Virtus OS Stdlib Service Reference

3,946 words · ~18 min read

---

*VCA-CSA-101 cross-chapter quick-reference handout. Anchors: §11.2 (compiler-side spec) + §12.4-§12.9 (OS-side implementation). *

Purpose: complete signature reference for every standard-library service the Virtus compiler emits calls against and the Virtus OS v1 implements. Print and pin during Lab 11.1 (library-call codegen), Lab 11.3 (Virtus Console end-to-end), Labs 12.1-12.5 (OS implementation + capstone). The compiler trusts this surface to exist; the OS contracts to provide it; **drift between the two is a curriculum bug**.

At a glance

Property	Value
Modules	**9** primary (`Math`, `Memory`, `Output`, `Console`, `Screen`, `GamePad`, `Sound`, `Sys`, `GPIO`) + **2** helpers (`String`, `Array`)
Total services	**~34** services (varies by minor revision)
Calling convention	Left-to-right argument push; void methods return placeholder zero; one word on stack at `return` (per Ch 8 §8.4 + Ch 11 §11.4)
Implementation	OS-resident; called via `jalr` (no `ecall` in v1; CSA-201 introduces the trap mechanism)
Total OS source	**~1,500 lines** of student-authored HLL + ~30 lines bootstrap assembly
Replaces	Ch 11 stub-OS bundle (Lab 11.3 stubs return trivial defaults)
CSA-201 evolution	Adds privilege boundary + `ecall` trap; library list grows by ~30%; mitigations toggleable

---

Calling convention (central)

Every stdlib service obeys the Ch 8 calling convention. **No special-case for OS code; OS code is user code at a different address.**

Property	Specification
Argument passing	Left-to-right onto VM-level stack; callee reads `argument 0..argument m-1`
Implicit `this`	None for stdlib calls (all stdlib services are `function`s, not `method`s)
Return value	One word on top of stack at `return`; void methods leave placeholder zero
Caller pop pattern	`do M.f(...)` discards via `pop temp 0`
Caller-saved state	Per Ch 8 §8.6 (RET + LCL + ARG + THIS + THAT)
Argument count after `call`	`len(args)`. No implicit-this for stdlib

`Virtus.syscall`. Kernel-mediated dispatch (R7.x post-discovery addition)

Added 2026-04-29 per audit cleanup. R7.x silicon-bring-up shipped a single dispatcher entry point that mediates all stdlib calls; per Ch 12 §12.3.2 prose, the dispatcher is structurally a regular Jack function with a 30-branch if-chain. There is no privilege-mode trap or syscall ABI distinct from Jack's calling convention.

Property	Specification
Signature	`Virtus.syscall(num, arg0, arg1) → result`
Source	`stdlib/Virtus.virtus:75-117` (30-branch if-else dispatcher)
Dispatch type	O(N) on N services; ~150 RV32I-Lite instructions per call (avg)
Calling convention	Same Jack convention as every other stdlib service (left-to-right stack push; one-word return)
ABI distinction	**None**. `Virtus.syscall` is a regular Jack function; not an `ecall`-style trap
Failure sentinel	Returns `-1` for invalid `num`

num → service mapping (canonical roster)

`num` range	Service family	Canonical service-num assignments
0	`Math.add` (canonical syscall test entry)	`syscall(0, a, b) → a + b`
1	`Math.multiply`	`syscall(1, a, b) → a * b`
2	`Math.divide`	`syscall(2, a, b) → a / b`
3	`Math.sqrt`	`syscall(3, n, _) → floor(sqrt(n))`
4	`Math.abs`	`syscall(4, n, _) → |n|`
10	`Memory.alloc`	`syscall(10, size, _) → ptr`
11	`Memory.dealloc`	`syscall(11, ptr, _) → 0`
12	`Memory.peek`	`syscall(12, addr, _) → M[addr]`
13	`Memory.poke`	`syscall(13, addr, value) → 0`
20	`Output.printChar`	`syscall(20, ch, _) → 0`
21	`Output.printString`	`syscall(21, str_ptr, _) → 0`
30	`Screen.drawPixel`	`syscall(30, packed_xy, color) → 0` (`packed_xy = (x << 16) | y`)
31	`Screen.setColor`	`syscall(31, color, _) → 0`
32	`Screen.clear`	`syscall(32, _, _) → 0`
40	`GamePad.readButton`	`syscall(40, n, _) → button_state` (renamed from `Keyboard.readButton` per CR2; DS2 12-button MVP bitmap per `GamePad` section below)
41	`GamePad.poll`	`syscall(41, _, _) → button_word` (renamed from `Keyboard.poll` per CR2; raw 16-bit DS2 bitmap)
50	`Sound.play`	`syscall(50, samples_ptr, length) → 0`
60	`Sys.halt`	`syscall(60, _, _)` → does not return
61	`Sys.error`	`syscall(61, code, _)` → does not return
70-79	`String.*` (helpers)	`String.new` / `String.appendChar` / `String.charAt` / `String.length`
80-89	`Array.*` (helpers)	`Array.new` / `Array.dispose`

`num` range	Service family	Canonical service-num assignments
default	(invalid num)	Returns `-1` sentinel

The dispatcher's source is at `stdlib/Virtus.virtus:75-117`; check the actual source for the canonical num assignments at any given commit (the table above reflects the R7.3 canonical roster; future revisions may renumber). Per Ch 12 §12.3.2, the indirection is forward-compatible with CSA-201's `ecall` privileged-mode dispatch, when CSA-201 lands, `Virtus.syscall(num, arg0, arg1)` becomes `ecall` with `num/arg0/arg1` in registers; the user-side call site does not change.

