero to select extended line mode.

Extended line mode does not clip to a viewport. Clip coordinates before starting the blit.

4.6.8 MASKED_COPY

MASKED_COPY is an RGBA32 sprite copy controlled by a 1-bit mask. It uses three mask-related fields: BLT_MASK, BLT_MASK_MOD, and BLT_MASK_SRCX. BLT_FLAGS bit 11 changes the bit order.

Register	Meaning
BLT_SRC	RGBA32 source base.
BLT_DST	RGBA32 destination base.
BLT_WIDTH, BLT_HEIGHT	Rectangle size in pixels.
BLT_SRC_STRIDE, BLT_DST_STRIDE	Source and destination row strides.
BLT_MASK	1-bit mask base address.
BLT_MASK_MOD	Mask bytes to advance after each mask row; 0 means packed rows.
BLT_MASK_SRCX	First bit offset in the mask row.
BLT_FLAGS bit 11	0 = LSB-first mask bits, 1 = MSB-first mask bits.

By default, mask bits are packed least-significant-bit first. Pixel x reads bit $((BLT_MASK_SRCX + x) \text{ MOD } 8)$ from byte $BLT_MASK + (BLT_MASK_SRCX + x) / 8$. A set bit copies the source pixel. A clear bit leaves the destination unchanged. If BLT_MASK_MOD is 0, the mask row stride is $(width + 7) / 8$ bytes. If it is non-zero, that value is used as the mask row stride.

Set BLT_FLAGS bit 11 for most-significant-bit first masks. In that form, bit 7 of the first mask byte is the leftmost pixel, then bit 6, and so on. This is useful when the mask data was prepared in the same order as a 1-bit display plane.

This example draws a small diamond sprite. The source is an ordinary 8 x 8 RGBA32 square, but the mask lets only the diamond-shaped part through:

```

10 REM MASKED COPY DIAMOND
20 FB=&H00100000:SPR=&H00600000:MSK=&H00610000
30 ST=320*4:SS=8*4
40 POKE32 &H00F0004,&H04
50 POKE32 &H00F0080,0
60 POKE32 &H00F0084,FB
70 POKE32 &H00F0000,1
80 BLIT FILL FB,320,200,&H00001020,ST
90 BLIT FILL SPR,8,8,&H000000FF,SS
100 BLIT FILL SPR+2*SS+2*4,4,4,&H0000FFFF,SS
110 DATA 24,60,126,255,255,126,60,24
120 FOR Y=0 TO 7
130 READ M
140 POKE8 MSK+Y,M
150 NEXT Y
160 POKE32 &H00F0024,SPR
170 POKE32 &H00F0028,FB+80*ST+156*4
180 POKE32 &H00F002C,8
190 POKE32 &H00F0030,8
200 POKE32 &H00F0034,SS
210 POKE32 &H00F0038,ST
220 POKE32 &H00F0040,MSK
230 POKE32 &H00F0020,3
240 POKE32 &H00F001C,1
250 PRINT PEEK32(&H00F0044)

```

The sprite and mask live in ordinary main memory. The mask bytes in line 110 are decimal forms of \$18, \$3C, \$7E, \$FF, \$FF, \$7E, \$3C, \$18. Lines 120-150 write them with POKE8 because the mask is a byte stream, not a table of 32-bit words. The final PRINT should show 2.

4.6.9 ALPHA_COPY

ALPHA_COPY has two forms. With BLT_FLAGS bit 12 clear, it is an RGBA32 sprite copy with source alpha. With bit 12 set, the source is an 8-bit alpha template and BLT_FG supplies the colour.

Source alpha	Result
0	Leave destination unchanged.
255	Copy source pixel exactly.
1-254	Blend source over destination with integer alpha.

The blend is per channel:

$$\text{out} = (\text{source} * \text{alpha} + \text{destination} * (255 - \text{alpha})) / 255$$

This is different from the compositor rule in Chapter 3. For ALPHA_COPY, a pixel such as \$00010203 is transparent because its alpha byte is zero.

