IP-Pack Interaction Patterns: Doorbell, Scratchpad, Descriptor

1,916 words · ~9 min read

---

*VCA-CSA-101 cross-chapter quick-reference handout. Anchors: §25.4 (canonical address map + 3-pattern overview); §5.7 (Memory-Mapped I/O); §12.6 (Driver-Layer OS Services). *

Purpose: the three structurally-distinct CPU-IP interaction patterns every Virtus Console student encounters at least once across CSA-101 Chapters 5 / 11 / 12 and the M0-2 IP Pack labs. Print and pin during Lab 5.4 (Virtus Console bring-up), Lab 11.3 (HDMI end-to-end), Lab 12.4 (Console drawing capstone), Lab 12.5 (audio-capstone silicon-cert). Each pattern names *when* to reach for it, *why* it exists, *what its tradeoffs are*, and *which IP-Pack peripherals exemplify it*.

At a glance

Pattern	Latency	CPU work per transaction	Best for
Doorbell register	One AXI cycle (write); polling cost variable on read side	One `sw` to a single MMIO offset	Sparse, low-priority events (vsync; buffer-half-empty; edge interrupts)
Scratchpad memory	One AXI cycle per word	One `sw` per word; many writes amortise the handshake	Bulk data transfer where the IP consumes at its own clock domain (tile-map fills; sample buffers; regfiles)
Descriptor-ring	One AXI cycle to advance the head pointer; IP autonomous after	Build descriptor in scratchpad, then ring the head doorbell	Streaming workloads where CPU and IP cooperate on a queue (audio playback; frame DMA; future VCP integration)

The three patterns compose. A descriptor-ring **uses** a scratchpad (for the descriptor entries themselves) **and** a doorbell (to advance the head pointer). A scratchpad-only design is the simplest case; a doorbell-only design is the second simplest; a descriptor-ring is the most complex and the most production-shaped.

Pattern 1: Doorbell register

Shape. Single bit (or single 32-bit word) at a fixed MMIO offset. CPU writes a `1` to "ring the doorbell"; the IP reacts. The IP also exposes a status bit the CPU can read to know whether the IP has completed work (or has new work pending).

Mechanism.

- CPU side: `sw t0, doorbell_offset(base_reg)`. One AXI4-Lite write transaction, one cycle on the manifold's combinational AWREADY canonical pattern.
- IP side: synchronous edge detect on the doorbell bit. The IP latches "doorbell rang" into an internal flag; processes the event on its own clock domain; clears the flag (or not. Depends on whether the doorbell is sticky).
- Reverse direction (IP → CPU): the IP raises an MMIO-readable status bit. CPU polls via `lw` or (when CSA-201 introduces interrupts) wires the bit into the trap-vector dispatch.

Where it lives in the IP Pack.

- **HDMI vsync** at `0x80100020`, IP raises this bit at the end of each 60 Hz frame; CPU polls (or, in CSA-201, traps) to know "the framebuffer is currently in the visible-back portion; safe to write without tearing."
- **Audio buffer-half-empty** at `0x80110010`, IP raises this bit when the playback ring buffer drains past 50%; CPU services by enqueueing more samples.
- **GPIO edge interrupts** at `0x8013000C` (`INT_STATUS` register). `W1C` semantics: read returns "1 if edge fired since last poll"; write-1 clears. Per pin, on rising/falling edge configurable in `INT_CONFIG`.
- **VCP doorbell** (forward-pointer; M0.5 integration). Main CPU rings to hand off a microcode job; VCP rings back with completion.

Tradeoffs.

- **Pro:** simplest mechanism in the kit. One MMIO offset; one AXI cycle; two integers in the IP HDL (the doorbell bit and the edge-detect flag). Combinational AWREADY canonical pattern handles it natively.

- **Pro:** maps cleanly to interrupts in CSA-201. The CSA-101 polling path becomes an `ecall`-driven trap path with no CPU-side source change.
- **Con:** polling cost. Without interrupts, the CPU spends cycles in a `Sys.wait` busy-loop or interleaves polling into the main loop. For sparse events (vsync at 60 Hz; gamepad presses at human speed) this is fine; for rapid events (audio at 22 kHz sample rate) it is not.
- **Con:** no payload. The doorbell carries one bit (or one word); any larger data must go through scratchpad or descriptor-ring.

When to reach for it. When the event is sparse, the payload is empty (or tiny), and you are fine with the CPU-side loop polling, or when you are setting up the future-CSA-201-interrupt infrastructure and want a working CSA-101 placeholder.

