VDP 64x48 Documentation

Table of Contents

Introduction

The Quadium VDP is a small video display processor designed for educational and visual experiments inside Quadium 4004 Workbench.

It provides a 64x48 pixel display with 16 colors. Programs running on the emulated Intel 4004 do not draw directly into the screen memory. Instead, they send compact commands to the VDP through 4001 ROM ports.

The design is inspired by classic video display processors, especially the Texas Instruments TMS9918 family, but it is not a full TMS9918 emulator. It borrows the most useful ideas for learning:

- A separate video device with its own video memory.
- Command/data style communication.
- A status bit for display synchronization.
- Explicit frame presentation.
- A small command protocol that a 4-bit CPU can realistically drive.

The goal is not historical purity. The goal is to make video programming possible, understandable, and enjoyable on a machine whose original hardware was never meant to draw a color bitmap.

Why a VDP Exists in a 4004 Workbench

The Intel 4004 and the original MCS-4 family were designed for calculators and embedded control systems, not for video graphics. A real 4004 system could control lights, printers, switches, displays, or other small devices, but a color bitmap display would be far beyond the normal use case of the original 1971 hardware.

The Quadium VDP is therefore an intentional anachronism.

It answers this question:

> What if a very early 4-bit system could talk to a small, friendly video chip?

This makes it possible to write programs that are visually clear:

- Test patterns.
- Pixel plotting.
- Lines and rectangles.
- Small animations.
- Games and graphical demonstrations.
- Educational experiments where the visual result matters.

Without a VDP, the MCS-4 would need to spend most of its time manually updating external display hardware one tiny piece at a time. That would be realistic in a strict low-level sense, but not very productive for learning graphics concepts.

The VDP gives the 4004 a partner: the CPU decides what to draw, while the video device stores and presents the picture.

Relationship with the TMS9918

The TMS9918 was a real video display processor used in several 1980s computers and consoles. It had its own video RAM, a data/control interface, status flags, and a display refresh process independent from the CPU.

Quadium's VDP partially follows that idea:

- The CPU does not own the display memory directly.
- The video chip has internal memory.
- The CPU communicates through ports.
- There is a command/control side and a data side.
- There is a status flag that can be polled to synchronize animation.
- The display refresh is conceptually separate from CPU speed.

However, the Quadium VDP is deliberately simpler.

It does not try to reproduce the real TMS9918 register set, video modes, name tables, pattern tables, sprites, color tables, timing rules, or memory access protocol. Those would be interesting, but they would also make the device much harder to teach and much harder to use from QuadBasic.

The result is a "TMS9918-style" VDP, not a TMS9918 clone.

What Is Emulated

The Quadium VDP currently emulates these broad concepts:

- A video device separate from the CPU.
- Internal VRAM owned by the video device.
- A 64x48 pixel bitmap.
- 16 color indexes.
- Port-based communication through 4001 ROM ports.
- A DATA port for command parameters and pixel data.
- A CTRL port for drawing commands.
- A STATUS port for synchronization.
- A VBLANK status bit with read-clear behavior.
- Explicit PRESENT command for showing a completed frame.
- Implicit presentation helpers for simple smoke tests that never call PRESENT.
- Drawing primitives such as clear, plot, line, rectangle, and circle.
- Tile mode with a 16x8 name table, an 8x6 viewport, and a 64-entry pattern generator.
- Text mode with a built-in ROM font on a 16x8 cell grid (3x6 glyph in a 4x6 cell).
- Hardware sprites with 16 entries, 8x8 patterns, and STATUS collision flags.

This is enough to explain the most important idea: the CPU sends high-level drawing instructions, and the video processor performs the drawing in its own video memory.

What Is Not Emulated

The Quadium VDP does not emulate the full TMS9918.

In particular, it does not include:

- The original TMS9918 register map.
- Original TMS9918 VRAM addressing rules.
- Tile or pattern tables as in the original TMS9918 memory map.
- Scanline-level timing.
- PAL/NTSC video timing.
- True character generator memory.
- Original color limitations per tile.
- Original VDP interrupt behavior.
- Real analog video effects.

It also does not try to emulate the electrical behavior of a real TMS9918 bus. The port protocol is adapted to the MCS-4/4001 style I/O model and to QuadBasic.

This simplification is intentional. It keeps the device small enough to understand, while still teaching the important concepts of a separate video processor.

Why the Display Is Not Controlled Pixel by Pixel from the MCS-4

A 64x48 display contains 3072 pixels.

Each pixel is a 4-bit color value. If the MCS-4 had to manually update every pixel through normal external I/O, even a very small animation would require thousands of writes per frame.

That would cause several problems:

- Programs would become too large.
- QuadBasic examples would be hard to read.
- Drawing a simple shape would require too much code.
- Animation speed would depend too strongly on CPU speed.
- The 4004 would spend nearly all its time moving pixels instead of expressing the program logic.

The VDP avoids that by moving the heavy work into the video device.

For example, instead of sending every pixel of a circle, the program sends:

- The command "fill circle".
- The center position.
- The radius.
- The color.

The VDP then updates its own VRAM.

This is the main reason the device exists: it turns graphics into a conversation that a 4-bit CPU can realistically have.