Pedagogical note for Lab 12.X: students implementing a syscall (extending the dispatcher with a new service) are *implementing a function call*, not a privilege-boundary trap. Per Ch 12 §12.3.2's closing pivot. *"when you reach Lab 12.X and find yourself implementing a syscall, you may be surprised that you are just implementing a function call."*

`Math`. Multiply, divide, square root on an ALU that has none

RV32I-Lite has no `mul`, no `div`, no shift instructions. **Every Math service is software.** Cycle costs are the chapter's most-feelable demonstration of the cost of "missing" instructions.

Service	Signature	Returns	Cycle cost	Error semantics
`Math.multiply(a, b)`	`(int, int) → int`	`a × b` (32-bit two's-complement; wraps on overflow)	~1,000 cycles (200 with mask-table opt)	Overflow wraps silently
`Math.divide(a, b)`	`(int, int) → int`	`a ÷ b` (signed; truncation toward zero)	~800 cycles	**Trap on `b = 0`** via system-control halt at `0x8003000C`; **trap on `0x80000000 / -1`** (overflow)
`Math.sqrt(n)`	`(int) → int`	`floor(sqrt(n))` via integer Newton iteration	~5,000 cycles (uses Math.divide internally)	`n < 0` returns `-1` sentinel
`Math.abs(n)`	`(int) → int`	`|n|`	~5 cycles	`Math.abs(0x80000000)` returns `0x80000000` (Java convention; two's-complement is asymmetric)

Algorithm summary:

- **multiply**. Shift-and-add, 32-iteration loop over `b`'s bits; "shift left by k" implemented as `k` self-adds (`add t0, t0, t0`) because RV32I-Lite has no `sll`.
- **divide**. Restoring division, 32-iteration shift-subtract-restore.
- **sqrt**, Newton's method, integer variant; iterate `x = (x + n/x) / 2` until `x_new == x` or `(x_new + 1) == x` (the bouncing-between-consecutive-integers fix).
- **abs**. `if n >= 0: return n; else: return 0 - n`.

CSA-201 forward-pointer. *Every Math service shrinks by 50-500× when the M extension's `mul`/`div` arrive, the gap is the speedup the student personally measures on the same student-silicon bitstream (Tang Primer 25K canonical Phase-1 baseline; Tang Nano 20K advanced-track variant).* CSA-301 extends to `Math.sin` / `Math.cos` / `Math.log` (FP extension required, F/D. Not in CSA-201, only later electives).

`Memory`. Heap allocator over 16 KiB

Free-list allocator with first-fit placement, split-on-allocate, and adjacent-block coalescing on free. Heap region: `0x00010000`. `0x00013FFF`.

Service	Signature	Returns	Notes
`Memory.init()`	`() → void`	-	Initialises free list to one 16,384-byte block at `0x00010000`
`Memory.alloc(size)`	`(int) → int`	Heap pointer or `0` on OOM	First-fit; minimum block 24 bytes; word-aligns; sets sentinel `0xDEADBEEF`
`Memory.dealloc(ptr)`	`(int) → void`	-	Sentinel-checks for double-free; inserts in address order; coalesces with adjacent free neighbors
`Memory.peek(addr)`	`(int) → int`	`M[addr]`	Bare load. Useful for OS-internal bookkeeping; not type-safe
`Memory.poke(addr, value)`	`(int, int) → void`	-	Bare store. Same caveat

Block header (8 bytes): `{size, next}`.

- Allocated blocks: `next = 0xDEADBEEF` (double-free sentinel).
- Free blocks: `next` = pointer to next free block (or `0` if last).
- Payload begins at `header_address + 8`.

Error semantics:

- `alloc(size)` returns `0` on out-of-memory (no exception in v1).
- `dealloc(ptr)` calls `Sys.error("dealloc on non-allocated block")` if sentinel mismatch.

XD-strand attack vectors (exploited in advanced security strand):

- Heap-overlap via forged `next` pointer
- Double-free into small-block list
- Header overflow into next-block metadata

CSA-201 forward-pointer. Adds canary words around payload, segregated free lists, ASLR for heap base; Ch 12 §12.5.4 catalogues the catalogue. CSA-201 §5 extends to bump allocator + slab allocator + tracing GC as performance comparisons.

`Output`. Text output to console

Two services. Text is rendered to the framebuffer's text region as 8×8-pixel glyphs (font supplied as `.bss` constant in the chapter's reference repo).

Service	Signature	Returns	Notes
`Output.printChar(c)`	`(int) → void`	-	Writes one character at current text cursor; advances cursor; wraps at line end; scrolls at screen end
`Output.printString(s)`	`(String) → void`	-	Iterates `String.length` calls of `printChar` (library composition demo)

Argument-count after `call`: `1` for both (no implicit `this`; stdlib functions are static-class).

CSA-201 evolution. Adds `Output.printInt(n)`, `Output.printFloat(f)`, `Output.println()`, formatted output `Output.printf(...)`. CSA-301's GUI library replaces text with proper text-rendering and font-table support.