This example writes a translucent red sprite one byte at a time so the alpha byte is exact. The blitter blends it over a dark blue background:

```

10 REM ALPHA COPY GLOW
20 FB=&H00100000:SPR=&H00620000
30 ST=320*4:SS=8*4
40 POKE32 &H000F0004,&H04
50 POKE32 &H000F0080,0
60 POKE32 &H000F0084,FB
70 POKE32 &H000F0000,1
80 BLIT FILL FB,320,200,&H00400000,ST
90 FOR Y=0 TO 7
100 FOR X=0 TO 7
110 A=SPR+(Y*8+X)*4
120 POKE8 A,255:POKE8 A+1,0:POKE8 A+2,0:POKE8 A+3,128
130 NEXT X:NEXT Y
140 POKE32 &H000F0024,SPR
150 POKE32 &H000F0028,FB+96*ST+156*4
160 POKE32 &H000F002C,8
170 POKE32 &H000F0030,8
180 POKE32 &H000F0034,SS
190 POKE32 &H000F0038,ST
200 POKE32 &H000F0020,4
210 POKE32 &H000F001C,1
220 PRINT PEEK32(&H000F0044)

```

Line 120 stores red, green, blue, and alpha bytes. The alpha value 128 makes the source about half opaque. The final PRINT should show 2.

For alpha-template mode, write the source as one byte per pixel. A source byte of 0 leaves the destination alone, 255 writes BLT_FG fully opaque, and values in between blend BLT_FG over the destination. If BLT_SRC_STRIDE is 0, the source stride defaults to the width in bytes, not to RGBA32 width.

This shorter example draws a soft four-pixel bar from an 8-bit alpha template:

```

10 REM ALPHA TEMPLATE BAR
20 FB=&H00100000:TPL=&H00630000
30 ST=320*4
40 POKE32 &H000F0004,&H04
50 POKE32 &H000F0080,0
60 POKE32 &H000F0084,FB
70 POKE32 &H000F0000,1
80 BLIT FILL FB,320,200,&H00000040,ST
90 DATA 32,96,192,255,255,192,96,32
100 FOR X=0 TO 7:READ A:POKE8 TPL+X,A:NEXT X
110 POKE32 &H000F0024,TPL
120 POKE32 &H000F0028,FB+100*ST+156*4
130 POKE32 &H000F002C,8
140 POKE32 &H000F0030,1
150 POKE32 &H000F0034,0
160 POKE32 &H000F0038,ST
170 POKE32 &H000F048C,&H0000C0FF
180 POKE32 &H000F0488,4096
190 POKE32 &H000F0020,4
200 POKE32 &H000F001C,1
210 PRINT PEEK32(&H000F0044)

```

Line 180 sets BLT_FLAGS bit 12. Line 170 sets BLT_FG. Line 150 sets the source stride to 0, so it defaults to eight alpha bytes for this row.

4.6.10 COLOR_EXPAND

COLOR_EXPAND turns a 1-bit template into pixels. This is useful for software fonts, monochrome sprites, stipples, and mask drawing. The operation can write RGBA32 or CLUT8 depending on BLT_FLAGS.

Register	Meaning
BLT_MASK	Template base address.
BLT_DST	Destination base address.
BLT_WIDTH, BLT_HEIGHT	Output size in pixels.
BLT_DST_STRIDE	Destination bytes per row.
BLT_FG	Foreground value for set template bits.
BLT_BG	Background value for clear template bits.
BLT_MASK_MOD	Template bytes to advance after each row.
BLT_MASK_SRCX	First bit offset in the template row.
BLT_FLAGS	BPP, JAM1, invert-template, and invert-mode bits.

Template bytes are read most-significant-bit first: bit 7 of the first byte is the leftmost pixel, then bit 6, and so on. If BLT_MASK_SRCX is 3, the first output pixel starts at the fourth template bit.

