Generated Specification Document

INTRODUCTION

-- Purpose --

Provide a Wishbone-attached watchdog/COP peripheral that software can configure to enforce system liveness: program a timeout and modes, enable optional early-warning interrupts, and safely service the watchdog via a two-key sequence. On missed service it asserts a reset and latches an event for diagnostics. The design includes lockable configuration to prevent unintended changes, pause behavior in debug/low-power modes, safe operation across bus and counter clock domains with robust clock-domain crossings, and software-readable status and counter snapshots for observability.

-- Key Features --

- Wishbone slave watchdog peripheral with selectable 8- or 16-bit data width and proper byte/halfword enables.
- Configurable bus timing: single-cycle combinational ACK/data or one-wait-state registered responses.
- Compact register space: 48-bit readback, 40-bit writable; top byte reserved/read-only.
- Width-aware access mapping and write adaptation, including 8-bit bus duplication and byte-specific timeout writes.
- Programmable down-counter (parameterized width) clocked by startup oscillator or bus clock in scan/test.
- Robust control: main enable (cop_ena), two-stage configuration lock (clck → cwp → cop_ena), and timeout write-protect while enabled.
- Low-power/debug: counter pause support in debug/wait/stop; expiry holds reset if reached while paused.
- Early-warning interrupt with selectable thresholds (≤16/≤32/≤64), two-stage IRQ pipeline, and a separate sticky event latch for expiry diagnostics.
- Safe service (reload) via two-key sequence (SERV_WD_0 then SERV_WD_1) producing a one-cycle reload pulse that clears cop_rst_o and reloads timeout.
- Clean clock-domain crossing: level-based reload handshake (bus→counter) and bus-domain snapshot of the live counter on each cop clock edge.
- Flexible reset: asynchronous reset with polarity control (ARST_LVL), synchronous bus reset, and power-on reset domain for expiry/event paths; parameterized initial enable (INIT_ENA).
- Explicit event clear pulse; event auto-clears on reload.
- Integration-ready partitioning: cop_wb_bus (bus interface), cop_regs (configuration/validation), cop_count (counter/outputs).
- Top-level outputs: cop_rst_o (watchdog reset) and cop_irq_o (early interrupt).

-- Block Composition --

cop_top is composed of three tightly-coupled blocks:
- cop_wb_bus: Wishbone slave front-end that performs bus handshake (SINGLE_CYCLE combinational or 1-wait-state registered), address decoding, byte/halfword select handling via wb_sel_i, and read-data multiplexing from a 48-bit internal readback bus (read_regs). It produces a

5-bit write strobe vector (write_regs[4:0]) aligned to the decoded write regions and gates writes with wb_wacc.
- cop_regs: Control/status register file that implements configuration and safety/lock rules. It holds cop_ena, debug_ena, stop_ena, wait_ena, cop_irq_en[1:0], cwp, clck (write-once), and clear_event pulse; programs timeout_value[COUNT_SIZE-1:0] with byte/halfword granularity (writes only when cop_ena==0; DWIDTH==8 duplicates control bytes as needed); and generates a one-cycle reload_count pulse from a two-key service write sequence. It assembles bus-domain readback fields together with cop_count status into read_regs[47:0] (topmost byte read-only) and enforces that mode changes occur only when the COP is disabled or being disabled by the same write; clck freezes cwp, and cwp blocks changes to cop_ena.
- cop_count: Watchdog down-counter running in the counter clock domain (cop_clk), selectable from startup_osc_i or wb_clk_i (scan/test). It supports pause modes driven by debug_ena/wait_ena/stop_ena, reload via a two-flop CDC handshake (reload_1 asserted in bus domain by reload_count or cop_ena deassert; reload_2 in cop_clk domain performs load of timeout_value and clears cop_rst_o), and bus-stable counter capture for software readback. Outputs include cop_rst_o on expiry (held until reload, even if paused), cop_irq_o early-warning interrupt with programmable thresholds (cop_irq_en=00 disable; 01/10/11 map to <=16/32/64), and an event latch (cop_event) that records expiry until cleared by reload_count or clear_event; POR clears expiry-related state.
Composition and interconnects:
- Data/read path: cop_regs and cop_count populate read_regs[47:0]; cop_wb_bus slices it to wb_dat_o per DWIDTH and wb_adr_i.
- Write path: cop_wb_bus decodes writes to produce write_regs[4:0]; cop_regs consumes these strobes to update control/status and timeout fields or trigger pulses.
- Clocks/resets: Bus domain (wb_clk_i) hosts cop_wb_bus and cop_regs and bus-side IRQ/event/capture logic; counter domain (cop_clk) hosts cop_count. Resets include asynchronous reset (ARST_LVL polarity) for bus/counter init, synchronous bus reset (wb_rst_i) for bus-side state, and POR (por_reset_i) for expiry/event clearing.
- Parameters: DWIDTH (8/16), SINGLE_CYCLE, ARST_LVL, COUNT_SIZE, INIT_ENA (default cop_ena), SERV_WD_0/SERV_WD_1 (service keys) configure interfaces and behavior.

-- Typical Use Cases --

- Basic system liveness supervision: firmware periodically services the watchdog via the two-key sequence; early-warning IRQ signals imminent expiry; final expiry asserts reset and latches an event for fault recovery.
- Boot-time protection: enable INIT_ENA to start the watchdog on POR using the startup oscillator so the system is protected before clocks/peripherals are configured; firmware services promptly during initialization.
- Low-power and debug modes: pause counting in wait/stop/debug to avoid unintended resets while the CPU sleeps or is halted; auto-resume on exit; if the counter reached zero while paused, reset remains asserted for clear fault indication.
- Safety-critical and production lockdown: use write-once cwp/clck locks to prevent post-bring-up changes to enable and configuration, supporting functional safety and anti-tamper requirements.
- Runtime reconfiguration with safeguards: temporarily disable COP, update timeout/mode bits using byte/halfword writes (8-/16-bit Wishbone), then re-enable; two-key servicing prevents spurious reloads.
- Post-mortem diagnostics: after a reset, software reads the sticky event flag and captured counter snapshot to confirm a watchdog cause and how close to expiry service occurred; clear the event after logging.
- Early-warning scheduling: select interrupt thresholds (<=16/32/64) to schedule timely servicing or initiate graceful recovery (state save, telemetry, fallback mode) before hard reset.
- DFT and verification: operate the counter from the bus clock in scantestmode; choose SINGLE_CYCLE or registered Wishbone responses to meet timing; validate reload handshake and

readback paths.
- Flexible bus integration: deploy in 8-bit or 16-bit Wishbone systems with byte/halfword strobes; support partial updates to timeout; use registered ACK for deterministic wait states under stalls.
- Always-on monitoring: rely on the startup oscillator so the watchdog continues counting when the bus is inactive; maintain asserted reset if expiry occurs while paused, ensuring persistent fault signaling.
- Robust cross-domain deployments: use built-in CDC handshakes so reload requests and status cross between bus and watchdog clocks reliably, even with bus stalls or independent clock domains.