`Console`. Text-mode tile output (the pedagogically-first I/O path)

Tile-mode character output over the IP Pack's HDMI tile-map peripheral. Logical surface: **80 × 30 character cells** (640 × 480 video output via 8 × 16 glyph ROM), 7-bit ASCII chars, 16-color CGA/EGA palette per cell (4 bits foreground + 4 bits background). AXI4-Lite slave window at MMIO base **`0x80100000`** (per `peripheral-ip-pack/sw/hdmi_demo/hdmi_tile_map.h`).

Why Console comes before Screen in pedagogical sequence. Text output is the first I/O path every running program needs. `Console.printString("hello")` is the smallest possible "the silicon is alive" demonstration. Pixel-mode (`Screen`, §below) requires that the student already have a framebuffer-update model in head; tile-mode lets the student write working programs from Lab 11.3 onward without touching pixel arithmetic at all. CSA-101 introduces Console first, Screen second, and the chapter's bring-up sequencing reflects that.

Service	**Signature**	**Returns**	**Notes**
`Console.init()`	`() → void`	-	Sets cursor to (0, 0); writes default palette (CGA 16-color); enables CTRL bit 0; clears tile-map RAM to space-on-black
`Console.clear()`	`() → void`	-	Fills tile-map RAM with `(0x20 << 8) | current_color` (space char on current fg/bg); resets cursor to (0, 0)
`Console.printChar(ch)`	`(int) → void`	-	Writes ASCII `ch` (low 7 bits) into the cell at current cursor; advances cursor; wraps at column 80; scrolls at row 30; honours `\n` (0x0A) for newline + `\r` (0x0D) for carriage return
`Console.printString(s)`	`(String) → void`	-	Iterates `String.length(s)` calls of `printChar`; library-composition demo
`Console.println(s)`	`(String) → void`	-	`printString(s)` then `printChar(0x0A)`; the convention every later capstone uses
`Console.setCursor(row, col)`	`(int, int) → void`	-	Direct cursor placement; clamps `row ∈ [0..29]`, `col ∈ [0..79]`; out-of-range silently clamped (not trapped)
`Console.setColor(fg, bg)`	`(int, int) → void`	-	Updates current-fg/bg state; subsequent `printChar` uses it; `fg`, `bg` ∈ `[0..15]` palette indices
`Console.scroll()`	`() → void`	-	Shifts all rows up by one; bottom row cleared to space-on-current-color; called automatically on cursor-overflow but exposed for capstone use
`Console.setPalette(idx, rgb565)`	`(int, int) → void`	-	Writes 16-bit RGB565 color into `palette[idx]` register at `0x80100100 + idx*4`; advanced; default palette is the canonical CGA 16-color set so most students never touch this

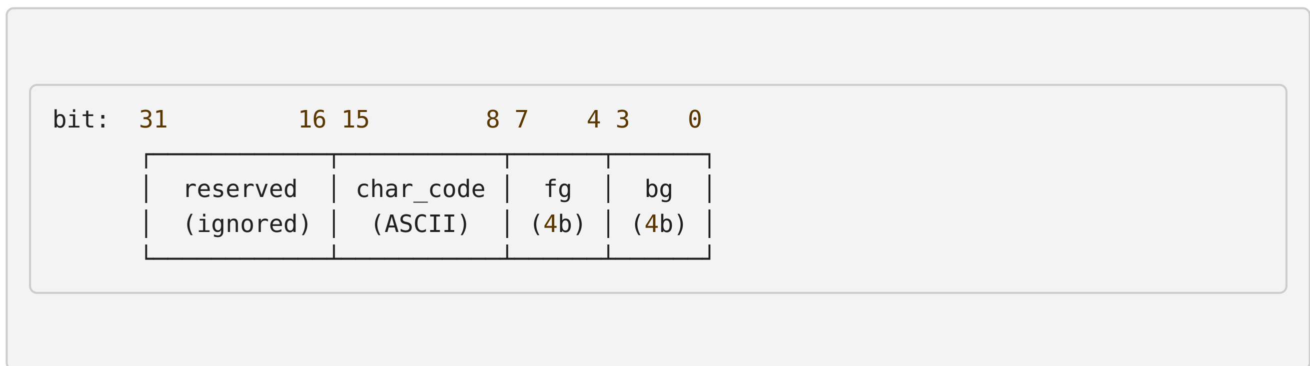
Calling convention. Same Ch 8 left-to-right stack-push + `jalr` + void-returns-zero convention every other stdlib module obeys. **No special-case for Console.** OS code is user code at a different address (per Ch 12 §12.3 + §12.3.2 framing).