Pattern 2: Scratchpad memory

Shape. A region of CPU-readable / CPU-writable memory inside the IP, accessible via AXI4-Lite as a contiguous address window. The IP consumes the memory's contents on its own clock domain; the CPU writes via plain `sw` instructions. There is no explicit "go" signal, the IP's state machine reads the scratchpad continuously.

Mechanism.

- CPU side: a sequence of `sw` instructions targeting `base + 0x000`, `base + 0x004`, `base + 0x008`, ..., each one AXI cycle. The combinational AWREADY canonical pattern handles each in one cycle, so the per-word cost is fixed.
- IP side: the IP holds an internal RAM (BRAM tile or distributed-RAM block) addressed by its own internal counter or state machine. The AXI4-Lite write port and the IP's read port are arbitrated either by a dual-port BRAM (most cases) or by a single-port BRAM with read/write multiplexing.
- The IP's state machine never blocks on the CPU; the scratchpad's contents are simply *whatever the CPU has most recently written*. If the CPU is mid-update, the IP may read transient stale values, that is the IP's problem to handle (typically by double-buffering or by latching at frame boundaries).

Where it lives in the IP Pack.

- **HDMI tile-map RAM** at `0x80100000..0x80100960`, 80×30 = 2400 character cells; CPU writes ASCII codes + palette indices; HDMI tile fetcher reads on every horizontal scan line. Console.printChar is `sw` to one of these offsets.

- **Audio sample buffer** at `0x80110100..0x801101FF`, 256-sample circular buffer; CPU enqueues 8-bit PCM samples; PWM-RC peripheral reads at 22 kHz on its own clock.
- **GPIO regfile** at `0x80130000..0x8013000C`, DIR / OUT / IN / INT_STATUS registers; CPU writes DIR + OUT, reads IN + INT_STATUS. The "scratchpad" framing is loose here (only 4 registers) but the structural pattern is identical.
- **HDMI palette** at `0x80100100..0x8010013F`, 16 RGB565 entries; CPU writes once at boot; HDMI tile fetcher reads to colour each cell.

Tradeoffs.

- **Pro:** amortises the AXI handshake across many writes. Once the CPU is in a `sw` loop, the per-word cost is the AXI minimum (one cycle).
- **Pro:** the IP can read at its own rate, asynchronously from CPU writes. Clock-domain crossing is handled by the BRAM tile's natural dual-port semantics.
- **Pro:** maps cleanly to descriptor-ring extension. A scratchpad becomes a descriptor-ring once the CPU stamps "head" and "tail" pointers in agreed-upon offsets.
- **Con:** no notion of "the IP has consumed this." The CPU can't tell when the audio peripheral has finished playing a buffer without adding a doorbell-pattern status register.
- **Con:** synchronisation hazards. If the CPU writes a half-updated tile-map cell mid-frame, HDMI may render a glitch frame. Mitigation: write at vsync (doorbell-gated), or double-buffer (extra BRAM).

When to reach for it. When you have bulk data the IP needs to read at its own clock domain, and you don't need the CPU to know exactly when the IP has finished. Most CSA-101 IP-Pack uses fall here.

Pattern 3: Descriptor-ring

Shape. A circular buffer of fixed-size descriptor entries in scratchpad memory, plus a *head* pointer the CPU advances and a *tail* pointer the IP advances. Each descriptor carries the parameters for one work unit (e.g., "play this audio buffer at this volume for this many samples"). The CPU enqueues by writing a descriptor and bumping head; the IP dequeues by reading the descriptor at tail and bumping tail.

Mechanism.

- Ring layout: `N` descriptor slots, each 16 or 32 bytes, in a contiguous scratchpad region. Two MMIO registers hold `head` and `tail`. When `head == tail`, the ring is empty; when `(head + 1) mod N == tail`, the ring is full.

- CPU side: builds a descriptor at `descriptor_base + (head * descriptor_size)`; performs one `sw` to advance the head register (this is the doorbell that wakes the IP). Optionally polls "tail caught up to head" to know when the IP has drained the queue.
- IP side: reads the descriptor at `descriptor_base + (tail * descriptor_size)`; processes the work unit (e.g., reads audio samples from the buffer the descriptor points at); advances tail once done.
- The pattern *composes the prior two*: scratchpad memory holds the descriptors; doorbell signals advance the head; doorbell signals on the IP side advance the tail.

Where it lives in the IP Pack.