There are three drawing behaviours:

Flags	Set template bit	Clear template bit
neither JAM1 nor invert-mode	write BLT_FG	write BLT_BG
JAM1	write BLT_FG	leave destination unchanged
invert-mode	XOR destination with all ones	leave destination unchanged

Invert-template flips the template bit before the table above is applied.

4.6.11 SCALE

SCALE copies a source rectangle to a destination rectangle using nearest-neighbour sampling. It supports RGBA32 and CLUT8.

Register	Meaning
BLT_SRC	Source base address.
BLT_DST	Destination base address.
BLT_WIDTH, BLT_HEIGHT	Source width and height.
BLT_COLOR	Destination size: low 16 bits = width, high 16 bits = height.
BLT_SRC_STRIDE, BLT_DST_STRIDE	Source and destination row strides.
BLT_FLAGS	BPP. Draw mode is not used by SCALE.

The packed destination size is:

```
BLT_COLOR = dstWidth + dstHeight * 65536
```

Scaling is nearest-neighbour. There is no filtering and no alpha blend.

4.6.12 A useful blitter BASIC example

This program uses the blitter as a drawing tool: it clears a 320 x 200 RGBA32 screen, builds a small off-screen sprite, copies that sprite to several positions, and draws crossing diagonal lines.

```
10 REM VIDEOCHIP BLITTER POSTER
20 FB=&H00100000:SPR=&H00140000
30 W=320:H=200:ST=W*4:SS=16*4
40 POKE32 &H000F0004,&H04
50 POKE32 &H000F0080,0
60 POKE32 &H000F0084,FB
70 POKE32 &H000F0000,1
80 BLIT FILL FB,W,H,&H00001020,ST
90 BLIT FILL FB+12*ST+20*4,280,40,&H000000AA,ST
100 BLIT FILL FB+58*ST+20*4,280,40,&H0000AA00,ST
110 BLIT FILL FB+104*ST+20*4,280,40,&H00AA0000,ST
120 BLIT FILL SPR,16,16,&H000000FF,SS
130 BLIT FILL SPR+4*SS+4*4,8,8,&H0000FFFF,SS
140 FOR I=0 TO 9
150 X=24+I*28:Y=150+INT(20*SIN(I))
160 BLIT COPY SPR,FB+Y*ST+X*4,16,16,SS,ST
170 NEXT I
180 BLIT LINE 0,0,319,199,&H00FFFFFF
190 BLIT LINE 0,199,319,0,&H00FFFFFF
200 PRINT PEEK32(&H000F0044)
```

Line 80 clears the screen. Lines 90-110 draw coloured bands. Lines 120-130 build a 16 x 16 off-screen sprite at SPR. Line 160 copies that sprite into the framebuffer with explicit source and destination strides. The final PRINT should show 2 for BLT_STATUS.DONE.

4.6.13 A colour-expand example

This raw-register example draws an 8-pixel template row. The template byte \$B0 is binary 10110000, so pixels 0, 2, and 3 use the foreground colour and the rest use the background.

```

10 REM COLOUR EXPAND TEMPLATE
20 FB=&H00100000:TM=&H00300000:ST=320*4
30 POKE32 &H000F0004,&H04
40 POKE32 &H000F0080,0
50 POKE32 &H000F0084,FB
60 POKE32 &H000F0000,1
70 BLIT FILL FB,320,200,&H00000000,ST
80 POKE8 TM,&HB0
90 POKE32 &H000F0040,TM
100 POKE32 &H000F0028,FB+80*ST+120*4
110 POKE32 &H000F002C,8
120 POKE32 &H000F0030,1
130 POKE32 &H000F0038,ST
140 POKE32 &H000F048C,&H000000FF
150 POKE32 &H000F0490,&H0000FF00
160 POKE32 &H000F0494,1
170 POKE32 &H000F0498,0
180 POKE32 &H000F0488,48
190 POKE32 &H000F0020,6
200 POKE32 &H000F001C,1
210 PRINT PEEK32(&H000F0044)

```

Line 80 uses POKE8 because the template is a byte stream. Line 180 writes BLT_FLAGS = 48, which is RGBA32 plus draw mode 3 (COPY). Line 190 selects COLOR_EXPAND. Line 200 starts the blitter.