Default QuadBasic Connections

The default QuadBasic wiring for the VDP uses three 4001 ROM ports:

- ROM7: DATA.
- ROM8: CTRL.
- ROM9: STATUS.

The DATA and CTRL ports are written by the program with `POKE`.

The STATUS port is read by the program with `PEEK`.

Example:

```
10 REM VDP DEFAULT CONNECTIONS
20 REM DATA   = ROM 7
30 REM CTRL   = ROM 8
40 REM STATUS = ROM 9
```

These defaults are used by the QuadBasic playground examples. If a user writes assembly manually and builds a custom peripheral layout, the physical port choices can be changed in the peripheral configuration.

VDP Command Model

The VDP uses a small command/data protocol.

The general pattern is:

```
POKE @ROMPORT(8), command
POKE @ROMPORT(7), parameter1
POKE @ROMPORT(7), parameter2
...
```

In the default layout:

- `@ROMPORT(8)` is the CTRL port.
- `@ROMPORT(7)` is the DATA port.

Some commands need no parameters. Other commands switch the DATA port into a temporary parameter-reading mode. Once the required parameters have been received, the command is executed.

All values are sent as nibbles, because the 4004 is a 4-bit CPU.

Coordinates larger than 15 are sent as two nibbles:

```
high nibble, low nibble
```

For example, decimal 42 is hexadecimal `2A`, so it is sent as:

```
POKE @ROMPORT(7), 2
POKE @ROMPORT(7), 10
```

QuadBasic Command Reference

QuadBasic provides high-level `VDP` statements that transpile to the same ROM-port traffic documented in the CTRL/DATA sections below. The default wiring is ROM 7 = DATA, ROM 8 = CTRL, ROM 9 = STATUS.

This section documents every VDP statement from the transpiler's point of view: what it does on the device, which CTRL command it emits, and any compile-time restrictions.

Quick map (statement to CTRL command)

QuadBasic statement	CTRL	Notes
`VDP PRESENT`	15	Show back buffer
`VDP WAIT VBLANK`	—	Poll STATUS (ROM 9)
`VDP FRAME`	15 + poll	`PRESENT` then `WAIT VBLANK`
`VDP MODE BITMAP`	11	DATA mode `0` (+ sprite flag if enabled)
`VDP MODE TILE`	11	DATA mode `1`
`VDP MODE TEXT fg, bg`	11	DATA mode `2`, then `fg`, `bg`
`VDP SPRITES ON` / `OFF`	11	Re-send mode with/without flag `4`
`VDP TEST`	1	Built-in test pattern
`VDP CLEAR color`	4	Clear back buffer
`VDP PLOT x, y, color`	5	One pixel
`VDP LINE x0, y0, x1, y1, color`	6	Line segment
`VDP RECT x, y, w, h, color`	7	Hollow rectangle
`VDP FILLRECT x, y, w, h, color`	8	Filled rectangle
`VDP CIRCLE x, y, r, color`	9	Hollow circle (bitmap)
`VDP FILLCIRCLE x, y, r, color`	10	Filled circle (bitmap)
`VDP TILE col, row, pattern`	12	One name-table cell (tile mode)
`VDP FILLMAP pattern`	14	Fill all 128 map cells
`VDP SCROLL sx, sy`	2	Tile viewport offset (pixels)
`VDP SCROLLHOME`	3	Reset tile scroll to origin
`VDP PATTERN id`	13	Begin 64-nibble pattern upload
`VDP PATTERNHOME`	10	Pattern stream address = 0
`VDP PATTERNADDR index`	9	12-bit pattern stream address
`VDP PATTERNDATA color`	DATA	One autoincrementing pattern pixel
`VDP CLS`	14	Text grid filled with space (code 0)
`VDP TEXTFILL code`	14	Text grid filled with one character
`VDP TEXT col, row, code`	12	One text cell
`VDP PRINT col, row, "text"`	12	One cell per supported character
`VDP SPRITE idx, x, y, pattern, color`	12	Sprite update (sprites enabled)
`VDP SPRITEHIDE idx`	12	Hide sprite (`Y = 255`)
`VDP STATUS variable`	—	`PEEK` STATUS into variable

Frame synchronization

VDP PRESENT

Shows the completed back buffer on the visible display. Equivalent to `POKE @ROMPORT(8), 15`.

```
10 VDP PRESENT
```

VDP WAIT VBLANK

Busy-waits until the VBLANK status bit is set on ROM 9. Equivalent to polling `PEEK @ROMPORT(9), V` until `V` is non-zero.

```
10 VDP WAIT VBLANK
```

VDP FRAME

Presents the current frame and then waits for VBLANK. Equivalent to `VDP PRESENT` followed by `VDP WAIT VBLANK`. This is the usual end-of-frame helper for animation loops.

```
10 VDP CLEAR 0
20 REM ... draw frame ...
30 VDP FRAME
40 GOTO 10
```

Video modes

VDP MODE BITMAP / VDP MODE TILE / VDP MODE TEXT

Select the base video mode with CTRL command 11. The transpiler remembers the last base mode and the last text colors.

```
10 VDP MODE BITMAP
20 VDP MODE TILE
30 VDP MODE TEXT 7, 0
```

Mode nibbles on DATA:

- `0` = bitmap (default).
- `1` = tile mode (16x8 name table, 8x6 viewport, 64 patterns).
- `2` = text mode (16x8 character grid, ROM font); followed by foreground and background palette indexes.