Hardware-side memory map (sourced from `peripheral-ip-pack/hdl/hdmi_tile_map/hdmi_tile_map_axi.v` + `sw/hdmi_demo/hdmi_tile_map.h`):

Offset (relative to `0x80100000`)	Width	Name	Direction	Purpose
`+0x000`	32	`CTRL`	R/W	bit 0 = enable; bits 31:1 reserved (read-as-zero)
`+0x004`	32	`STATUS`	R	bit 0 = `in_active` (asserted while video region scanning); bits 31:1 reserved
`+0x008`	32	`FRAME`	R	Frame counter (incremented once per vblank in `clk_core` domain)
`+0x100..+0x13C`	32 (× 16 entries)	`palette[0..15]`	R/W	Low 16 bits = RGB565; high 16 bits ignored. Default = CGA 16-color (palette[0] = black; palette[15] = white)
`+0x4000..+0x6FFC`	32 (× 2400 entries)	`tile_map_ram[0..2399]`	R/W	`cell_index = row × 80 + col` (row ∈ [0..29], col ∈ [0..79]); `cell_value = (char_code`

Offset (relative to `0x80100000`)	Width	Name	Direction	Purpose
				`<< 8) | (fg << 4) | bg`; `char_code` = 7-bit ASCII (low 7 bits used by `char_rom`); `fg`, `bg` = 4-bit palette indices

Cell encoding (central for Lab 11.3 + Lab 12.X capstones writing direct cell values):



16-color palette (CGA/EGA canonical). Default values written at `Console.init` time:

idx	Name	RGB565	idx	Name	RGB565
0	Black	`0x0000`	8	Dark gray	`0x52AA`
1	Blue	`0x0010`	9	Light blue	`0x435F`
2	Green	`0x0400`	10	Light green	`0x07E6`
3	Cyan	`0x0410`	11	Light cyan	`0x07FF`
4	Red	`0x8000`	12	Light red	`0xF9C7`
5	Magenta	`0x8010`	13	Light magenta	`0xF81F`
6	Brown	`0x8200`	14	Yellow	`0xFFE6`
7	Light gray	`0xC638`	15	White	`0xFFFF`

Cross-chapter cross-cuts:

- **Ch 5 §5.7** (Architecture Comparison Sidebar. Heterogeneous-multi-processor SoCs), Console + Screen are the two HDMI I/O paths exemplifying CPU → AXI manifold → specialized peripherals; Console targets the tile-map peripheral, Screen targets the framebuffer peripheral. Both are AXI4-Lite slaves; both decode against the §5.7.4 5-signal handshake; both honour the §5.7.5 16-bit truncation memory-map limit (Console's `0x80100000+offset` window is reachable since the high bit triggers MMIO routing per §5.7.1 memory map).
- **Ch 11 §11.2** (compiler-side stdlib emit spec). Compiler emits `call Console.printString 1` when student writes `print("hello")` in `.virtus` source (per Ch 11 §11.5 string-handling convention). Console method signatures slot into the compiler's known-method registry alongside the other 7 modules.
- **Ch 12 §12.6**, OS-side `Console.virtus` implementation walkthrough; cell-encoding pack/unpack helpers; the cursor/scroll state machine; default-palette initialisation.
- **Lab 11.3** ("Virtus Console end-to-end on HDMI"). Already named for this peripheral; the lab is now explicitly Console-mediated rather than Screen-mediated. Print and pin this handout.
- **Lab 12.3** ("Implement Screen lib"). May want sibling **Lab 12.3a** ("Implement Console lib") that lands first pedagogically (text-out before pixel-out).
- **Lab 11.3 + Lab 12.X capstones**, Console.printString is the canonical "is the silicon alive?" check students run as Step 1 of capstone bring-up. Sibling to the §5.9.1 bring-up checklist's HDMI handshake step.

CSA-201 evolution. Adds `Console.printInt(n)`, `Console.printHex(n)`, formatted output `Console.printf(...)`. The CGA/EGA palette opens to a full 6-6-5 palette-RAM model (256-color or beyond). Tile-map RAM expands to a paged-tile model (multiple foreground/background tile sets selectable per cell). Most CSA-201 driver-track work targets the CSA-101 tile-map peripheral as the substrate.

`Screen`. Pixels, lines, rectangles, circles

Pixel-mode counterpart to `Console` (§above). Console is text/tile-mode @ MMIO `0x80100000`; Screen is pixel-mode @ MMIO `0x80000000`. The two peripherals coexist; capstones may use either or both.

Drawing primitives over the IP Pack's HDMI framebuffer. Logical surface: **96 × 64 pixels @ 16 bits-per-pixel** (RGB565); virtual framebuffer in OS `.bss` (12 KiB); IP Pack physical framebuffer at `0x80000000`.

Service	Signature	Returns	Notes
`Screen.init()`	`() → void`	-	Zeroes virtual framebuffer; initialises dirty-row tracking
`Screen.clear()`	`() → void`	-	Fills virtual framebuffer with current background color (~1,500 `sw` instructions)
`Screen.setColor(c)`	`(int) → void`	-	Updates foreground-color register; subsequent `drawPixel(x, y, sentinel)` uses it
`Screen.drawPixel(x, y, color)`	`(int, int, int) → void`	-	Read-modify-write on containing 32-bit word; out-of-bounds silently dropped
`Screen.drawLine(x1, y1, x2, y2, color)`	`(int, int, int, int, int) → void`	-	Bresenham midpoint algorithm; correct in all 8 octants; integer-only
`Screen.drawRectangle(x, y, w, h, color)`	`(int, int, int, int, int) → void`	-	4 calls to `drawLine` (top/bottom/left/right edges)
`Screen.drawCircle(cx, cy, r, color)`	`(int, int, int, int) → void`	-	Bresenham midpoint circle; 8-octant symmetry

Color encoding (RGB565):

- bits 15:11 = Red (5 bits)
- bits 10:5 = Green (6 bits)
- bits 4:0 = Blue (5 bits)

Read-modify-write hazard (Ch 12 §12.6.5): two pixels per 32-bit word means every `drawPixel` is an RMW; under preemption (CSA-201) this becomes a torn-write hazard requiring atomic primitives. **Virtus OS v1 has no concurrency, so the hazard is a forward-pointer, not a present problem.**

CSA-201 evolution. Adds `Screen.fillRectangle`, `Screen.drawTriangle`, `Screen.blit(src, dst, w, h)` for sprite blits, and the atomic-write primitives that close the RMW hazard. CSA-301's `con-101` retro-FPU course adds floating-point support for proper geometry math.