4.6.14 A scale example

This example draws a small source square, then scales it to 64 x 64 with raw blitter registers:

```

10 REM NEAREST SCALE
20 FB=&H00100000:SRC=&H00150000:DST=FB+80*320*4+120*4
30 ST=320*4:SS=16*4
40 POKE32 &H000F0004,&H04
50 POKE32 &H000F0080,0
60 POKE32 &H000F0084,FB
70 POKE32 &H000F0000,1
80 BLIT FILL FB,320,200,&H00000000,ST
90 BLIT FILL SRC,16,16,&H000000CC,SS
100 BLIT FILL SRC+4*SS+4*4,8,8,&H00CCCC00,SS
110 POKE32 &H000F0024,SRC
120 POKE32 &H000F0028,DST
130 POKE32 &H000F002C,16
140 POKE32 &H000F0030,16
150 POKE32 &H000F0034,SS
160 POKE32 &H000F0038,ST
170 POKE32 &H000F003C,64+64*65536
180 POKE32 &H000F0488,48
190 POKE32 &H000F0020,7
200 POKE32 &H000F001C,1
210 PRINT PEEK32(&H000F0044)

```

Line 170 packs destination width and height into BLT_COLOR. Line 190 selects SCALE.

4.6.15 Mode 7 affine texture mapping

BLT_OP = 5 selects the Mode 7 unit. It renders a rectangular destination by sampling a source texture through an affine transform. For every destination pixel (x,y), the chip computes:

$$U = U0 + x * DU_COL + y * DU_ROW$$
$$V = V0 + x * DV_COL + y * DV_ROW$$

U, V, and all four delta registers are signed 16.16 fixed-point numbers:

Value	Meaning
\$00010000	1.0 texel
\$00008000	0.5 texel
\$00000000	0.0
\$FFFF0000	-1.0 texel

The integer texel coordinate is the upper 16 bits. The fractional part gives sub-pixel movement, but sampling is nearest-lower-texel: there is no bilinear filtering. After extracting the integer part, the chip wraps the coordinate with the texture masks:

$$\text{tex_x} = (U \gg 16) \text{ AND } \text{BLT_MODE7_TEX_W}$$
$$\text{tex_y} = (V \gg 16) \text{ AND } \text{BLT_MODE7_TEX_H}$$

BLT_MODE7_TEX_W and BLT_MODE7_TEX_H are masks, not literal widths. They must be one less than a power of two:

Texture size	Mask
2	1
4	3
8	7
64	63
256	255

An invalid mask, such as 5, sets BLT_STATUS bit 0 (ERR) and the blit stops. Negative coordinates wrap naturally through the same mask: -1.0 with mask 3 samples texel 3.

Mode 7 honours the BLT_FLAGS BPP field: it samples texels and writes destination pixels as RGBA32 (four bytes) by default, or as CLUT8 (one index byte) when BLT_FLAGS selects CLUT8. The texture and destination strides are interpreted in that pixel size. Mode 7 does not use BLT_COLOR, BLT_MASK, BLT_FG, or BLT_BG.

4.6.16 Mode 7 setup

The raw-register setup order is:

1. Put the texture in memory. Each texel is four bytes (RGBA32) or one index byte (CLUT8), matching the BLT_FLAGS BPP field.
2. Write BLT_SRC to the texture base address.
3. Write BLT_DST to the destination base address.
4. Write BLT_WIDTH and BLT_HEIGHT to the destination size in pixels.