VDP SPRITES ON / VDP SPRITES OFF

Enables or disables hardware sprites on the current base mode by re-sending command 11 with or without the sprite flag (`4`). For text mode, the transpiler also re-sends the last `fg` and `bg` values.

```
10 VDP MODE BITMAP
20 VDP SPRITES ON
30 VDP SPRITES OFF
```

With sprites enabled, effective mode values are `4` (bitmap), `5` (tile), and `6` (text).

Bitmap drawing

These commands apply in bitmap mode (and compose the back buffer there). Use them for pixel plots, vector shapes, and full-screen clears before `VDP PRESENT` or `VDP FRAME`.

VDP TEST

Loads the built-in test pattern (CTRL command 1).

```
10 VDP TEST
```

VDP CLEAR

Clears the back buffer to a palette index (CTRL command 4).

```
10 VDP CLEAR 12
```

VDP PLOT

Plots one pixel (CTRL command 5). Coordinates and color may be compile-time literals or runtime scalar expressions (typically `BYTE` or `NIBBLE` variables); the transpiler emits two DATA nibbles per coordinate and one DATA nibble for color.

```
10 DIM X AS BYTE
20 DIM Y AS BYTE
30 VDP MODE BITMAP
40 X = 32
50 Y = 24
60 VDP PLOT X, Y, 8
```

VDP LINE / VDP RECT / VDP FILLRECT / VDP CIRCLE / VDP FILLCIRCLE

Drawing primitives mapped to CTRL commands 6 through 10. Coordinates, sizes, radius, and color may be literals or runtime scalar expressions (same lowering as `VDP PLOT`).

```
10 VDP MODE BITMAP
20 VDP LINE 4, 4, 60, 44, 12
30 VDP FILLRECT 20, 18, 24, 12, 8
40 VDP FILLCIRCLE 32, 24, 8, 9
```

Text mode

Use after `VDP MODE TEXT fg, bg`. The grid is 16 columns by 8 rows. Character codes are `0..63` (see command 12 below for the ROM font map). `VDP CLS` is the same as `VDP TEXTFILL 0`.

VDP CLS / VDP TEXTFILL

Fill every text cell with one character code (CTRL command 14).

```
10 VDP MODE TEXT 7, 0
20 VDP CLS
30 VDP TEXTFILL 37
```

`TEXTFILL` requires a compile-time character code literal.

VDP TEXT

Write one cell at `col`, `row` with a compile-time character code (CTRL command 12).

```
10 VDP TEXT 0, 0, 18
```

Column and row may be variables; the character code must be a literal.

VDP PRINT

Write a string literal left to right on one row. The transpiler expands each supported character to command 12 at compile time. When the string has more than one character, `col` must be a numeric literal.

```
10 VDP MODE TEXT 7, 0
20 VDP PRINT 0, 0, "SCORE 07"
```

Supported characters: space, digits `0..9`, `A..Z`, and `. , : ; ! ? - + = / ( )`.

Tile mode

Use after `VDP MODE TILE`. The visible 64x48 image is an 8x6 window into a 16x8 name table; each cell stores a pattern index `0..63` into the pattern generator.

VDP TILE / VDP FILLMAP

Update one map cell or fill all 128 cells with the same pattern (CTRL commands 12 and 14). `VDP TILE` accepts runtime variables for column, row, and pattern index.

```
10 VDP MODE TILE
20 VDP FILLMAP 1
30 VDP TILE 0, 0, 2
40 VDP TILE 7, 5, 2
```

VDP SCROLL / VDP SCROLLHOME

Move the tile viewport in pixels or reset scroll to the origin (CTRL commands 2 and 3 in tile mode). Scroll offsets may be runtime variables.

```
10 VDP SCROLL 8, 0
20 VDP SCROLLHOME
```

VDP PATTERN / VDP PATTERNHOME / VDP PATTERNADDR / VDP PATTERNDATA

Pattern-generator helpers (CTRL commands 13, 10, 9, and DATA stream writes). `VDP PATTERN id` opens command 13 and expects 64 following `VDP PATTERNDATA` writes (or a stream started with `PATTERNHOME` / `PATTERNADDR`). Pattern index and `PATTERNDATA` color may be variables; `PATTERNADDR` alone requires a compile-time address literal `0..4095`.

```
10 VDP MODE TILE
20 VDP PATTERNHOME
30 VDP PATTERNDATA 8
40 VDP PATTERN 1
```

Hardware sprites

Enable sprites with `VDP MODE BITMAP` (or tile/text) followed by `VDP SPRITES ON`. Sprites use the shared 8x8 pattern generator and are composited on `VDP PRESENT`.

VDP SPRITE / VDP SPRITEHIDE

Update one sprite (CTRL command 12, eight DATA nibbles) or hide it by forcing `Y = 255`. Operand order: sprite index, X, Y, pattern index, color. All five operands may be runtime scalar expressions.

```
10 VDP MODE BITMAP
20 VDP SPRITES ON
30 VDP SPRITE 0, 20, 16, 1, 8
40 VDP SPRITEHIDE 0
```

Pattern index `0` in a sprite pixel is transparent. Non-zero color replaces the pattern color for opaque pixels.