- **Audio playback queue** at `0x80110100..0x801101FF` (descriptors) + `0x80110200` (head) + `0x80110204` (tail). Audio capstone Lab 12.5 uses this for streaming audio without per-sample CPU service. CPU stages 4-8 descriptor entries; IP plays each in order.
- **HDMI frame-list** at `0x80100200..0x801002FF` (forward-pointer; M0.5 capstone-extension scope), CPU stages a list of "draw rectangle / draw bitmap / set palette" descriptors; HDMI peripheral processes them in vertical-blanking. CSA-201 driver-track lab.
- **VCP peripheral integration** (forward-pointer; M0.5+). Main CPU enqueues microcode jobs in a descriptor ring; VCP processes; results land in a result-ring back to main CPU. The hybrid C+B architecture from §23.

Tradeoffs.

- **Pro:** decouples CPU and IP timing entirely. CPU can stage many work units in advance; IP processes at its own pace. Streaming workloads (audio at 22 kHz; HDMI at 60 Hz frame rate) are exactly this shape.
- **Pro:** introduces real hardware-software cooperation timing. Students see the canonical pattern that production embedded systems use (TI Sitara DMA, ESP32 SoC peripheral drivers, ARM AMBA-AHB DMAC's all use descriptor rings). This is curriculum-payload pedagogy.
- **Pro:** flow-control falls out naturally. The "head wraps to tail" full-condition makes back-pressure visible; the CPU can't enqueue when the IP is behind.
- **Con:** most complex pattern in the kit. Two pointers, modular arithmetic, descriptor-format design choice, full/empty edge cases. Lab 12.5 is the chapter's most demanding lab partly for this reason.

- **Con:** descriptor format is itself a design decision. CSA-101 fixes it at 16 bytes (parameter address + length + flags + reserved); CSA-201 expands to chained descriptors with linked-list semantics.

When to reach for it. When CPU and IP need to cooperate on a streaming workload, when work units are heterogeneous (need parameter passing not just bit-set), or when you want the CPU to be free to do other work between batches. Audio capstone (Lab 12.5) is the canonical CSA-101 use site.

Cross-chapter placement table

Chapter	Section	Pattern introduced	Lab anchor
Ch 5	§5.7 (Memory-Mapped I/O)	Scratchpad memory (HDMI tile-map; GPIO regfile)	Lab 5.4 (Virtus Console bring-up)
Ch 5	§5.10 (sim-then-silicon)	Doorbell register (HDMI vsync polling in sim)	(sidebar; no lab)
Ch 11	§11.10 (end-to-end demo)	Scratchpad memory (Console.printChar to tile-map)	Lab 11.3 (HDMI end-to-end)
Ch 12	§12.6 (driver-layer OS services)	All three patterns (driver implementations)	Lab 12.4 (Console drawing capstone)
Ch 12	§12.8 (audio integration)	Descriptor-ring (audio playback queue)	Lab 12.5 (audio-capstone silicon-cert)
Ch 12	§12.12 (heterogeneous-multi-processor sidebar)	Doorbell + descriptor-ring (VCP forward-pointer)	(sidebar; no lab)
CSA-201	§14 (driver-writing track)	Descriptor-ring with chained descriptors (DMAC patterns)	Driver-track capstone

---

Forward-compatibility notes

- **CSA-201 interrupts.** Doorbell-pattern status bits become interrupt sources when CSA-201 introduces the `ecall` trap mechanism + interrupt controller. CSA-101 polling code ports unchanged; the `Sys.wait` busy-loop is replaced by an `wfi` instruction that wakes on interrupt.

- **CSA-201 DMA.** Descriptor-ring pattern extends to chained descriptors (each descriptor carries a `next` pointer); the IP reads the chain autonomously without per-descriptor CPU intervention.
- **CSA-201 cache coherence.** When CSA-201 introduces L1/L2 caches, scratchpad memory gains a `coherent` vs `non-coherent` distinction, IPs reading from cached regions need explicit cache flushes (`fence` instruction). CSA-101 has no caches; this concern does not arise.
- **VCP integration (M0.5+).** The VCP coprocessor uses all three patterns: main CPU loads VCP microcode via scratchpad (`local_ram` at `0x80010000`); main CPU rings VCP doorbell to start; VCP returns results via descriptor-ring back to main CPU. The hybrid C+B architecture (per Findings §23 + R1T9.5 silicon-validation) is the canonical CSA-101 example of all three patterns composed.

Where to read more

- `handouts/cross-chapter-axi-manifold-base-addresses.md`. Canonical AXI4-Lite manifold address map; 3-pattern overview.
- `handouts/cross-chapter-stdlib-service-reference.md`, OS-side stdlib contract (each module's MMIO offsets per IP slave).
- `handouts/cross-chapter-instr-mem-layout.md`. Instr_mem + data_mem layout; complements the MMIO map.
- Ch 5 §5.7, Memory-Mapped I/O chapter prose.
- Ch 12 §12.6 + §12.8. Driver-layer + audio-driver chapter prose.