5. Write BLT_SRC_STRIDE to the texture row stride in bytes, or 0 to use $(\text{BLT_MODE7_TEX_W} + 1) * \text{bytes_per_pixel}$ (4 for RGBA32, 1 for CLUT8).
6. Write BLT_DST_STRIDE to the destination row stride in bytes, or 0 to use the current framebuffer stride when the destination is in VRAM.
7. Write the six transform registers: U0, V0, DU_COL, DV_COL, DU_ROW, DV_ROW.
8. Write the two texture masks.
9. Write 5 to BLT_OP.
10. Write 1 to BLT_CTRL.

The BASIC helper has the same shape:

```
BLIT MODE7 src, dst, w, h, u0, v0, duCol, dvCol, duRow, dvRow, uMask, vMask
BLIT MODE7 src, dst, w, h, u0, v0, duCol, dvCol, duRow, dvRow, uMask, vMask, srcStride,
dstStride
```

The 12-argument form clears both stride registers and uses the defaults. If you supply strides, supply both.

For rotation and zoom, compute:

```
CA = COS(angle) / scale
SA = SIN(angle) / scale
DU_COL = CA * 65536
DV_COL = SA * 65536
DU_ROW = -SA * 65536
DV_ROW = CA * 65536
```

To keep the texture centre (TC, TC) at the centre of a destination rectangle with half-width HW and half-height HH:

```
U0 = (TC - HW*CA + HH*SA) * 65536
V0 = (TC - HW*SA - HH*CA) * 65536
```

This is an affine transform. It rotates, scales, skews, and scrolls, but it does not do perspective.

4.6.17 A Mode 7 BASIC example

This program builds a 64 x 64 checkerboard texture in memory, turns on the 320 x 200 VideoChip mode, and renders the texture rotated into the framebuffer:

```

10 REM MODE7 CHECKERBOARD
20 TB=&H00500000:FB=&H00100000:FP=65536
30 W=320:H=200:T=64:TS=T*4
40 POKE32 &H000F0004,&H04
50 POKE32 &H000F0080,0
60 POKE32 &H000F0084,FB
70 POKE32 &H000F0000,1
80 FOR Y=0 TO T-1
90 FOR X=0 TO T-1
100 C=(INT(X/8)+INT(Y/8)) AND 1
110 IF C=0 THEN P=&H000000FF
120 IF C<>0 THEN P=&H0000FFFF
130 POKE32 TB+(Y*T+X)*4,P
140 NEXT X:NEXT Y
150 A=.55:SC=1.15
160 CA=COS(A)/SC:SA=SIN(A)/SC
170 DC=INT(CA*FP):DS=INT(SA*FP)
180 HW=W/2:HH=H/2:TC=T/2
190 U0=INT((TC-HW*CA+HH*SA)*FP)
200 V0=INT((TC-HW*SA-HH*CA)*FP)
210 BLIT MODE7 TB,FB,W,H,U0,V0,DC,DS,0-DS,DC,T-1,T-1,TS,W*4
220 PRINT PEEK32(&H000F0044)

```

Line 210 is the whole Mode 7 operation. DC and DS are the fixed-point cosine and sine terms. 0-DS is the negative sine term for DU_ROW. T-1 gives masks 63,63, which wrap the 64 x 64 texture. TS is the source stride (64 * 4), and W*4 is the destination stride (320 * 4). The final PRINT should show 2, meaning BLT_STATUS.DONE is set and ERR is clear.

4.7 The copper list

The blitter changes memory when your program asks it to. The copper changes registers while the display is being drawn. Use it when a horizontal split, colour bar, palette change, or per-scanline effect must happen at a raster position instead of at the next convenient BASIC line.

The copper is a scanline-driven register writer. It reads a list of 32-bit words from main memory. During each frame it waits for raster positions and writes MMIO registers. Use it for raster colour bars, mid-frame palette changes, display splits, and per-scanline setup that should not burn CPU time.