VDP STATUS

Reads the STATUS port into a variable. Equivalent to `PEEK @ROMPORT(9), variable`. Bit 0 is VBLANK; bit 1 reports more than four sprites on one scanline; bit 2 reports sprite overlap. Reading STATUS clears the latched bits until the next event.

```
10 DIM V AS NIBBLE
20 VDP STATUS V
30 IF V = 0 THEN GOTO 20
```

Compile-time vs runtime operands

Statement	Must be a compile-time literal
`VDP TEXTFILL`	character code `0..63`
`VDP TEXT`	character code `0..63`
`VDP PRINT`	column when the string has more than one character
`VDP PATTERNADDR`	pattern-stream address `0..4095`

Statement	Runtime variables allowed (typical)
`VDP PLOT`, `VDP LINE`, `VDP RECT`, `VDP FILLRECT`, `VDP CIRCLE`, `VDP FILLCIRCLE`, `VDP CLEAR`	coordinates, sizes, radius, color
`VDP TILE`, `VDP FILLMAP`, `VDP PATTERN`, `VDP PATTERNDATA`	column, row, pattern index, color
`VDP SCROLL`	scroll X, Y
`VDP SPRITE`, `VDP SPRITEHIDE`	index, X, Y, pattern, color (hide: index)
`VDP TEXT`	column, row
`VDP PRINT`	row; column if the string is one character
`VDP MODE TEXT fg, bg`	foreground and background colors

For runtime byte-sized operands, the transpiler emits two DATA nibbles (high then low) and re-selects the DATA ROM port before each `WRR`.

Low-level CTRL commands (POKE reference)

The following commands are sent directly to the CTRL port with `POKE @ROMPORT(8), command` (default layout). QuadBasic `VDP` statements above expand to this protocol.

Command 0: Clear to black

Clears the VDP back buffer to color 0.

```
POKE @ROMPORT(8), 0
```

Before the program has ever sent `PRESENT`, this command may also refresh the visible image as part of the VDP's implicit presentation helpers. After the first `PRESENT`, only `PRESENT` copies the back buffer to the visible display.

Command 1: Test pattern

Loads a built-in test pattern.

```
POKE @ROMPORT(8), 1
```

The pattern is built into the VDP. It is not read from program memory. The same implicit/explicit presentation rules as command 0 apply.

Command 2: Set VRAM address

Sets the internal sequential write address for pixel streaming. The next three DATA nibbles form a 12-bit pixel index (0 to 3071), sent high nibble first.

```
POKE @ROMPORT(8), 2
POKE @ROMPORT(7), ah
POKE @ROMPORT(7), am
POKE @ROMPORT(7), al
```

After the address is loaded, each further DATA nibble writes one pixel at the current address and then advances to the next pixel in row-major order (left to right, top to bottom). Addresses wrap at 3072 pixels.

In tile mode, command 2 sets the tile-map scroll offset in pixels instead of selecting a VRAM stream address. The next four DATA nibbles are:

- Scroll X high nibble.
- Scroll X low nibble.
- Scroll Y high nibble.
- Scroll Y low nibble.

The VDP clamps the offset to the valid range for the 16x8 map viewed through the 8x6 window (0 to 63 on X, 0 to 15 on Y). The name table is not rewritten; only the viewport origin changes.

This is mainly useful for low-level pixel streaming in bitmap mode.

Command 3: Home

In bitmap mode, sets the internal VRAM address to zero.

```
POKE @ROMPORT(8), 3
```

In tile mode, resets tile scroll to `(0, 0)` and recomposes the viewport from the current name table.

Command 4: Clear to color

Clears the back buffer to a specific palette index. The next DATA nibble is the color.

```
POKE @ROMPORT(8), 4
POKE @ROMPORT(7), color
```

The same implicit/explicit presentation rules as command 0 apply.

Command 5: Plot pixel

Plots one pixel.

Parameters:

- X high nibble.
- X low nibble.
- Y high nibble.
- Y low nibble.
- Color.

```
POKE @ROMPORT(8), 5
POKE @ROMPORT(7), xh
POKE @ROMPORT(7), xl
POKE @ROMPORT(7), yh
POKE @ROMPORT(7), yl
POKE @ROMPORT(7), color
```

Command 6: Line

Draws a line.

Parameters:

- X0 high nibble.
- X0 low nibble.
- Y0 high nibble.
- Y0 low nibble.
- X1 high nibble.
- X1 low nibble.
- Y1 high nibble.
- Y1 low nibble.
- Color.

```
POKE @ROMPORT(8), 6
POKE @ROMPORT(7), x0h
POKE @ROMPORT(7), x0l
POKE @ROMPORT(7), y0h
POKE @ROMPORT(7), y0l
POKE @ROMPORT(7), x1h
POKE @ROMPORT(7), x1l
POKE @ROMPORT(7), y1h
POKE @ROMPORT(7), y1l
POKE @ROMPORT(7), color
```

Command 7: Rectangle outline

Draws the outline of a rectangle.