`GamePad`: DS2 controller polling

Renamed from `Keyboard`. The service interface and decoder pattern are identical; the canonical hardware target is the DS2 controller (PMOD_DS2x2 ships with the lab kit; SNES adapter scope was dropped). The 12-button MVP mapping below is a strict subset of the DS2's richer state (analog sticks + analog pressure + 8 face buttons); forward-stretch labs in CSA-201 + CON-101 expose the richer state. The original 8-button SNES set (B/Y/Select/Start/Up/Down/Left/Right) maps as a strict subset of the 12-button surface; SNES historical hardware is anchored in the Ch 5 §5.7 Architecture Comparison Sidebar (8-bit/PPU/PSG era), where it stays as historical pedagogy. The PS/2 `Keyboard` service is documented separately.

Polled (not interrupt-driven in v1) at the IP Pack's dedicated GamePad decoder at AXI4-Lite slave window **`0x80140000`** (per Findings §25.2 canonical address map; M0.5 deferred per A7. Base reserved). Bitmap layout matches the DS2 gamepad's 12-button MVP roster (physical DS2 has 8 face buttons + 4 directional + analog L1/L2/R1/R2. Full surface deferred to CSA-201 driver track). *(Pre-Audit-#2 placeholder address `0x80020000` superseded 2026-04-30 by canonical manifold-aligned address; see also `handouts/cross-chapter-axi-manifold-base-addresses.md` for the full MMIO map.)*

Service	Signature	Returns	Notes
`GamePad.init()`	`() → void`	-	Zeroes 4-frame state-history buffer for software debounce
`GamePad.readButton(n)`	`(int) → int`	`1` if button `n` pressed, `0` otherwise	Single-bit query; `n` ∈ `0..15`
`GamePad.poll()`	`() → int`	16-bit button bitmap (raw)	`lw` from `0x80140000` (canonical per Findings §25.2; M0.5 deferred per A7)
`GamePad.pollDebounced()`	`() → int`	16-bit rising-edge bitmap	4-frame state-machine; only emits press events on 0→1 transition with prior 3 frames = 0

DS2 button bit layout (12-button MVP mapping):

Bit	Button	Bit	Button
0	× (Cross / "B" in SNES analog)	8	△ (Triangle / "X" in SNES analog)
1	□ (Square / "Y" in SNES analog)	9	○ (Circle / "A" in SNES analog)
2	Select	10	L1
3	Start	11	R1
4	Up	12	L2 (analog; 0/1 in MVP)
5	Down	13	R2 (analog; 0/1 in MVP)
6	Left	14	L3 (analog-stick click)
7	Right	15	R3 (analog-stick click)

SNES historical mapping (for capstones porting retro game-controller protocols): the 8-button SNES set (B/Y/Select/Start/Up/Down/Left/Right) maps directly to bits 0-7 of the DS2 surface above (DS2 face buttons × / □ on bits 0/1; D-pad on bits 4-7; Select / Start on bits 2/3). DS2 bits 8-15 expose the richer state SNES did not have. Per `feedback_hardware_superset_mapping.md`: provide a mapping/adapter at the firmware/IP-Pack layer rather than rewriting the curriculum to match the controller's richer surface.

CSA-201 evolution. Adds interrupt-driven gamepad (`GamePad.IRQ_handler`) that pushes events into a ring buffer; CSA-201's driver-track lab opens the GamePad decoder itself (Verilog reimplementation from DS2 protocol datasheet). Adds analog-stick + analog-pressure exposure once 12-button MVP is operational. CSA-301 adds USB-HID gamepad + USB-HID keyboard support.

`Sound`: VCP coprocessor commanding

Heterogeneous-multi-processor service. The Virtus Co-Processor (VCP) runs concurrently with the main RV32I-Lite CPU; communication via shared memory at `0x80010000` plus an IRQ line.

Service	Signature	Returns	Notes
`Sound.init()`	`() → void`	-	Initialises VCP shared-memory region; registers IRQ handler
`Sound.play(samples_ptr, length)`	`(int, int) → void`	-	Writes 6 words to VCP shared memory; VCP starts playing within 1 cycle; main CPU returns immediately
`Sound.IRQ_handler()`	`() → void`	-	(Internal. Wired by `crt0.S`) Refills sample buffer on VCP underrun; supports double-buffered streaming

Sample format: 16-bit signed PCM at 22 kHz.

Cycle-counter measurement. Audio is **free**, with-audio cycles equal without-audio cycles within measurement noise. *This is the property that makes coprocessors valuable* in production embedded silicon. Lab 12.4 Part B confirms.