BASIC has a copper-list builder:

BASIC form	Effect
COPPER LIST [addr]	Set COPPER_PTR and set the BASIC build cursor. With no addr, BASIC allocates a 4 KB public MEMALLOC copper list.
COPPER WAIT y	Append a wait for scanline y, X position 0.
COPPER MOVE addr,value	Append a write to absolute MMIO address addr.
COPPER END	Append an end word.
COPPER ON	Enable the copper.
COPPER OFF	Disable the copper.

COPPER MOVE accepts MMIO addresses at or above \$000A0000. It emits a SETBASE for the exact address, then emits a MOVE with register index 0. This is larger than hand-packed copper code, but it is easy to type and can target VideoChip or other

MMIO blocks.

4.7.1 Copper control and status

Register	Meaning
COPPER_CTRL	Bit 0 enables. Bit 1 resets/latches the staged pointer.
COPPER_PTR	Address of the first copper word.
COPPER_PC	Address of the next copper word.
COPPER_STATUS	Running, waiting, and halted bits.

Enabling the copper latches COPPER_PTR into COPPER_PC and clears waiting/halted state. At the start of every frame, the copper starts again from COPPER_PTR, clears waiting/halted state, and resets its MMIO base to \$000F0000.

COPPER_STATUS has three read-only bits:

Bit	Name	Meaning
0	RUNNING	The copper is enabled and not halted.
1	WAITING	The copper is paused on a WAIT instruction.
2	HALTED	The copper has hit END (or a bad opcode) this frame.

4.7.2 Copper instruction words

Each copper instruction is a 32-bit word. The top two bits select the opcode:

Top bits	Opcode	Length	Effect
00	WAIT	1 word	Wait until raster reaches (Y, X).
01	MOVE	2 words	Write the following data word to IO_BASE + index*4.
10	SETBASE	1 word	Change IO_BASE.
11	END	1 word	Halt until the next frame.

Word layouts:

Instruction	Word layout
WAIT	bits 23-12 = Y, bits 11-0 = X.
MOVE	bits 23-16 = register index. The next word is the data.
SETBASE	bits 23-0 = base >> 2. The base must be 4-byte aligned.
END	\$C0000000.

The wait test is current Y > wait Y, or current Y = wait Y and current X >= wait X. BASIC COPPER WAIT only emits X 0. Hand written copper lists may use any 12-bit X value.

If IO_BASE is \$000F0000, MOVE register index 15 writes \$000F003C, which is BLT_COLOR. A hand-packed move to BLT_COLOR therefore uses word \$400F0000, followed by the colour word.

4.7.3 A copper colour-bar example

This program clears the screen, builds a copper list, and lets the copper draw three horizontal bands by changing the raster-band colour at selected scanlines.

```
10 REM COPPER RASTER BANDS
20 FB=&H00100000:CL=MEMALLOC(4096,4):ST=320*4
30 POKE32 &H000F0004,&H04
40 POKE32 &H000F0080,0
50 POKE32 &H000F0084,FB
60 POKE32 &H000F0000,1
70 BLIT FILL FB,320,200,&H00000000,ST
80 COPPER LIST CL
90 COPPER WAIT 0
100 COPPER MOVE &H000F0048,0
110 COPPER MOVE &H000F004C,67
120 COPPER MOVE &H000F0050,&H000000AA
130 COPPER MOVE &H000F0054,1
140 COPPER WAIT 67
150 COPPER MOVE &H000F0048,67
160 COPPER MOVE &H000F004C,66
170 COPPER MOVE &H000F0050,&H0000AA00
180 COPPER MOVE &H000F0054,1
190 COPPER WAIT 133
200 COPPER MOVE &H000F0048,133
210 COPPER MOVE &H000F004C,67
220 COPPER MOVE &H000F0050,&H00AA0000
230 COPPER MOVE &H000F0054,1
240 COPPER END
250 COPPER ON
260 PRINT PEEK32(&H000F0018)
```

The list writes VIDEO_RASTER_Y, VIDEO_RASTER_HEIGHT, VIDEO_RASTER_COLOR, and VIDEO_RASTER_CTRL at three scanlines. The final status print usually shows 1 immediately after enabling because the copper is running; after a frame reaches COPPER END, bit 2 (HALTED) is set until the next frame restart.