-- Safety and Reliability Mechanisms --

Safety and reliability mechanisms integrated across COP modules:
- Configuration protection and write locks: Two-stage lock chain (clck, cwp) prevents accidental enable changes; clck is write-once sticky high; cwp blocks writes to cop_ena when set. Mode bits (debug_ena, stop_ena, wait_ena) update only when COP is disabled or when the same write disables COP. Timeout_value writes are ignored while COP is enabled; byte writes are supported on 8-bit buses but remain gated by cop_ena=0. Servicing requires a validated two-key sequence (SERV_WD_0 then SERV_WD_1 at 5'b10000) to generate a one-cycle reload_count pulse; spurious writes do not reload.
- Event integrity and diagnostics: cop_event is a sticky latch recording watchdog expiry (cop_rst_o) and persists until cleared by a reload or an intentional clear_event pulse generated via specific write addresses. Software reads use cop_capture snapshots to avoid live counter CDC hazards.
- Reset and initialization robustness: Supports asynchronous reset with configurable polarity (ARST_LVL) and synchronous bus reset. Power-on reset forces initial counter load and clears cop_rst_o. Disabling cop_ena or successful service asserts a bus-to-counter reload request that clears cop_rst_o and reloads the counter, ensuring defined recovery. Conservative defaults on reset (timeout_value=all 1s, cop_irq_en=00, mode bits=0); INIT_ENA controls default enable.
- Clock-domain crossing safety: Level-sticky reload handshake (reload_1 in bus_clk, reload_2 in cop_clk) guarantees reliable delivery and self-clear after observation; POR forces reload_2=1 for initial load. Double-flop resampling of cop_clk provides a clean bus-domain capture trigger. The interrupt is pipelined in the bus domain for timing reliability.
- Watchdog operation safeguards: Early warning interrupt thresholds selectable (01→<=16, 10→<=32, 11→<=64); disabled when cop_irq_en=00. Counter pauses when debug/wait/stop modes are enabled, preventing unintended resets during those system states. Reset output asserts on expiry and remains asserted at zero if paused, providing fail-safe behavior.
- Bus interface safety mechanisms: Writes are accepted only on ACK (wb_wacc) and are qualified by byte enables; optional registered ACK mode (SINGLE_CYCLE=0) provides deterministic one-wait-state handshakes; deassert STB between transfers for clean protocol. Read/write mapping restricts the writable region (40/48 bits), with the topmost byte [47:40] read-only to reduce risk of unintended state changes. 8-bit bus adaptation duplicates payload internally to 16 bits while preserving gating semantics; address decoding separates writable control/timeout fields from read-only status.
- Test/scantest accommodation: Counter can run from the bus clock in scan/test mode to support DFT while retaining reload and CDC protections.

-- Design Assumptions and Limitations --

- Bus/protocol assumptions
- Wishbone classic slave; only DWIDTH=8 or 16 are supported.
- Internal read space is 48 bits; writes cover only 5 bytes (40 bits). The topmost byte [47:40] is reserved/read-only.
- SINGLE_CYCLE=1: reads/writes complete in a single cycle when CYC/STB are asserted. SINGLE_CYCLE=0: read data is registered and ACK pulses every other cycle if the master holds CYC/STB; writes take effect on ACK. Masters should deassert STB between transfers or tolerate wait states.
- Addressing requires proper alignment and wb_sel_i usage; the map assumes straightforward

byte/halfword slicing without endianness conversion.

- Clocking and reset assumptions
- Two clock domains: bus_clk (register/bus domain) and cop_clk (counter domain). cop_clk normally comes from startup_osc_i; when scantestmode=1, cop_clk is sourced from bus_clk for DFT.
- Two independent asynchronous resets: async_rst_b (polarity selectable via ARST_LVL) for bus-side logic, and a separate active-low POR reset for the counter/event path and initial reload (cop_rst_o side). wb_rst_i synchronously resets bus-domain state. POR drives the initial counter load via the reload handshake.

- Configuration and write-protection limitations
- Timeout value writes are ignored while cop_ena=1; software must disable the COP to change the timeout.
- clck is a sticky write-one lock that permanently freezes cwp; when cwp=1, cop_ena cannot be changed.
- debug_ena/wait_ena/stop_ena can change only when COP is disabled, or when the write transaction explicitly disables COP in the same accepted write.
- Servicing (reload) requires two consecutive accepted writes to the same address with the exact key sequence SERV_WD_0 then SERV_WD_1; any deviation (wrong value, different address, or a gap without ACK acceptance) cancels the reload.

- Width and data-path constraints
- The timeout/control path is built around a 16-bit payload. COUNT_SIZE is parameterized, but only the lower 16 bits are software-programmable via the defined register map; higher bits (if any) are not writable via this interface.
- For DWIDTH==8, the 8-bit bus write data is duplicated internally to 16 bits to preserve semantics; software should use byte-specific addresses for partial timeout writes. Due to duplication, certain control bits (e.g., clear_event) may appear at mirrored positions and behave identically.

- Interrupt, reset, and event behavior
- Early interrupt threshold is limited to three preset levels: counter <=16, <=32, or <=64 counts; arbitrary thresholds are not supported.
- cop_event is a latched indicator that sets on reset assertion and clears only on a successful service reload or an explicit clear_event write. Disabling COP does not clear the event latch.
- cop_rst_o asserts when the counter reaches zero and deasserts on the reload handshake completion. If the counter hits zero while counting is paused (due to debug/wait/stop with corresponding enables), cop_rst_o remains asserted until the counter is serviced or the pause is released.

- CDC and observability assumptions
- Reloads cross from bus_clk to cop_clk via a level-sticky handshake (reload_1/reload_2); proper CDC constraints are required to avoid missed or duplicate reloads.
- Early IRQ logic compares a live view of the counter in the bus domain, introducing CDC sensitivity. Software should use the captured snapshot (cop_capture) for stable readback, and system timing closure should account for the comparator’s CDC path.

- Addressability and mapping
- Read mapping exposes six bytes (48 bits). Write mapping covers five bytes; the highest byte is read-only.
- 8-bit map: reads at addresses 000..101; writes at 000..100. 16-bit map: reads at 000..010; writes at 000, 001, 010 with five effective write strobes. Masters must follow the documented address/sel conventions.

- Defaults and side effects

- On reset, timeout_value defaults to all 1s; cop_ena defaults per INIT_ENA. Writes to timeout_value are gated by cop_ena==0. In SINGLE_CYCLE=0, only ACKed writes take effect.
- Throughput under SINGLE_CYCLE=0 is limited to one transfer per two cycles if CYC/STB remain asserted; system masters should plan for this handshake behavior.

IO PORTS

-- Wishbone Interface Signals --

Wishbone B4 classic slave interface on wb_clk_i with parameterizable data width and single-/two-cycle timing.

Parameters
- DWIDTH: 8 or 16. Sets wb_dat_i/o width and wb_sel_i width (DWIDTH/8).
- SINGLE_CYCLE: 1 = zero wait state (ACK same cycle, read data combinational); 0 = one wait state (ACK next cycle, read data registered).

Clock and Reset
- wb_clk_i (in): Wishbone clock.
- wb_rst_i (in): synchronous reset for bus-side handshake and data path.

Master-to-Slave Signals
- wb_cyc_i (in): cycle valid; qualifies a bus cycle.
- wb_stb_i (in): strobe/select for this slave instance.
- wb_we_i (in): write enable; 1=write, 0=read.
- wb_adr_i (in): address; low-order bits select an 8-bit or 16-bit slice of a 48-bit register space (exact width is system-dependent; only low bits used here).
- wb_sel_i [DWIDTH/8-1:0] (in): byte enables; 1 bit when DWIDTH=8, 2 bits when DWIDTH=16. Gated into write updates.
- wb_dat_i [DWIDTH-1:0] (in): write data from master.

Slave-to-Master Signals
- wb_dat_o [DWIDTH-1:0] (out): read data to master. Combinational when SINGLE_CYCLE=1; registered when SINGLE_CYCLE=0.
- wb_ack_o (out): acknowledge; sole handshake response (no ERR/RTY/STALL).

Handshake and Timing
- Active transfer when module_sel = (wb_cyc_i && wb_stb_i).
- SINGLE_CYCLE=1: wb_ack_o asserts in the same cycle as module_sel; wb_dat_o driven combinationally from selected read slice.
- SINGLE_CYCLE=0: one wait state. For reads, data is captured on the first cycle (module_sel && !wb_we_i) and wb_ack_o asserts the next cycle. If CYC/STB remain asserted, ACK pulses every other cycle.

Read Data Mapping (from internal 48-bit read_regs)
- DWIDTH=8: wb_adr_i 0..5 map to bytes [7:0], [15:8], [23:16], [31:24], [39:32], [47:40].
- DWIDTH=16: wb_adr_i 0..2 map to halfwords [15:0], [31:16], [47:32].