Parameters:

- X high nibble.
- X low nibble.
- Y high nibble.
- Y low nibble.
- Width high nibble.
- Width low nibble.
- Height high nibble.
- Height low nibble.
- Color.

```
POKE @ROMPORT(8), 7
POKE @ROMPORT(7), xh
POKE @ROMPORT(7), xl
POKE @ROMPORT(7), yh
POKE @ROMPORT(7), yl
POKE @ROMPORT(7), wh
POKE @ROMPORT(7), wl
POKE @ROMPORT(7), hh
POKE @ROMPORT(7), hl
POKE @ROMPORT(7), color
```

Command 8: Filled rectangle

Draws a filled rectangle. It uses the same parameters as command 7.

```
POKE @ROMPORT(8), 8
POKE @ROMPORT(7), xh
POKE @ROMPORT(7), xl
POKE @ROMPORT(7), yh
POKE @ROMPORT(7), yl
POKE @ROMPORT(7), wh
POKE @ROMPORT(7), wl
POKE @ROMPORT(7), hh
POKE @ROMPORT(7), hl
POKE @ROMPORT(7), color
```

Command 9: Circle outline

In bitmap mode, draws the outline of a circle.

Parameters:

- Center X high nibble.
- Center X low nibble.
- Center Y high nibble.
- Center Y low nibble.
- Radius high nibble.
- Radius low nibble.
- Color.

```
POKE @ROMPORT(8), 9
POKE @ROMPORT(7), cxh
POKE @ROMPORT(7), cxl
POKE @ROMPORT(7), cyh
POKE @ROMPORT(7), cyl
POKE @ROMPORT(7), rh
POKE @ROMPORT(7), rl
POKE @ROMPORT(7), color
```

In tile mode, command 9 selects a write address in the pattern generator. The next three DATA nibbles form a 12-bit pixel index (0 to 4095), high nibble first. After the address is loaded, each further DATA nibble writes one 4-bit color index into the generator and advances to the next pixel in row-major order across the 64 patterns. Indexes wrap at 4096 pixels. Pattern uploads do not compose the visible screen.

```
POKE @ROMPORT(8), 9
POKE @ROMPORT(7), ah
POKE @ROMPORT(7), am
POKE @ROMPORT(7), al
POKE @ROMPORT(7), pixel0
...
```

Command 10: Filled circle

In bitmap mode, draws a filled circle. It uses the same parameters as command 9.

```
POKE @ROMPORT(8), 10
POKE @ROMPORT(7), cxh
POKE @ROMPORT(7), cxl
POKE @ROMPORT(7), cyh
POKE @ROMPORT(7), cyl
POKE @ROMPORT(7), rh
POKE @ROMPORT(7), rl
POKE @ROMPORT(7), color
```

In tile mode, command 10 resets the pattern-generator write address to 0 and leaves the VDP in pattern-stream mode.

Command 11: Set video mode

Selects bitmap mode, tile mode, or text mode. The next DATA nibble is the mode value.

```
POKE @ROMPORT(8), 11
POKE @ROMPORT(7), mode
```

For text mode, send two more DATA nibbles after the mode value:

```
POKE @ROMPORT(8), 11
POKE @ROMPORT(7), 2
POKE @ROMPORT(7), fg
POKE @ROMPORT(7), bg
```

Mode values:

- `0` = bitmap mode (default).
- `1` = tile mode (Graphic I style).
- `2` = text mode (4x6 ROM glyphs on a 16x8 cell grid).
- Add `4` to enable hardware sprites on top of the current base mode. For example, `4` = bitmap + sprites, `5` = tile + sprites, `6` = text + sprites.

In tile mode, the visible image is built from a 16x8 name table of pattern indexes and a pattern generator with 64 patterns. Each pattern is an 8x8 tile of 4-bit color indexes. The 64x48 display shows an 8x6 viewport into that map. Commands 2 and 3 move or reset the viewport without rewriting cells.

In text mode, a 16x8 cell grid stores one character code per cell. The VDP draws each cell from a native 4x6 ROM font on the 64x48 display. Command 13 does not change the ROM font. Entering text mode with command 11 also accepts the foreground and background colors used when the text screen is composed.

Command 12: Write name table entry or update sprite

In tile mode, command 12 writes one name-table cell. The next four DATA nibbles are:

- Column (0 to 15).
- Row (0 to 7).
- Pattern index high nibble.
- Pattern index low nibble.

In text mode, command 12 writes one character cell. The next four DATA nibbles are:

- Column (0 to 15).
- Row (0 to 7).
- Character code high nibble.
- Character code low nibble.

Character codes are 0 to 63. Send the code as two DATA nibbles (high, then low). Code `0` is space. Codes `1` to `10` are digits `0` to `9`. Codes `11` to `36` are `A` to `Z`. Codes `37` to `48` are punctuation: `. , : ; ! ? - + = / ( )`. Other codes render as blank cells.

The text grid is 16 columns by 8 rows. Each glyph is 3x6 pixels inside a 4-pixel-wide cell, leaving a 1-pixel gap between characters.