4.7.4 Hand-packed copper example

The same mechanism can be entered directly as words. This short list waits for scanline 100, writes BLT_COLOR, and ends:

```
10 CL=&H00390000
20 POKE32 CL+0,&H00064000
30 POKE32 CL+4,&H400F0000
40 POKE32 CL+8,&H000000FF
50 POKE32 CL+12,&HC00000000
60 POKE32 &H000F0010,CL
70 POKE32 &H000F000C,1
```

\$00064000 is WAIT 100,0. \$400F0000 is MOVE register index 15, and the following word is the data to write.

4.8 The raster band

The raster band is a single-shot horizontal fill. It is separate from the blitter and is especially useful from copper lists.

Register	Meaning
VIDEO_RASTER_Y	First scanline to fill.
VIDEO_RASTER_HEIGHT	Number of scanlines. 0 means 1.
VIDEO_RASTER_COLOR	RGBA32 colour, or CLUT8 index in the low byte.
VIDEO_RASTER_CTRL	Write bit 0 to start the fill.

In RGBA32 mode, the band writes VIDEO_RASTER_COLOR as pixels. In CLUT8 mode, it writes the low byte of VIDEO_RASTER_COLOR as the palette index. If Y is outside the current mode, the request is ignored. If the band extends past the bottom of the screen, it is clipped at the bottom.

The band follows the active framebuffer path. If VIDEO_FB_BASE points at a valid RAM or VRAM framebuffer, the band writes that memory. If the chip is using direct VRAM without a separate base, it writes direct VRAM. If neither path is active, it writes the internal front buffer. During compositor-managed copper rendering of a high framebuffer outside the legacy VRAM aperture, raster-band writes are skipped for that high buffer so that the compositor's per-scanline path remains in charge.

Example:

```
10 FB=&H00100000:ST=320*4
20 POKE32 &H000F0004,&H04
30 POKE32 &H000F0080,0
40 POKE32 &H000F0084,FB
50 POKE32 &H000F0000,1
60 POKE32 &H000F0048,90
70 POKE32 &H000F004C,20
80 POKE32 &H000F0050,&H0000FFFF
90 POKE32 &H000F0054,1
```

This fills scanlines 90 through 109 with cyan.

4.9 VIDEO_STATUS

VIDEO_STATUS is read-only and reports three bits:

Bit	Name	Meaning
0	HAS_CONTENT	At least one pixel has been written to VRAM since the chip was enabled.
1	VBLANK	The chip is currently in the vertical-retrace interval.
2	FB_ERR	The framebuffer source is unreachable or undersized for the current mode.

The VBLANK bit is the safest place to poll if you have no interrupt handler installed and need to wait for the retrace.

FB_ERR is useful when using FB_BASE. It is set when the selected framebuffer does not contain enough bytes for the current mode and colour mode. For example, 1920 x 1080 RGBA32 needs 8,294,400 bytes, so it cannot fit wholly inside the five-megabyte VRAM window. Point FB_BASE at ordinary memory with enough space, or use CLUT8.

4.10 Putting it together

A minimal "set a mode, load a palette, draw a frame" sequence from machine language looks like this:

1. Write VIDEO_MODE and FB_BASE.
2. Write COLOR_MODE (0 for RGBA32, non-zero for CLUT8).
3. If CLUT8: write 256 palette entries via PAL_INDEX/PAL_DATA.
4. Write VIDEO_CTRL = 1.
5. Fill the framebuffer with whatever you want to display.
6. Optionally start the copper to change registers per-scanline, start the blitter for fast copies, or program a raster band.

Chapter 24 describes how the address constants above appear in machine-language programs. Chapter 10 shows how tile and sprite layers map onto the framebuffer.