Write Acceptance and Byte Enables
- A write is accepted when wb_wacc = module_sel && wb_we_i && (SINGLE_CYCLE || wb_ack_o).
- wb_sel_i qualifies which byte(s)/halfword(s) update on write; only 40 of 48 bits are writable (topmost byte is read-only).
- Write strobe decode into a 5-bit write_regs vector:
- DWIDTH=8: adr 0..4 -> 00001, 00010, 00100, 01000, 10000 respectively; adr 5 is read-only.
- DWIDTH=16: adr 0 -> 00011 (lower two bytes), adr 1 -> 01100 (middle two bytes), adr 2 -> 10000 (top single byte).

Protocol Notes
- No ERR/RTY/STALL signals are implemented; masters should deassert STB between transfers in SINGLE_CYCLE=0 mode or expect ACK every other cycle.
- An additional asynchronous reset may exist internally (arst_i with ARST_LVL) but is not part of the Wishbone port.

-- Data Width Configuration --

- DWIDTH parameter selects Wishbone data width: 8 or 16 bits (set in cop_wb_bus and mirrored in cop_regs). In DWIDTH=8, the 8-bit write_bus is duplicated internally to 16 bits to keep control-field positions consistent; in DWIDTH=16, write_data is passed through unchanged.
- Read mapping from internal 48-bit read_regs to wb_dat_o:
- DWIDTH=8: wb_adr_i 000→[7:0], 001→[15:8], 002→[23:16], 003→[31:24], 004→[39:32], 005→[47:40].
- DWIDTH=16: wb_adr_i 000→[15:0], 001→[31:16], 002→[47:32].
- Write decode and granularity (wb_sel_i honored in all cases):
- DWIDTH=8: writes at 000 (control), 001 (event clear), 010 (timeout low byte), 011 (timeout high byte), 100 (service). Address 101 (top byte [47:40]) is read-only. Multi-byte fields require multiple byte writes.
- DWIDTH=16: writes at 000 (control + clear), 001 (timeout, two bytes at once), 002 (service). Halfword accesses update two bytes at a time.
- Writable region excludes the topmost byte [47:40]; effective writable width is 40 of 48 bits in all modes.
- Address map consistency: decode uses {eight_bit_bus, wb_adr_i} so software-visible locations remain aligned across DWIDTH settings.
- Behavior unaffected: DWIDTH changes only bus presentation and write-strobe width; internal counter width (COUNT_SIZE) and cop_count operation are unchanged. Service keys and control compares use the adapted write_data so semantics are consistent across widths.

-- Clock Inputs --

Two external clock inputs define the timing domains. wb_clk_i is the Wishbone bus clock and the primary bus-domain clock; it drives the bus front-end (cop_wb_bus), the register block (cop_regs), and the bus-clock portions of cop_count (IRQ pipeline, counter snapshot/capture logic, and the bus-side of the reload handshake). startup_osc_i is the watchdog/counter clock and the primary counter-domain clock; it drives the down-counter in cop_count during normal operation and is asynchronous to wb_clk_i. The counter domain clock (cop_clk) equals startup_osc_i by default; when scantestmode=1 (DFT/test), cop_clk is taken from wb_clk_i so the design runs in a single clock domain. No internal clock division or gating is used; pause/stop behavior is implemented via enable logic inside cop_count. In normal mode treat wb_clk_i and startup_osc_i as fully asynchronous and constrain them independently; verify CDC paths (level-sticky reload handshake from bus→counter and cop_clk edge detection/counter snapshot from counter→bus). In scan/test mode both domains are synchronous to wb_clk_i; CDC structures remain but operate synchronously. Resets are not clock inputs; bus-domain synchronous resets align to wb_clk_i and POR/async resets are separate.

-- Reset Inputs --

- Reset signals and domains
- arst_i: External asynchronous reset; polarity selectable via parameter ARST_LVL. Internally used as async_rst_b = arst_i ^ ARST_LVL (active-low) and affects both bus and watchdog logic where implemented as async clear.
- wb_rst_i: Bus-domain synchronous reset (synchronous to wb_clk_i). Affects only bus-domain flops/state.
- por_reset_i: Power-on reset, asynchronous active-low in the watchdog (cop) clock domain. Guarantees initial counter load and event/rst clearing.

- Effects by module/domain
- cop_wb_bus (bus interface):
- wb_rst_i: Synchronously clears bus_wait_state (affects ACK timing).
- arst_i (async_rst_b low): Asynchronously clears bus_wait_state.
- cop_regs (register file, bus domain):
- wb_rst_i or arst_i: Restores register defaults identically:
- timeout_value = all 1s
- cop_irq_en = 2'b00
- cop_ena = INIT_ENA
- debug/stop/wait = 0
- cwp = 0, clck = 0 (clck becomes sticky once set after reset)
- reload_count = 0, service_cop = 0
- cop_count (watchdog counter, spans domains):
- wb_rst_i (bus side): Clears bus-domain flops (reload_1, capture pipeline, IRQ pipeline). Does not request or perform a counter reload.
- arst_i (async_rst_b low): Asynchronously sets the counter to all 1s and clears resync/capture/reload_1/IRQ pipeline. Does not clear cop_rst_o.
- por_reset_i (active-low, cop clock): Clears cop_rst_o and cop_event; initializes reload_2=1 so that on POR release the counter loads timeout_value.

- Behavioral/reset interactions
- Disabling cop_ena or a valid service sequence asserts reload_1 (bus domain), which handshakes to reload_2 (cop clock) to reload the counter.
- cop_rst_o is cleared by POR and by a counter reload; it is not cleared by arst_i or wb_rst_i alone.
- cop_event (sticky event) is cleared by POR or by an event_reset action (reload_count or clear_event). It is not cleared by wb_rst_i or arst_i alone; software must clear it if needed.
- POR is the only reset that guarantees both an initial counter load and clearing of cop_rst_o and the event latch.

- Clock/reset coherency notes
- Two clock domains: bus_clk (wb_clk_i) and cop_clk (startup_osc_i; may be wb_clk_i in scan/test). Treat async_rst_b as active-low everywhere it is used.
- sync_reset referenced in submodules corresponds to wb_rst_i and only affects bus-domain logic.
- Neither wb_rst_i nor arst_i inherently triggers a counter reload; only POR or explicit reload/service does.

-- Mode/Power-State Inputs --

External mode/power-state inputs to cop_top can pause the watchdog counter when corresponding enable bits are set in cop_regs. Inputs: debug_mode_i, wait_mode_i, stop_mode_i (all level-sensitive,

active-high, effective immediately in the cop_clk domain), and scantestmode (active-high). Enable bits: debug_ena, wait_ena, stop_ena in cop_regs. Pause gating: stop_counter = (debug_mode_i & debug_ena) | (wait_mode_i & wait_ena) | (stop_mode_i & stop_ena). While stop_counter=1, the counter holds its current value and decrementing is halted. Pausing does not mask early interrupt generation; interrupts remain based on the live counter value. If the counter reaches 0 while paused, cop_rst_o stays asserted until a reload occurs (service sequence or disabling cop_ena). scantestmode selects wb_clk_i as the watchdog counter clock instead of the startup oscillator; it is orthogonal to pause modes and does not itself pause counting.
Configuration constraints: debug_ena, wait_ena, stop_ena may be changed only when COP is disabled or when the write disables COP, preventing unsafe enable+relax in one step.
Clock-domain considerations: mode inputs are external to the bus domain and no explicit synchronizers are provided; they must be stable/synchronous to cop_clk to avoid metastability.

-- Test/Scan Inputs --

- scantestmode (DFT/test-mode): Assert to collapse all logic onto wb_clk_i; forces cop_count to use wb_clk_i and ignores startup_osc_i; simplifies reload handshake to a single clock domain.
- wb_clk_i (bus clock): Sole active clock in scan when scantestmode=1; drives bus-side logic and watchdog counter; ensure stable, deterministic frequency in test.
- startup_osc_i (watchdog oscillator): Normal functional clock; must be ignored during scan by asserting scantestmode=1 to avoid asynchronous domains.
- por_reset_i (active-low POR): Hold low during scan to keep outputs deterministic; releasing POR reloads the counter from timeout_value; use for power-on sequencing, not routine bus-domain init.
- wb_rst_i (synchronous bus reset): Preferred reset in scan/ATPG for deterministic initialization of bus/register domain; avoid mixing with asynchronous resets mid-scan.
- arst_i (async reset, polarity per ARST_LVL): External asynchronous clear for bus-side FSMs/registers; use only when an async clear is required; honor ARST_LVL polarity.
- debug_mode_i, wait_mode_i, stop_mode_i (mode holds): Can pause counting when corresponding enables are set in cop_regs; keep deasserted for normal scan unless intentionally freezing activity; configure enables first or these inputs have no effect.
- SINGLE_CYCLE (test parameter): Set to 0 in scan to register ACK/data and reduce combinational depth; set to 1 only for single-cycle functional testing.
- Scan operation guidance: Keep cop_ena=0 for quiet scan or pause via mode inputs; avoid issuing the service key sequence unless testing reload behavior; constrain writes to avoid the read-only top byte in the 48-bit read space.