In bitmap mode with sprites enabled, command 12 updates one sprite. The next eight DATA nibbles are:

- Sprite index (0 to 15).
- Y high nibble.
- Y low nibble.
- X high nibble.
- X low nibble.
- Pattern index high nibble.
- Pattern index low nibble.
- Color.

Pattern index 0 in a sprite pixel means transparent. If the color nibble is non-zero, it replaces the pattern color for non-transparent pixels.

Set `Y = 255` to hide a sprite.

```
POKE @ROMPORT(8), 12
POKE @ROMPORT(7), col
POKE @ROMPORT(7), row
POKE @ROMPORT(7), ph
POKE @ROMPORT(7), pl
```

Command 13: Define pattern

Selects a pattern index, then streams 64 DATA nibbles into that pattern in row-major order (one nibble per pixel). The upload updates only the pattern generator. It does not compose the visible screen.

```
POKE @ROMPORT(8), 13
POKE @ROMPORT(7), ph
POKE @ROMPORT(7), pl
POKE @ROMPORT(7), pixel0
...
```

Pattern indexes are 0 to 63.

Command 14: Fill name table

Fills all 128 tile-map cells, or all 128 text cells, with the same pattern or character code. The next two DATA nibbles are the pattern index or character code.

```
POKE @ROMPORT(8), 14
POKE @ROMPORT(7), ph
POKE @ROMPORT(7), pl
```

Command 15: Present

Shows the completed back buffer on the visible display.

```
POKE @ROMPORT(8), 15
```

`PRESENT` does not wait for the next display refresh. It only makes the finished frame visible. If the program wants stable animation, it should wait for VBLANK using the STATUS port.

The first `PRESENT` also switches the VDP into explicit presentation mode. From that point on, only `PRESENT` copies the finished back buffer to the visible display. Frame-start commands such as `CLEAR`, `CLEAR_COLOR`, and `TEST_PATTERN` prepare the back buffer but no longer refresh the visible image by themselves.

In text mode, `CLEAR` fills every cell with the space character. `CLEAR_COLOR` sets the background color and clears every cell to space.

Frame Presentation and VBLANK

The VDP separates drawing from presentation.

Implicit and explicit presentation

Until the program sends `PRESENT` for the first time, the VDP stays in implicit presentation mode. In that mode, simple programs can see results without a full animation loop:

- Frame-start commands (`CLEAR`, `CLEAR_COLOR`, `TEST_PATTERN`) may refresh the visible image when they complete.
- After a batch of drawing writes finishes, the VDP may copy a completed back buffer to the visible display if the program has not yet switched to explicit presentation.

This is useful for transport smoke tests and other tiny demos. It is not the recommended model for stable animation.

After the first `PRESENT`, the VDP switches to explicit presentation mode. In that mode:

- Drawing commands and frame-start commands modify only the back buffer.
- Only `PRESENT` shows the finished frame on the visible display.
- This avoids briefly showing the previous frame while a new one is being built.

For stable animation, always use explicit presentation: draw one complete frame, send `PRESENT`, then wait for VBLANK.

A program should normally:

1. Draw a complete frame.
2. Send `PRESENT`.
3. Wait for the next VBLANK status bit.
4. Update the simulation.
5. Draw the next frame.

The STATUS port uses bit 0 as a VBLANK flag.

Default port:

```
PEEK @ROMPORT(9), V
```

Meaning:

- If `V = 0`, no VBLANK is pending.
- If bit 0 is set, a display tick has occurred.
- If bit 1 is set, more than four sprites were active on at least one scanline.
- If bit 2 is set, two sprites overlapped on at least one non-transparent pixel.
- Reading STATUS clears the VBLANK bit and the latched sprite status bits until the next event.

The usual loop is:

```
100 POKE @ROMPORT(8), 15
110 PEEK @ROMPORT(9), V
120 IF V = 0 THEN GOTO 110
130 REM Continue with next frame
```

This busy-wait is intentional.

At low IPS, the CPU may not finish a frame before the next display tick. In that case the program naturally runs below 60 frames per second.

At high IPS, the CPU finishes early and spends the extra cycles waiting for VBLANK. This keeps the visible animation speed stable instead of making it faster and faster as the IPS setting increases.

This is the same basic idea used by many early video systems: the CPU is allowed to run, but video animation is synchronized to the display refresh.

Why STATUS is separate from CTRL

The VDP uses a separate STATUS port instead of reading the VBLANK bit from the CTRL port.

This matters because a 4001 ROM port keeps a latch of the last value written to it. If the program writes:

```
POKE @ROMPORT(8), 15
```

then the CTRL port latch contains `15`. If the program immediately reads the same port, it would see the latched command value, not a clean zero/one status bit.

Using ROM9 as a separate read-only STATUS port avoids this problem. The program never writes to ROM9, so the latch stays at zero and the VBLANK bit can be read clearly.