Shared-memory region (8 fields × 4 bytes):

Offset	Field	Direction
`0x80010000`	`samples_ptr`	CPU → VCP
`0x80010004`	`length`	CPU → VCP
`0x80010008`	`read_position`	VCP → CPU
`0x8001000C`	`go`	CPU → VCP
`0x80010010`	`irq_status`	VCP → CPU
`0x80010014`	`irq_ack`	CPU → VCP
`0x80010018`	`repeat`	CPU → VCP
`0x8001001C`	`volume`	CPU → VCP

Bootstrap-time microcode load (informational; not part of the Sound API). The VCP MMIO peripheral region is 512 B (`0x80010000`-`0x800101FF`); the 32 B shared-memory register file above occupies `+0x000..+0x01F`, a 32 B Control Region (CPU → VCP `local_ram[0x00..0x1F]` mirror. Industry-pattern doorbell + scratchpad / NVMe submission queue / NIC descriptor ring; cross-cut #1 hybrid C+B confirmed 2026-04-28) occupies `+0x020..+0x03F`, the 256 B microcode RAM occupies `+0x100..+0x1FF`, and `+0x040..+0x0FF` is reserved. CPU writes microcode bytes to the microcode-RAM window during boot before issuing the first `go`; this is one-time bootstrap done by `crt0.S`, not

by `Sound.play`. CPU writes `Sound.play` parameters (samples_ptr / length / volume / repeat) to the Control Region; HDL mirrors into VCP local data RAM; microcode reads with normal `ld`. See `virtus-peripheral-ip-pack/docs/vcp-design-memo.md` §5 + `vca-csa-101/docs/bus-protocol.md` *VCP MMIO sub-map*.

CSA-201 evolution. Adds `Sound.stop`, `Sound.setVolume`, multi-channel mixing (VCP microcode rewrite required); `con-101` adds full-band DSP via the Virtus Console retro-FPU. CSA-301 (advanced electives) explores multi-coprocessor offload patterns.

`Sys`. System control

System-level utilities. `Sys.init` is the program entry point called by the bootstrap `_start`.

Service	Signature	Returns	Notes
`Sys.init()`	`() → void`	-	Performs final OS-level setup (currently nothing in v1; reserved for CSA-201's expansion); calls `Main.main`
`Sys.halt()`	`() → void`	(never returns)	Writes any value to system-control halt register at `0x8003000C`; FPGA freezes CPU clock
`Sys.wait(ms)`	`(int) → void`	-	Busy-loop based on cycle counter at `0x80030000`; ~27,000 cycles per ms at 27 MHz
`Sys.error(msg_str)`	`(String) → void`	(never returns)	Prints `msg_str` via `Output.printString`; halts via `Sys.halt`

Bootstrap sequence (Ch 12 §12.10.1):

1. CPU resets at `0x00000000`.
2. `_start` initialises `sp`, segment-base registers, BSS.
3. Library `init` calls in dependency order: `Math.init`, `Memory.init`, `Console.init`, `Screen.init`, `GamePad.init`, `Sound.init`, `GPIO.init` (renamed from `Keyboard.init` per CR2; expanded to canonical 9-primary-module init order per CR1/CR2/CR3 baseline).
4. Wires IRQ vector at `_irq_vector`.
5. Calls `Sys.init`.
6. `Sys.init` calls `Main.main`.

7. `Main.main` runs the user's program.
8. `Main.main` returns (or calls `Sys.halt`).
9. Control returns to `_start`'s `_halt:` label.
10. `Sys.halt` writes the system-control register; CPU clock freezes.

CSA-201 evolution. Adds `Sys.fork`, `Sys.exec`, `Sys.exit` (process-management; requires preemption + scheduler); `Sys.error` becomes structured exception with stack-unwind. `ecall` mechanism replaces direct-call into `Sys` services for the user/supervisor split.

`GPIO`. General-purpose I/O escape hatch

*GPIO is the canonical escape hatch for student capstones that need to drive arbitrary external hardware (LEDs, switches, sensors, breadboard-prototyped peripherals) without authoring a dedicated stdlib service per device. AXI4-Lite slave window at MMIO base **`0x80130000`**, 16-bit pin width per silicon default (`N_PINS=16` parameter in `gpio_axi.v`; M0.5 widening to 24/32-bit is a 1-line parameter change per HDL header).*

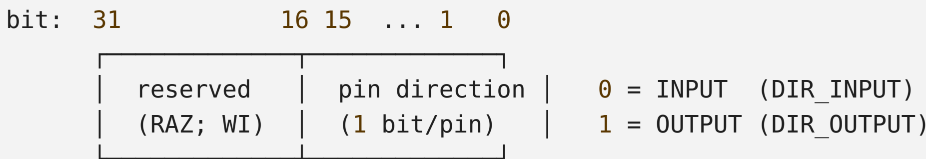
Service	Signature	Returns	Notes
`GPIO.read(pin)`	`(int) → int`	`1` if pin's IN bit is high, `0` otherwise	Reads bit `pin` of `IN` register at `0x80130008`; `pin ∈ [0..15]`; out-of-range returns `0` (not trapped)
`GPIO.write(pin, value)`	`(int, int) → void`	-	Writes bit `pin` of `OUT` register at `0x80130004`; only takes effect when `DIR[pin] = 1` (output); `value ∈ {0, 1}`; non-zero treated as `1`
`GPIO.setDirection(pin, dir)`	`(int, int) → void`	-	Writes bit `pin` of `DIR` register at `0x80130000`; `dir = 0` → input, `dir = 1` → output; default (post-init) is all-input
`GPIO.pollEdge(pin)`	`(int) → int`	`1` if edge fired since last poll, `0` otherwise; ALSO clears the sticky bit (W1C)	Reads bit `pin` of `INT_STATUS` register at `0x8013000C`, returns it, then writes a `1` to clear (write-1-to-clear convention per HDL)
`GPIO.init()`	`() → void`	-	Sets `DIR` to all-input (defensive default; prevents hot-plugged peripherals from being driven before student configures direction); clears `INT_STATUS`