-- Outputs --

- wb_dat_o (bus_clk): Wishbone read-data output, width DWIDTH (8 or 16). Drives a byte/halfword slice of the internal 48-bit readback space selected by address and DWIDTH. SINGLE_CYCLE=0: data is registered and valid on the ACK cycle. SINGLE_CYCLE=1: data is combinational. The topmost byte in the readback map is read-only (affects content only).
- wb_ack_o (bus_clk): Wishbone acknowledge. SINGLE_CYCLE=1: combinational, asserted in the same cycle as a valid CYC/STB access to this slave (module select true). SINGLE_CYCLE=0: one-wait-state response; ACK pulses every other cycle when CYC/STB are held. Qualifies write acceptance and indicates when wb_dat_o is valid for reads.
- cop_irq_o (bus_clk): Early-warning interrupt. Asserted when the live counter is <= threshold selected by cop_irq_en (00: disabled; 01: <=16; 10: <=32; 11: <=64); deasserted otherwise. Derived from the live counter (not the captured bus snapshot) and passed through a two-stage pipeline in bus_clk for timing stability.
- cop_rst_o (cop_clk): Watchdog reset. Asserted when the counter reaches zero; cleared on POR or any reload. If the counter hits zero while paused (debug/wait/stop), cop_rst_o remains asserted until a reload or POR. Generated in the cop_clk domain.

Notes: wb_dat_o and wb_ack_o, as well as cop_irq_o, are in the bus_clk domain; cop_rst_o is in the cop_clk domain.

-- Signal Summary --

- Clocks and resets
- wb_clk_i (input): Wishbone bus clock; drives cop_wb_bus, cop_regs, and bus-domain flops in cop_count.
- wb_rst_i (input, active-high): Synchronous bus-domain reset.
- arst_i (input, async): Asynchronous bus-side reset; active level is ARST_LVL.
- startup_osc_i (input): External oscillator driving the watchdog counter clock.
- por_reset_i (input, active-low): Power-on reset for cop_count reset/event path.
- scantestmode (input): When 1, selects wb_clk_i as cop_count clock for scan/DFT.

- Mode inputs
- debug_mode_i, wait_mode_i, stop_mode_i (inputs): Pause the counter only when their corresponding enable bits (debug_ena/wait_ena/stop_ena) are set.

- Wishbone interface
- wb_cyc_i, wb_stb_i (inputs): Cycle and strobe qualifiers.
- wb_we_i (input): Write enable.
- wb_adr_i (input): Address; low bits select slices of the 48-bit read space (6 byte addresses for DWIDTH=8; 3 halfword addresses for DWIDTH=16).
- wb_sel_i (input [(DWIDTH/8)-1:0]): Byte/halfword enables.
- wb_dat_i (input [DWIDTH-1:0]): Write data.
- wb_dat_o (output [DWIDTH-1:0]): Read data sourced from internal 48-bit readback bundle.
- wb_ack_o (output): Acknowledge; SINGLE_CYCLE=1 combinational response, SINGLE_CYCLE=0 one wait-state with registered data.

- System outputs
- cop_rst_o (output): Watchdog reset asserted when the counter reaches zero; cleared on reload or POR; remains asserted if zero occurs while paused until reload/POR.
- cop_irq_o (output): Early-warning interrupt; thresholds via cop_irq_en: 00 disabled, 01 -> count <=16, 10 -> <=32, 11 -> <=64.

- Internal interconnect
- read_regs (wire [47:0]): Aggregated control/status and snapshot readback from cop_regs/cop_count to cop_wb_bus (topmost byte read-only; lower 40 bits writable via write_regs).
- write_regs (wire [4:0]): Write strobes from cop_wb_bus into cop_regs.
- timeout_value (wire [COUNT_SIZE-1:0]): Programmable reload value from cop_regs to cop_count.
- cop_ena, cwp, clck (wires, 1-bit): Enable and lock bits from cop_regs to cop_count.
- debug_ena, stop_ena, wait_ena (wires, 1-bit): Mode enables from cop_regs to cop_count.
- cop_irq_en (wire [1:0]): Interrupt threshold selection from cop_regs to cop_count.
- reload_count (wire, bus-clock pulse): Service/reload request from cop_regs to cop_count; crosses clock domains via reload handshake.
- clear_event (wire, bus-clock pulse): Event clear request from cop_regs to cop_count.
- cop_capture (wire [COUNT_SIZE-1:0]): Bus-domain snapshot of the counter from cop_count for software readback.

- Notes impacting signals
- DWIDTH parameter (8 or 16) sets widths of wb_dat_i/o and wb_sel_i and affects slice decoding; in DWIDTH=8, some writes duplicate 8-bit data into 16 bits; timeout_value supports byte-specific writes.

- CDC paths: reload_count uses a two-flop handshake (reload_1 bus, reload_2 cop clock); cop_capture provides a stable bus-clock snapshot.
- Reset polarities: por_reset_i active-low; arst_i active level via ARST_LVL; wb_rst_i active-high synchronous.

ARCHITECTURE

-- Top-Level Data and Control Flow --

- Architectural flow
- cop_top integrates a Wishbone slave front-end (cop_wb_bus), a register/configuration block (cop_regs), and the watchdog counter engine (cop_count) across two clock domains: wb_clk_i (bus/control, IRQ/event, capture) and cop_clk (counter/reset generation).

- Wishbone ingress and write acceptance (bus domain)
- Transactions enter via wb_cyc_i/wb_stb_i with wb_we_i, wb_adr_i, wb_dat_i, and produce wb_ack_o and wb_dat_o.
- SINGLE_CYCLE=1: combinational read data and ACK (no wait state). SINGLE_CYCLE=0: one-cycle latency; under continuous CYC/STB, ACK pulses every other cycle; writes qualify on wb_wacc in the ACK cycle.
- Write decode produces a 5-bit write_regs strobe vector (only 5 of 6 bytes writable); the highest read_regs byte is read-only. For DWIDTH=16, adr[2:0]=000/001/010 map to strobe bits 00011/01100/10000. For DWIDTH=8, adr[2:0]=000..100 map to 00001/00010/00100/01000/10000. On DWIDTH=8, wb_dat_i is byte-replicated to 16 bits; timeout low/high have byte-specific strobes.
- Read mux slices a 48-bit read_regs bus into wb_dat_o: DWIDTH=8 uses adr[2:0]=000..101 → [7:0]..[47:40]; DWIDTH=16 uses adr[2:0]=000..010 → [15:0],[31:16],[47:32].

- Register/config control (bus domain)
- cop_regs latches configuration on write_regs: cop_ena, mode masks (debug_ena, stop_ena, wait_ena), cop_irq_en, timeout_value, and lock bits (cwp, clck).
- Protection rules: timeout_value writes ignored while cop_ena=1; mode bits change only when COP is disabled or in the same write that disables it; cop_ena writable only if cwp=0; cwp writable only if clck=0; clck is sticky once set.
- Service sequence: two consecutive writes of SERV_WD_0 then SERV_WD_1 to the service address generate a one-cycle reload_count pulse; any other write aborts the sequence.
- clear_event is a write-1 pulse at specific decoded addresses; event can also be cleared by reload.