Basic Examples

QuadBasic: clear, draw, and frame loop

High-level equivalent of the POKE examples below. Uses explicit presentation and VBLANK synchronization.

```
10 REM QUADBASIC FRAME LOOP
20 VDP MODE BITMAP
30 VDP CLEAR 0
40 VDP FILLCIRCLE 32, 24, 8, 8
50 VDP FRAME
60 GOTO 30
```

QuadBasic: text HUD

```
10 REM QUADBASIC TEXT HUD
20 VDP MODE TEXT 7, 0
30 VDP CLS
40 VDP PRINT 0, 0, "SCORE 07"
50 VDP PRINT 0, 7, "OK"
60 VDP PRESENT
70 END
```

QuadBasic: tile mode corner stamp

```
10 REM QUADBASIC TILE CORNERS
20 VDP MODE TILE
30 VDP FILLMAP 1
40 VDP TILE 0, 0, 2
50 VDP TILE 7, 5, 2
60 VDP PRESENT
70 END
```

Clear the screen to blue and present (POKE)

```
10 REM CLEAR TO COLOR 12
20 POKE @ROMPORT(8), 4
30 POKE @ROMPORT(7), 12
40 POKE @ROMPORT(8), 15
50 END
```

Plot one pixel at X=20, Y=10

```
10 REM PLOT PIXEL X=20, Y=10, COLOR=8
20 POKE @ROMPORT(8), 5
30 POKE @ROMPORT(7), 1
40 POKE @ROMPORT(7), 4
50 POKE @ROMPORT(7), 0
60 POKE @ROMPORT(7), 10
70 POKE @ROMPORT(7), 8
80 POKE @ROMPORT(8), 15
90 END
```

Explanation:

- X=20 is hexadecimal `14`, so it is sent as `1`, `4`.
- Y=10 is hexadecimal `0A`, so it is sent as `0`, `10`.

Draw a filled circle and wait for VBLANK

```
10 REM FILLED CIRCLE, THEN WAIT FOR DISPLAY TICK
20 POKE @ROMPORT(8), 4
30 POKE @ROMPORT(7), 0
40 POKE @ROMPORT(8), 10
50 POKE @ROMPORT(7), 2
60 POKE @ROMPORT(7), 0
70 POKE @ROMPORT(7), 1
80 POKE @ROMPORT(7), 8
90 POKE @ROMPORT(7), 0
100 POKE @ROMPORT(7), 5
110 POKE @ROMPORT(7), 8
120 POKE @ROMPORT(8), 15
130 PEEK @ROMPORT(9), V
140 IF V = 0 THEN GOTO 130
150 END
```

This draws a filled circle at X=32, Y=24, radius 5, color 8.

Technical Notes

This section describes the behavior of the VDP as seen by a program. It does not describe the internal implementation of Quadium 4004 Workbench.

Resolution and colors

The visible display is 64 pixels wide and 48 pixels high.

Each pixel stores a 4-bit color index, so there are 16 possible color values: 0 to 15.

The VDP uses a fixed default palette based on the PICO-8 color set. Programs should treat colors as indexes, not as RGB values. QuadBasic has no command to change the palette at runtime.

Back buffer and front buffer

The VDP behaves as if it has two image buffers:

- A back buffer, where drawing commands write.
- A front buffer, which is visible to the user.

Most drawing commands modify the back buffer. The `PRESENT` command copies the completed back buffer to the visible front buffer.

In implicit presentation mode, the VDP may also refresh the visible image when certain commands complete or when a batch of drawing writes finishes. In explicit presentation mode, only `PRESENT` performs that copy.

This avoids showing a half-drawn frame while the program is still clearing the screen and drawing the next object.

VRAM addressing and pixel stream

Internally, the VDP stores 3072 pixels. Command 2 (`SET_ADDR`) selects a pixel index from 0 to 3071. After that, each DATA nibble in pixel-stream mode writes one pixel and advances the address. Command 3 (`HOME`) resets the stream address to 0.

Tile mode (Graphic I style)

Tile mode uses an 8x6 viewport of 8x8 cells over the 64x48 display. The backing name table is 16 columns by 8 rows (128 cells).

- Each cell stores a pattern index from 0 to 63.
- The pattern generator stores 64 patterns. Each pattern is 8x8 pixels of 4-bit color indexes.
- Command 11 selects bitmap or tile mode.
- Commands 12 and 14 update the name table.
- Command 2 sets scroll X/Y in pixels. Command 3 resets scroll to the origin.
- Commands 9 and 10 select or reset the pattern-generator write address in tile mode.
- Command 13 uploads one pattern by index.
- In tile mode, the VDP composes the back buffer from the name table and pattern generator when the map, scroll, or `PRESENT` runs, not on every pattern upload.
- `CLEAR` and `CLEAR_COLOR` in tile mode rebuild the screen from pattern 0. `CLEAR_COLOR` also redefines pattern 0 as a solid color.

Text mode

Text mode uses a 16x8 cell grid. Each cell stores a character code instead of a pattern index.

- The ROM font provides 64 fixed glyphs, drawn as 3x6 pixels inside a 4-pixel-wide cell.
- Command 11 selects bitmap, tile, or text mode. Text mode also takes foreground and background colors.
- Commands 12 and 14 update the character grid.
- Command 12 redraws only the touched cell in the back buffer. Full-screen composition still happens on `CLEAR`, `CLEAR_COLOR`, and `PRESENT`.
- `CLEAR` and `CLEAR_COLOR` clear every cell to space.