Calling convention. Same Ch 8 left-to-right stack-push + `jalr` + void-returns-zero convention every other stdlib module obeys. **No special-case for GPIO.** OS code is user code at a different address (per Ch 12 §12.3 + §12.3.2 framing).

Hardware-side memory map:

Offset (relative to `0x80130000`)	**Width**	**Name**	**Direction**	**Purpose**
`+0x000`	32 (low 16 used; bits 31:16 reserved)	`DIR`	R/W	Per-pin direction; bit `n` = direction of pin `n` (0 = input, 1 = output). Reset value = `0x00000000` (all-input by default)
`+0x004`	32 (low 16 used)	`OUT`	R/W	Per-pin output value; drives pin `n` when `DIR[n] = 1`. Reset value = `0x00000000`
`+0x008`	32 (low 16 used)	`IN`	R	Per-pin sampled input value (synchronized through 2 flip-flops to cross from external clock domain)
`+0x00C`	32 (low 16 used)	`INT_STATUS`	R/W1C	Per-pin edge-trigger sticky flag; bit `n` = 1 if edge detected on pin `n` since last clear; **write 1 to clear** the bit (W1C convention); ORed across all bits drives `irq` output to CPU (M0-2 stretch; **`INT_ENABLE` per-pin masking deferred to M0.5** per HDL header)

Direction encoding (`DIR` register; per-pin):



Helper macro: `gpio_pin_mask(pin) = 1u << pin`. Used to construct read-modify-write masks for setting/clearing individual bits across `DIR` / `OUT` / `INT_STATUS`.

Cross-chapter cross-cuts:

- **Ch 5 §5.7** (Architecture Comparison Sidebar. Heterogeneous-multi-processor SoCs), GPIO is the canonical *escape hatch* path complementing the dedicated I/O paths (Console + Screen + GamePad + Sound). Where the dedicated services exist for known peripherals (HDMI tile-map, HDMI framebuffer, DS2 gamepad, VCP audio), GPIO covers the open set: any peripheral the student wants to drive that doesn't have a dedicated service yet. The pattern matches RP2040 PIO + ESP32 IOMUX + every modern MCU's GPIO bank.
- **Ch 11 §11.2** (compiler-side stdlib emit spec). Compiler emits `call GPIO.write 2` when student writes `GPIO.write(pin, value)` in `.virtus` source. GPIO method signatures slot into the compiler's known-method registry alongside the other 10 stdlib modules.
- **Ch 12**, OS-side `GPIO.virtus` implementation walkthrough; `setDirection` defensive-default-to-input rationale; `pollEdge` W1C sticky-bit handshake; the 2-flip-flop synchronizer pattern (`IN` register's metastability protection. Pedagogically interesting cross-cut to Ch 3's flip-flop work).
- **Lab 11.3** ("Virtus Console end-to-end on HDMI"). Capstone bring-up MAY want a GPIO-mediated peripheral exercise (e.g., a single LED toggled by a button via `GPIO.read` + `GPIO.write` + `GPIO.setDirection`) as a "is the silicon alive?" smoke-test sibling to `Console.printString`.
- **Lab 12.5** ("Capstone: Ship the Virtus Console"). Capstones extending the Virtus Console with custom external hardware (a breadboard-prototyped 7-segment display, a piezo buzzer beyond the VCP, a sensor mat) use GPIO as the access mechanism. The stdlib roster makes this a one-line call rather than a per-capstone HDL build.

- `cross-chapter-prerequisite-map.md`, GPIO as alternate access path complements the existing CSA-101 internal-runtime dependency chain. Worth noting as a "side-channel" entry in the chapter dependency chain.

CSA-201 evolution. Adds `INT_ENABLE` per-pin masking register at `+0x010` (deferred from M0-2 per HDL header note); adds `GPIO.installInterruptHandler(pin, handler_addr)` that registers a handler called from `Sys.IRQ_dispatch` when `INT_STATUS[pin]` fires. Widens `N_PINS` parameter from 16 → 24 or 32 (1-line HDL change). **GPIO becomes the canonical example of a stdlib service that needs careful permission gating once CSA-201 introduces the U/S-mode privilege boundary**. Driving a pin that's wired to a power supply or a destructive output (motor controller, high-current LED array, irreversible-write peripheral) without permission gating is a real safety hazard. Students who care about Lab 12.5 capstone safety care about CSA-201's GPIO trap path: `ecall` traps to S-mode, the kernel checks the per-process GPIO permission bitmap, and either dispatches `GPIO.write` or returns -1. The `Sys.error` channel from §12.3.2 is what carries the failure case.