- Reload/enable handshake and CDC
- In the bus domain, reload_count and cop_ena drive a level-sticky reload_1 request. Disabling cop_ena also requests reload. reload_1 remains asserted until observed in cop_clk.
- In the counter domain, reload_1 is synchronized as reload_2 to load timeout_value and clear cop_rst_o/event. Observation of reload_2 auto-clears reload_1 back in the bus domain.

- Counter and reset generation (counter domain)
- cop_count is a down-counter clocked by cop_clk; on reload_2 it loads timeout_value, otherwise decrements when not paused.
- Pausing is driven by stop_counter gating derived from enabled modes

(debug_ena/stop_ena/wait_ena) and their corresponding system states; if paused at 0, cop_rst_o stays asserted until a reload.
- cop_rst_o asserts when the counter reaches zero and is deasserted by reload_2.

- Status, capture, and IRQ (bus domain)
- Counter capture: an edge-detect/synchronizer snapshots the live cop_counter into cop_capture for coherent software reads; read_regs sources cop_capture, not the asynchronous live counter.
- IRQ: a bus-domain comparator asserts cop_irq_o when the live counter is below an early-warning threshold selected by cop_irq_en (<=16/32/64); disabled when cop_irq_en==0.
- Event latch (cop_event): sets when cop_rst_o asserts; cleared by reload_count/reload_2 or clear_event writes; POR clears it.
- read_regs composition includes configuration, status (locks, enable, mode), captured count, IRQ/event state, and any ancillary flags; the topmost byte is read-only.

- Reset behavior
- Bus side (cop_wb_bus FSM, cop_regs): asynchronous arst_i (polarity via ARST_LVL) and synchronous wb_rst_i; defaults include timeout_value=all 1s and cop_ena=INIT_ENA.
- Counter side: por_reset_i forces an initial reload (reload_2=1) and clears cop_rst_o/event; async_rst_b resets the counter and clears capture/IRQ pipelines.

- Test/DFT
- In scantestmode, cop_clk is sourced from wb_clk_i for test; CDC handshakes and synchronizers remain active.

- Parameters influencing flow
- COUNT_SIZE (counter width), INIT_ENA (default enable), ARST_LVL (async reset polarity), SERV_WD_0/SERV_WD_1 (service keys), DWIDTH (8/16-bit bus), SINGLE_CYCLE (ACK/data timing).

-- Wishbone Front-End --

Wishbone Front-End (cop_wb_bus) is a Wishbone Classic slave interface for the cop_top watchdog block. It bridges the bus to internal registers via a 48-bit read_regs input and a 5-bit write_regs strobe output, supporting DWIDTH=8 or 16 and two handshake modes via SINGLE_CYCLE. Parameters: DWIDTH ∈ {8,16}, SINGLE_CYCLE ∈ {0,1}, ARST_LVL for asynchronous reset polarity. Interface (typical Wishbone signals): wb_clk_i, wb_rst_i, arst_i; wb_cyc_i, wb_stb_i, wb_we_i, wb_adr_i, wb_dat_i, wb_sel_i; wb_dat_o, wb_ack_o; plus read_regs[47:0] (from back-end) and write_regs[4:0] (to back-end). Handshake: module_sel = wb_cyc_i && wb_stb_i. SINGLE_CYCLE=1 provides combinational ACK (wb_ack_o=module_sel) and rd_data_mux on the same cycle; writes are accepted immediately. SINGLE_CYCLE=0 inserts one wait state: ACK asserts on the second cycle while STB is held; read data is captured on the first read cycle and presented on the ACK cycle; writes are qualified only on the ACK cycle. Guidance: deassert STB between transfers to receive single ACK pulses in two■cycle mode. Read mapping from read_regs by DWIDTH and address: 8-bit mode adr 000→[7:0], 001→[15:8], 002→[23:16], 003→[31:24], 004→[39:32], 005→[47:40]; 16-bit mode adr 000→[15:0], 001→[31:16], 002→[47:32]. The topmost byte [47:40] is read■only/reserved. Write qualification: wb_wacc = module_sel && wb_we_i && (SINGLE_CYCLE || wb_ack_o); write_regs asserts only when wb_wacc=1 and is decoded by DWIDTH/address. 8-bit mode: adr 000→00001, 001→00010, 002→00100, 003→01000, 004→10000 (adr 005 is read■only). 16-bit mode: adr 000→00011, 001→01100, 002→10000. This yields 40/48 bits writable. Byte/halfword enables (wb_sel_i) correctly qualify partial writes in both bus widths. Reset: asynchronous reset via arst_i honoring ARST_LVL; synchronous wb_rst_i clears the internal wait/ACK FSM and registered read datapath. All logic resides in the bus clock domain and presents a Wishbone■compliant front end that fans write_regs to cop_regs and multiplexes read_regs (from cop_regs/cop_count) onto wb_dat_o according to DWIDTH

and address.

-- Register File --

Overview
- Register file implemented in cop_regs and exposed via cop_wb_bus (Wishbone slave).
- 48-bit readback space with five writable byte lanes (one top byte is read-only).
- Identical semantics for 8-bit and 16-bit Wishbone data widths.

Reset defaults
- timeout_value: all 1s.
- cop_irq_en: 00.
- debug_ena, stop_ena, wait_ena: 0.
- cop_ena: INIT_ENA (parameter-controlled).
- cwp: 0; clck: 0 (clck is sticky write-1).
- reload_count: 0; service_cop: 0.

Access types and protections
- cop_ena writable only when cwp == 0.
- cwp writable only when clck == 0.
- clck is write-once, sticky 1.
- debug_ena/stop_ena/wait_ena writable only when COP is disabled, or when the same write also sets cop_ena = 0.
- timeout_value writable only when cop_ena == 0; writes while enabled are ignored. Byte or halfword writes supported per bus width.
- clear_event is a write-1 pulse: 8-bit mode uses a dedicated byte; 16-bit mode uses bit [8] in the CTRL+CLR halfword.
- SERVICE (reload) requires two consecutive writes of SERV_WD_0 then SERV_WD_1 to the SERVICE address; any other write clears the arm (service_cop). Successful sequence generates a reload_count pulse.

Readback
- Reads return slices of a 48-bit read_regs vector containing control/status, a captured down-counter snapshot, and event/IRQ status.
- The topmost byte is read-only.
- The counter value is a snapshot captured on cop_clk edges; not a live combinational value.

Address map
- 8-bit Wishbone (adr[2:0] = 000..101):
- 000: CTRL (RW). Bits: [7:6]=cop_irq_en, [5]=debug_ena, [4]=stop_ena, [3]=wait_ena, [2]=cop_ena (gated by cwp), [1]=cwp (gated by clck), [0]=clck (sticky 1).
- 001: EVT_CLR (WO). write_data[0]=1 generates clear_event pulse.
- 010: TIMEOUT_LO (RW when disabled). timeout_value[7:0].
- 011: TIMEOUT_HI (RW when disabled). timeout_value[15:8].
- 100: SERVICE (WO). Two-key service sequence target.
- 101: Status (RO). Read-only byte.
- 16-bit Wishbone (adr[2:0] = 000..010):
- 000: CTRL+CLR (RW/WO). Lower 8 bits as CTRL above; bit [8]=1 generates clear_event pulse; upper bits otherwise reserved/ignored on write.
- 001: TIMEOUT (RW when disabled). 16-bit write to timeout_value[15:0].
- 010: SERVICE (WO). Two-key service target in top 16-bit word; only one byte in this halfword is write-enabled for service; the other byte is read-only.

Bus width adaptation
- On DWIDTH==8, write_data is internally formed as {byte, byte} so common decode logic applies; byte-specific semantics preserved.

Write acceptance
- Writes take effect only when wb_wacc is true. In single-cycle mode this is the request cycle; in registered mode it is the ACK cycle.

Effects
- A successful SERVICE two-key sequence or disabling cop_ena issues a reload request to the counter domain.
- clear_event or a successful reload clears the sticky event latch; expiry latches the event until cleared.

Read slicing
- 8-bit reads: adr 000..101 map to bytes [7:0], [15:8], [23:16], [31:24], [39:32], [47:40].
- 16-bit reads: adr 000..010 map to [15:0], [31:16], [47:32].