Use text mode for short HUD labels, scores, and status words. Long tutorials and multi-line prose belong outside the VDP.

Hardware sprites

When sprites are enabled, the VDP keeps 16 sprite entries. Each sprite uses the shared 8x8 pattern generator, draws on top of the current background, and is composited again on `PRESENT`.

- Command 12 updates one sprite in bitmap mode. The visible image is composed on `PRESENT`, not on every attribute write.
- Command 13 defines the 8x8 pattern used by tiles and sprites.
- Sprite index 0 has the highest priority.
- Pattern color 0 is transparent.
- STATUS bit 1 reports more than four sprites on one scanline.
- STATUS bit 2 reports sprite overlap.

Geometry and clipping

Coordinate components are sent as two nibbles each and may represent values from 0 to 255, but only the visible 64x48 area is drawn. Coordinates outside that area are clipped.

For rectangles, width and height are pixel extents from the top-left corner. Outline commands draw a 1-pixel border. Filled commands paint the interior.

Circles use a center point and a radius in pixels. Outline and filled variants use the same parameters.

Sequential data mode

The DATA port can be used as a stream. Certain commands configure what the next DATA nibbles mean.

For example, after a filled circle command, the VDP expects seven DATA nibbles:

- Center X high nibble.
- Center X low nibble.
- Center Y high nibble.
- Center Y low nibble.
- Radius high nibble.
- Radius low nibble.
- Color.

Once all required nibbles have arrived, the command is executed.

STATUS and read-clear behavior

The VBLANK status bit is read-clear.

This means:

1. The display system sets bit 0 when a display tick occurs.
2. The program reads STATUS.
3. The read returns the current bit value.
4. The bit is cleared.
5. It stays clear until the next display tick.

This is useful because the program can detect a new frame boundary without needing interrupts.

CPU speed and animation speed

The IPS selector controls how fast the emulated 4004 executes instructions.

The display refresh is a separate concept. A well-behaved animation should not move faster just because the CPU has more IPS available. Instead, it should draw a frame, present it, and wait for the next VBLANK.

At high IPS, the program spends more time waiting. At low IPS, the program may not be able to keep up. This is expected and is close to how many early video systems were programmed.

Recommended Programming Style

For stable animation:

1. Clear the back buffer.
2. Draw all objects for the current frame.
3. Send `PRESENT`.
4. Poll STATUS until VBLANK is set.
5. Update positions or game state.
6. Repeat.

Example structure with QuadBasic:

```
10 REM FRAME LOOP (QUADBASIC)
20 VDP MODE BITMAP
30 VDP CLEAR 0
40 REM DRAW OBJECTS HERE
50 VDP FRAME
60 REM UPDATE OBJECTS HERE
70 GOTO 30
```

The same loop with POKE:

```
10 REM FRAME LOOP (POKE)
20 POKE @ROMPORT(8), 4
30 POKE @ROMPORT(7), 0
40 REM DRAW OBJECTS HERE
50 POKE @ROMPORT(8), 15
60 PEEK @ROMPORT(9), V
70 IF V = 0 THEN GOTO 60
80 REM UPDATE OBJECTS HERE
90 GOTO 20
```

Avoid presenting the frame in the middle of drawing. Also avoid updating object positions multiple times before waiting for VBLANK, unless the goal is deliberately fast simulation rather than stable visual animation.

In text mode, prefer one or two short status rows instead of filling the screen with prose. Update only the cells that changed when refreshing a score or state label.

The main mental model is simple:

> Draw one complete frame, present it once, wait for the display tick, then move the world forward.

LEGAL NOTICE

> **Legal Notice and Fair Use Disclaimer:**

This project, Quadium 4004 Workbench, is an independent educational tool developed for historical preservation, research, and instructional purposes regarding early computing architectures.

Trademark Acknowledgement: Intel, MCS-4, 4001, 4002, 4003, and 4004 are registered trademarks of Intel Corporation. Texas Instruments, TMS9918, and related VDP marks may be trademarks of Texas Instruments Incorporated. These terms are used herein solely for nominative purposes to describe technical compatibility and historical context, which constitutes "fair use" under intellectual property laws. All references to these products are made for descriptive and compatibility purposes only.

Non-Affiliation: This project is not affiliated with, authorized, sponsored, or endorsed by Intel Corporation or Texas Instruments Incorporated.

Original Documentation: This documentation is an independent work created for Quadium 4004 Workbench. QuadBasic is an original language implementation designed for educational use with the emulated MCS-4 system.

Third-party marks and materials: Intel, MCS-4, 4004, 4001, 4002, and 4003 may be trademarks of Intel Corporation. Texas Instruments, TMS9918, and related VDP marks may be trademarks of Texas Instruments Incorporated. This product is not affiliated with or endorsed by Intel or Texas Instruments. Quadium does not distribute third-party ROM dumps, Intel manuals, Texas Instruments manuals, or datasheets. QuadBasic sample programs supplied with the product are authored by the publisher and transpile to 4004 assembly. Users are responsible for the rights to any files they load into the simulator.

Nature of Simulation: All logic described in this documentation represents an independent software implementation of publicly documented hardware specifications.

Document version: 0.80 (WIP)
Date: May 2026
Language: QuadBasic