`String`. Heap-allocated immutable-ish strings

Helper module. Strings are objects with a `length` field, `maxLength` field, and flat character buffer. The compiler relies on `String.appendChar` for string-literal expansion (Ch 11 §11.5).

Service	Signature	Returns	Notes
`String.new(maxLength)`	`(int) → String`	New String object	`Memory.alloc`-backed; sets length=0
`String.dispose(s)`	`(String) → void`	-	`Memory.dealloc`
`String.length(s)`	`(String) → int`	Current character count	
`String.charAt(s, j)`	`(String, int) → int`	ASCII character at index `j`	
`String.setCharAt(s, j, c)`	`(String, int, int) → void`	-	
`String.appendChar(s, c)`	`(String, int) → String`	`s` (returns receiver)	**Returns receiver to enable method chaining**. See Ch 11 §11.5.1 for compiler's chained-emission discipline
`String.eraseLastChar(s)`	`(String) → void`	-	
`String.intValue(s)`	`(String) → int`	Parses leading decimal digits	
`String.setInt(s, i)`	`(String, int) → void`	-	Mutates `s` to decimal representation of `i`

Compiler relies on the contract: `String.appendChar` *must* return its receiver. *If the OS implements it as `void`, the compiler's chained-appendChar emission for string literals fails.*

13-character literal "hello, world!" emits ~27 VM commands (one `String.new`, 13 `String.appendChar` chain). See Ch 11 §11.5 + the bloat reckoning at §11.9.

CSA-201 evolution. Strings become **interned** via constant-pool optimisation in the compiler; production-runtime equivalent of Java's `String.intern()`. Reduces 13-char literal emission from ~27 VM commands to 1 (`push constant <interned-pointer>`).

`Array`. Heap-allocated fixed-size arrays

Helper module. Thin wrappers over `Memory.alloc` / `Memory.dealloc`.

Service	Signature	Returns	Notes
`Array.new(size)`	`(int) → Array`	Heap-allocated array of `size` words	`Memory.alloc(size * 4)` underneath
`Array.dispose(a)`	`(Array) → void`	-	`Memory.dealloc(a)`

Element access is via VM `that` segment indexing. `let arr = Array.new(10); let arr[3] = 42;` translates to `pop pointer 1; pop that 0` per Ch 7 §7.6.3 idioms.

CSA-201 evolution. Adds `Array.length`, `Array.copy`, `Array.fill`, bounds checking. CSA-301 introduces `List`, `Map`, `Set` as standard collections.

Forward-compatibility note (CSA-201 / CSA-301 / con-101)

The 14-service v1 surface from Ch 11 §11.2 is the **minimum** the curriculum supports. CSA-201's Virtus OS v2 grows by ~30%, mostly in two directions:

Direction	Why	Examples
Privilege-separated services	`ecall` + supervisor mode	`Sys.fork` / `Sys.exec` / process-isolated `Memory.alloc`
Higher-level abstractions	Richer language tier	`Output.printf` / `Math.sin` (FP) / `String.intern` / `List.add`

The compiler does not need to change for most of these, the standard-library registry (Ch 11 §11.2.1) is just a dictionary the student extends. *Stable IR + extensible library = the architecture every modern language ecosystem ships.*

con-101's Virtus Console retro-FPU introduces F-extension floating-point support; `Math.sin`, `Math.cos`, `Math.log` arrive there. **CSA-301's GUI library** introduces window/widget services on top of `Screen`; the structure is `Screen` + retained-mode tree of objects + event dispatch.

OS-side implementation size budget

Library	Approximate size (RV32I-Lite instructions)	Lab
`Math`	~120	12.1
`Memory`	~180	12.2
`String`	~150	(paired w/ 12.2)
`Array`	~30	(paired w/ 12.2)
`Output`	~60 (font glyph blit)	(paired w/ 12.3)
`Screen`	~220	12.3
`Keyboard`	~80	12.4 (Part A)
`Sound`	~120 (+ ~50 VCP microcode)	12.4 (Part B)
`Sys`	~60	(init in §12.10)
Total	**~1,020 RV32I-Lite instructions**	
Plus bootstrap `crt0.S`	~30	(supplied)
Grand total	**~1,050 instructions / ~1,500 source lines**	

A student reading at five lines per minute can read the whole OS in an afternoon. *This is the smallest scale at which a working operating system can exist.*

Where to read more

- **Ch 11** *Compiler III: OS-Aware Compilation*, §11.2 (the spec table; the compiler-side contract); §11.3 (library-call codegen); §11.5 (String.appendChar chaining).
- **Ch 12** *Virtus OS: Math/Memory/I-O/Screen/Keyboard*, §12.1 (what an OS *is* at this scale + what it deliberately is not); §12.2 (memory map); §12.4 (Math); §12.5 (Memory); §12.6 (Screen); §12.7 (Keyboard); §12.8 (Sound + VCP); §12.9 (String/ Array); §12.10 (runtime image + bootstrap).
- **Ch 8 §8.4**. Calling convention every stdlib service obeys.
- **Findings §7.1** + **§16**, RV32I-Lite spec + 8-segment naming.
- **Findings §23**, VCP architecture (the coprocessor `Sound` commands).
- **N2T Project 12**, the structural parallel for the OS-side implementation.

© Virtus Cyber Academy. Generated 2026-05-08.