Notes
- Exact 48-bit read_regs field ordering is integration-dependent; guaranteed to include control/status, captured counter, and event/IRQ status with top byte read-only.
- Write decode summary: 8-bit — 000: CTRL, 001: EVT_CLR, 010: TIMEOUT_LO, 011: TIMEOUT_HI, 100: SERVICE, 101: RO. 16-bit — 000: CTRL+CLR, 001: TIMEOUT, 010: SERVICE (top word byte-enabled).

-- Watchdog Counter --

- Purpose: Parameterizable (COUNT_SIZE) down-counter watchdog that asserts a reset on expiry, provides an early warning interrupt, supports pause modes, and software servicing.
- Clocks: Runs in the cop_clk domain (startup oscillator). In scantestmode=1, cop_clk is sourced from the bus clock. IRQ and capture pipeline flops reside in the bus_clk domain.
- Resets: POR (active-low) clears cop_rst_o, reload_2, and the event latch; async_rst_b initializes the counter to all 1s; sync_reset clears bus-domain capture/IRQ flops. On POR release, reload_2 is preset to 1 to force the initial load of timeout_value.
- Operation: When enabled and not paused, the counter decrements by 1 each cop_clk. A reload loads timeout_value; pausing holds the current count. stop_counter is asserted when any of (debug_mode_i & debug_ena), (wait_mode_i & wait_ena), or (stop_mode_i & stop_ena) are true.
- Service/Reload: Software services via a two-key sequence (SERV_WD_0 then SERV_WD_1) that generates a one-cycle reload_count in the bus domain. Disabling cop_ena also requests reload. Handshake: reload_1 (bus_clk) asserts on (reload_count || !cop_ena || (reload_1 && !reload_2)) and is cleared by async_rst_b or sync_reset; reload_2 (cop_clk) samples reload_1 and is POR-cleared to 1. Any reload deasserts cop_rst_o.
- Reset output: cop_rst_o asserts when the counter reaches 0; it remains asserted if the counter is held at 0 while paused; any reload deasserts cop_rst_o.
- Early IRQ: Programmable thresholds relative to the live counter value via cop_irq_en[1:0]: 00=disabled, 01=<=16, 10=<=32, 11=<=64. IRQ is generated from the cop_clk condition and pipelined through two stages in the bus_clk domain.
- Counter capture: cop_capture is a stable snapshot of the counter in the bus_clk domain, updated on each detected cop_clk rising edge for software readback via Wishbone; software should use cop_capture rather than the live counter.
- Event latch: cop_event sets when cop_rst_o asserts and holds until event_reset; event_reset = (reload_count || clear_event). POR also clears cop_event.
- Configuration/Access: timeout_value, enable/pause mode bits, and IRQ enables are driven by

cop_regs via Wishbone. Writes to timeout_value are ignored while cop_ena==1; low-power/debug pause bits change only when COP is disabled or the write disables it.
- Defaults: On reset, timeout_value and cop_capture initialize to all 1s; cop_irq remains disabled until enabled via cop_regs.

-- Address Decode and Data Muxing --

cop_wb_bus implements a DWIDTH-selectable Wishbone slave that decodes a word-aligned address index (adr) and multiplexes a 48-bit internal read_regs bus onto wb_dat_o. Address decode is normalized with a key {eight_bit_bus, adr}, where eight_bit_bus = (DWIDTH == 8), so one mapping covers both widths. Read slice mapping: DWIDTH=8 (eight_bit_bus=1): adr 000 -> read_regs[7:0], adr 001 -> read_regs[15:8], adr 010 -> read_regs[23:16], adr 011 -> read_regs[31:24], adr 100 -> read_regs[39:32], adr 101 -> read_regs[47:40] (top byte is reserved/read-only). DWIDTH=16 (eight_bit_bus=0): adr 000 -> read_regs[15:0], adr 001 -> read_regs[31:16], adr 010 -> read_regs[47:32]. The data mux output (rd_data_mux) is returned on wb_dat_o per SINGLE_CYCLE: SINGLE_CYCLE=1 drives rd_data_mux combinationally for same-cycle reads; SINGLE_CYCLE=0 registers rd_data_mux on the first read cycle and presents it on the ACK cycle. wb_sel_i does not alter the read slice selection; the full DWIDTH slice is returned for each valid adr. The 48-bit read_regs bundle is sourced by cop_regs and cop_count; cop_wb_bus only slices and presents the selected segment as software-visible data per the above address mapping.

-- Read/Write Path and Handshake --

Wishbone transfer detection and response
- A transfer to cop_top is active when wb_cyc_i && wb_stb_i (module_sel).
- ACK generation is controlled by SINGLE_CYCLE:
- SINGLE_CYCLE=1: wb_ack_o asserts combinationally in the same cycle as module_sel; read data is driven combinationally; writes are accepted immediately in that cycle.
- SINGLE_CYCLE=0: wb_ack_o asserts exactly one cycle after module_sel; a 1-bit bus_wait_state FSM inserts one wait state. If CYC/STB are held continuously, ACK pulses every other cycle; masters should deassert STB between transfers for single-pulse ACKs.
- Write acceptance (wb_wacc): writes assert only when module_sel && wb_we_i && (SINGLE_CYCLE || wb_ack_o), ensuring correct timing in both single- and two-cycle modes.
- Byte/halfword enables (wb_sel_i): write qualification honors wb_sel_i; only enabled lanes generate write strobes downstream.

Read path (bus -> internal)
- cop_wb_bus returns slices of a 48-bit read_regs vector assembled from cop_regs/cop_count.
- DWIDTH-dependent mapping:
- DWIDTH=8: address selects successive bytes: [7:0], [15:8], [23:16], [31:24], [39:32], [47:40].
- DWIDTH=16: address selects successive halfwords: [15:0], [31:16], [47:32].
- Timing:
- SINGLE_CYCLE=1: rd_data is combinational in the same cycle as ACK.
- SINGLE_CYCLE=0: rd_data_reg captures on the first cycle of a read (module_sel && !wb_we_i); data is valid/stable on the ACK cycle.
- The topmost byte [47:40] is read-only and never mapped to a write strobe.
- cop_count provides a coherent snapshot of the cop_clk counter into bus_clk; software reads it via read_regs with data stability aligned to ACK.

Write path (bus -> cop_regs)
- Address and DWIDTH decode into a 5-bit write_regs vector; strobes assert only on wb_wacc.
- DWIDTH=8 mapping when accepted:
- adr 000 -> 00001 (control base)

- adr 001 -> 00010 (event clear)
- adr 010 -> 00100 (timeout low byte)
- adr 011 -> 01000 (timeout high byte)
- adr 100 -> 10000 (service sequence)
- adr 101 -> no write strobe (read-only top byte)
- DWIDTH=16 mapping when accepted:
- adr 000 -> 00011 (control base + event clear)
- adr 001 -> 01100 (timeout full-width write)
- adr 010 -> 10000 (service sequence in the top word; only the writable lanes are honored; [47:40] remains read-only)
- Data adaptation:
- DWIDTH=8: write payload is duplicated internally to 16 bits ({byte, byte}) for shared decodes; timeout_value also supports byte-specific updates.
- Field protections enforced by cop_regs:
- timeout_value updates only when cop_ena==0 (byte or full-width).
- debug_ena/stop_ena/wait_ena update only when COP is disabled or the same write disables it.
- cop_ena updates only when cwp==0; cwp updates only when clck==0; clck is sticky set-only.
- clear_event is a one-cycle write-1 pulse from either control+clear or event-clear decodes.

Reset and handshake robustness
- Asynchronous reset polarity (ARST_LVL) applies to the wait-state FSM and CDC elements; synchronous wb_rst_i clears bus-domain flops and bus_wait_state.
- After reset, read data and write strobes revert to safe defaults; POR initializes the counter reload path.
- The overall bus behavior is deterministic and width-adaptive: acceptance, ACK timing, and data mapping remain consistent across SINGLE_CYCLE modes and DWIDTH configurations.

-- Readback Bundle Mapping --

Readback Bundle (read_regs) is a 48-bit unified bus assembled by cop_regs and cop_count and exposed to software via cop_wb_bus. The bus interface slices read_regs onto the Wishbone data bus based on DWIDTH and the address.

Address-to-slice mapping (Wishbone read):
- DWIDTH == 8 (8-bit bus):
- Address 000 -> read_regs[7:0]
- Address 001 -> read_regs[15:8]
- Address 010 -> read_regs[23:16]
- Address 011 -> read_regs[31:24]
- Address 100 -> read_regs[39:32]
- Address 101 -> read_regs[47:40] (topmost byte; read-only)
- DWIDTH == 16 (16-bit bus):
- Address 000 -> read_regs[15:0]
- Address 001 -> read_regs[31:16]
- Address 010 -> read_regs[47:32] (topmost word; read path returns full 16 bits)

Writable vs read-only regions:
- read_regs[47:40] (topmost byte) is read-only from the bus; software can read it but cannot write it.
- The remaining 40 bits [39:0] form the writable register space. For DWIDTH == 16 at Address 010, only bits [39:32] are writable; bits [47:40] remain read-only.

Counter readback:
- The watchdog counter value is a bus-clock-domain snapshot captured by cop_count; the snapshot

bytes/halfword are included within read_regs so software reads a stable counter value via the above slices.

Read timing:
- SINGLE_CYCLE = 1: read data is combinational; wb_ack_o asserts in the same cycle as the request.
- SINGLE_CYCLE = 0: read data is registered on the pre-ACK cycle and presented on the ACK cycle; wb_ack_o asserts one cycle after the request.

Programming note:
- Exact field placement within read_regs (control/status, timeout, snapshot, event/IRQ status) is defined by cop_regs and cop_count; software should use those register definitions while relying on the slicing described above and the read-only nature of the top byte.

-- Writable vs Read-Only Regions --

Overview
- The register window is 48 bits wide and fully readable. The top byte [47:40] is reserved and read-only. Effective writable width is 40 bits.

DWIDTH=8 (8-bit bus)
- adr 000 → Writable: control fields (with protections); reads return [7:0].
- adr 001 → Writable as a write-1 pulse: clear_event; no stored bit; reads return [15:8].
- adr 010 → Writable: timeout low byte [7:0], only when COP is disabled; reads return [23:16].
- adr 011 → Writable: timeout high byte [15:8], only when COP is disabled; reads return [31:24].
- adr 100 → Write-only effect: service key sequence; generates reload on valid keys; reads return [39:32].
- adr 101 → Read-only: top byte [47:40].

DWIDTH=16 (16-bit bus)
- adr 000 → Writable: lower two bytes [15:0] (control fields plus clear_event pulse); reads return [15:0].
- adr 001 → Writable: timeout [15:0], only when COP is disabled; reads return [31:16].
- adr 010 → Partially writable: only byte [39:32]; byte [47:40] is read-only; reads return [47:32].

Write gating and lockouts
- cop_ena is writable only if cwp==0.
- cwp is writable only if clck==0.
- clck is sticky set-only (can be set, not cleared), which can indirectly lock cwp and cop_ena.
- debug_ena, stop_ena, wait_ena can change only while COP is disabled, or when the same write disables it (cop_ena=0).
- timeout_value writes are ignored when cop_ena==1.
- clear_event and service keys are write-only effects; reads always return the mapped status slices, not the last write value.

Write acceptance
- Writes take effect only when the bus transfer is accepted (wb_wacc per Wishbone handshake). In SINGLE_CYCLE=0 mode, write strobes assert on the ACK cycle.

-- Clock Domain Crossing --

Clock domains and CDC strategy: The design spans two primary clock domains—wb_clk_i (bus/regs/IRQ/event latch) and cop_clk (watchdog counter/reset). In scan/test mode (scantestmode=1), cop_clk is tied to wb_clk_i, eliminating CDC during test. No CDC exists inside the Wishbone front-end.

Bus → cop_clk crossings:
- Counter reload handshake: A bus-domain pulse (reload_count) and level enable (cop_ena) drive a sticky level (reload_1) that persists until observed in cop_clk as reload_2. The counter loads when reload_2 asserts, then reload_1 auto-clears. POR forces reload_2=1 for an initial load. This ensures no missed pulses and a clean, single load per request.
- Multi-bit timeout_value: Resides in the bus domain and is only writable when cop_ena=0. The counter samples it on reload_2, guaranteeing stability at capture time.
- Mode/enables: cop_ena, debug_ena, stop_ena, wait_ena cross as level controls to the cop_clk domain (e.g., for stop gating). Updates are restricted (e.g., while COP disabled) to minimize hazards and avoid metastability-sensitive edges.

cop_clk → bus crossings:
- Counter snapshot for software reads: cop_clk is resampled through two flops in the bus domain. A derived cop_clk_posedge pulse captures cop_counter into cop_capture for coherent, software-visible readback via read_regs.
- Early IRQ generation: A threshold compare and two-stage pipeline run in the bus domain using the live counter value from cop_clk. Pipelining plus coarse thresholds mitigate transient hazards, but software should prefer the synchronized cop_capture for accurate reads.
- Reset/event signaling: cop_rst_o is generated in cop_clk and treated as an asynchronous level that sets a sticky bus-domain event latch (cop_event). The latch is cleared synchronously by reload_count or clear_event.

Reset and test considerations:
- POR (active-low) resets cop_rst_o, reload_2, and cop_event to establish safe startup and force an initial counter load.
- async_rst_b (active-low) resets bus-domain flops and initializes the counter to all 1s; wb_rst_i (synchronous) clears bus-domain state. The wait-state FSM uses ARST_LVL for async behavior as appropriate.

CDC safety notes:
- Handshake semantics guarantee that reload requests are not lost and multi-bit data is sampled only when stable.
- Level controls crossing to cop_clk are treated as quasi-static and constrained to change only when safe (e.g., COP disabled), reducing metastability risk.
- In test mode, tying cop_clk to wb_clk_i collapses all crossings to single-domain behavior; handshake logic remains benign.

OPERATION

-- Reset and Initialization --

Reset sources and domains: an internal active-low async reset (async_rst_b) is derived from (arst_i ^ ARST_LVL) to support either external polarity; a synchronous bus reset (wb_rst_i) is used for bus-facing logic; a dedicated active-low power-on reset (por_reset_i) is used in the counter/event domain. Domains: bus domain runs on wb_clk_i with resets async_rst_b and wb_rst_i; counter domain runs on cop_clk with resets por_reset_i and async_rst_b. In scan/test (scantestmode=1), cop_clk=wb_clk_i; partitioning/behavior unchanged. Bus/Wishbone interface: on async_rst_b or

wb_rst_i, the wait/ACK FSM is cleared to idle (bus_wait_state=0); ACKs only occur on valid cycles post-reset; no write strobes are retained across reset. Register defaults (on async_rst_b or wb_rst_i): timeout_value=all 1s; cop_irq_en=00; debug_ena=0; stop_ena=0; wait_ena=0; cop_ena=INIT_ENA; cwp=0; clck=0; service_cop=0; reload_count=0; clear_event=0. Counter and outputs: por_reset_i clears cop_rst_o and cop_event, and asynchronously forces reload_2=1 to guarantee an initial load of timeout_value on POR release; async_rst_b resets the counter value to all 1s and clears bus-domain flops (resynchronizers, capture, reload_1, IRQ pipeline); wb_rst_i clears bus-domain flops/capture without requesting a reload; bus resets do not clear cop_event. Initialization load and reload handshake: upon POR release, reload_2=1 causes the counter to load timeout_value; in the bus domain, reload_1 asserts when cop_ena=0 or a valid service occurs, remains level-sticky until observed (reload_2), and is cleared by async_rst_b or wb_rst_i; reload_2 is async-cleared to 1 by POR. Outputs and event: cop_rst_o is deasserted by POR and any reload; it asserts when the counter reaches 0 and remains asserted until a reload or POR; cop_event latches high when cop_rst_o asserts and remains set until cleared by a service or clear_event pulse. Capture: cop_capture resets to all 1s in the bus domain and updates on the next detected cop_clk edge. Polarity and CDC: async resets inside submodules are active-low; ARST_LVL selects external reset polarity; reload_1/reload_2 form a CDC handshake between wb_clk_i and cop_clk ensuring safe initialization and reload across domains.

-- Configuration and Locking Flow --

Overview
- Configuration occurs in the bus clock domain through cop_wb_bus into cop_regs. cop_regs enforces all write gating and the two-stage locking that controls whether cop_ena and certain mode fields may change.

Reset defaults
- On reset: timeout_value = all 1s; cop_irq_en = 00; debug_ena = 0; stop_ena = 0; wait_ena = 0; cop_ena = INIT_ENA; cwp = 0; clck = 0.
- clck is not retained across reset (clears to 0). Both lock bits must be re-established after any reset.

Write acceptance and width
- Writes are accepted only when wb_wacc qualifies them: SINGLE_CYCLE mode accepts writes in the same cycle; two-cycle mode accepts writes on the ACK cycle. Masters must honor the handshake to avoid missed writes.
- Only 40 of 48 read_regs bits are writable; the topmost byte is read-only.
- For DWIDTH==8, the 8-bit payload is internally duplicated to 16 bits for uniform decode. Byte-specific timeout writes are supported on 8-bit buses.

Address map for configuration writes
- DWIDTH==8:
- Control register: adr 000.
- Event clear: adr 001.
- Timeout low byte: adr 010.
- Timeout high byte: adr 011.
- Service (two-key): adr 100.
- DWIDTH==16:
- Control + clear (clear_event at bit [8]): adr 000.
- Timeout full width: adr 001.
- Service (two-key): adr 010.

Enable and mode write rules
- cop_ena (enable) can be written only when cwp==0. If cwp==1, any attempt to change cop_ena is ignored.

- Mode bits (debug_ena, stop_ena, wait_ena) and timeout_value can only change while the COP is disabled (cop_ena==0), or when the same control write drives cop_ena from 1 to 0 in that transaction (atomic disable+update). This prevents enabling while simultaneously relaxing pause behavior.
- cop_irq_en follows normal write acceptance; it is not additionally gated by cwp/clck and does not require COP to be disabled.

Timeout programming
- Any write to timeout_value (full width or byte slices) is ignored while cop_ena==1. Disable the COP first or perform an atomic disable+update write as described above.

Service (reload) operation
- The counter reload is issued via a two-write key sequence to the Service address: first write SERV_WD_0 to arm, then immediately write SERV_WD_1 to generate a one-cycle reload_count pulse.
- Service/reload is not gated by cwp or clck and remains functional regardless of lock state.

Two-stage locking flow
- Stage 1: cwp (config write-protect)
- Function: when cwp==1, cop_ena becomes immutable (cannot be changed).
- Constraint: cwp can only be modified while clck==0.
- Stage 2: clck (config lock)
- Write-once: clck <= clck OR write_data[0]; once set to 1, it cannot be cleared except by reset.
- Effect: when clck==1, cwp is frozen and cannot change.

Practical implications of locking
- With clck==1, you cannot change cwp until reset.
- With cwp==1, you cannot change cop_ena. If the COP is enabled in this state, you cannot disable it, which indirectly prevents changes to mode bits and timeout_value (because those require the COP to be disabled). cop_irq_en remains configurable.
- Service remains available under all lock states.

Recommended configuration and lock sequence
- If INIT_ENA==1 and reconfiguration is needed, first disable the COP (while cwp==0).
- Program timeout_value (full width or byte-wise per DWIDTH).
- Set debug_ena, stop_ena, wait_ena as desired.
- Configure cop_irq_en as required.
- Enable the COP by writing cop_ena=1 (only possible if cwp==0).
- Optional: Set cwp=1 to freeze cop_ena.
- Optional: Set clck=1 to permanently freeze cwp (and thereby the enable state) until reset.
- Periodically service using the two-key sequence.

Side effects and CDC
- Disabling cop_ena or issuing a service request generates a reload request into the counter clock domain via the reload handshake; a reload clears cop_rst_o and loads timeout_value.
- The sticky event latch (cop_event) clears on reload or on an explicit clear_event write (DWIDTH-dependent addressing).

Notes and caveats
- Lock bits reset to 0; reassess and reapply locks after any reset.
- Writes in two-cycle mode take effect on ACK; ensure masters do not issue back-to-back writes without respecting the handshake.
- Configuration writes target only decoded writable bits; the read-only top byte is unaffected.

-- Service (Reload) Sequence --

Servicing (reloading) the watchdog is performed via a protected two-key write sequence to the Service register decoded as write_regs == 5'b10000. A write is only considered accepted when the Wishbone slave asserts ACK; in registered (two-cycle) mode the effective write occurs on the ACK cycle. Sequence: (1) First accepted write with data == SERV_WD_0 sets the step-latch (service_cop = 1); no reload occurs yet. (2) A subsequent accepted write to the same Service register with data == SERV_WD_1 while service_cop == 1 generates reload_count (a one bus_clk cycle pulse) and clears service_cop. Any accepted write to the Service register whose data is not SERV_WD_0 clears service_cop and aborts the sequence; intervening writes to other addresses do not affect service_cop. The second key does not need to be in the immediately following cycle; service_cop is only updated on Service register writes. Bus width behavior: DWIDTH == 8 duplicates the 8-bit payload internally ({byte, byte}); therefore SERV_WD_* constants must be repeated-byte values and software must write the specified 8-bit keys. DWIDTH == 16 requires writing the full 16-bit SERV_WD_* constants. Address decode: Service register is selected when write_regs == 5'b10000; for DWIDTH == 8 the address slot adr=100, for DWIDTH == 16 adr=010. Effects of a successful service (reload_count = 1): reload_1 asserts in the bus clock domain and, via CDC synchronization, reload_2 asserts in the watchdog (cop_clk) domain to synchronously load the counter from timeout_value, clear the watchdog reset output (cop_rst_o), and clear the sticky event latch (cop_event) via event_reset = reload_count || clear_event. CDC handshake: reload_1 is level-sticky until observed in cop_clk and auto-clears after reload_2 is seen. Additional reload sources: disabling the watchdog (cop_ena = 0) asserts a reload via the same handshake path; on power-on reset, reload_2 is forced high to guarantee an initial load. Constraints and recommendations: timeout_value may only be changed while the watchdog is disabled; service reload always loads the current timeout_value. Pause modes (debug/wait/stop when enabled) halt decrementing but do not block a reload; a service will load the counter and deassert cop_rst_o even if the counter was at 0 while paused. Perform the two-key writes as two distinct, acknowledged Wishbone transactions. Writing SERV_WD_1 without a valid prior SERV_WD_0 produces no reload. The Service register resides in the writable 5-byte region (byte [39:32]); unrelated read-only space does not affect service.

-- Countdown and Pause Modes --

The watchdog is a COUNT_SIZE-wide down-counter (cop_counter) clocked by cop_clk. On power-on reset (POR) or on any reload request, cop_counter loads timeout_value and cop_rst_o is cleared. When COP is enabled (cop_ena=1) and not paused, cop_counter decrements by one each cop_clk tick; when it reaches zero, cop_rst_o asserts and remains asserted until a reload occurs. Early-warning interrupts are generated as the live counter value crosses programmable thresholds (cop_irq_en=01/10/11 for <=16/<=32/<=64, 00 disables) and are not masked by pause.
Pause behavior is controlled by three modes—debug, wait, and stop—with the gating function: stop_counter = (debug_mode_i & debug_ena) | (wait_mode_i & wait_ena) | (stop_mode_i & stop_ena). When stop_counter is true, decrementing halts and cop_counter holds its current value; the clock is not gated. If held at zero, cop_rst_o remains asserted until a reload. The bus-domain capture of cop_counter remains constant while paused. Mode enable bits (debug_ena/wait_ena/stop_ena) can only be changed when cop_ena==0 or in a write that disables COP, and lock bits (cwp/clck) enforce protection. Disabling COP also requests a reload; after reload, counting resumes on the next cop_clk tick if not paused.

REGISTERS

CORE CONFIGURATION

CLOCKS