Generated Specification Document

INTRODUCTION

-- Overview --

i2c_master_top is a Wishbone-attached I2C master that provides a compact, memory-mapped register set for programming SCL timing, enabling the core, issuing byte-level commands (START, READ, WRITE, STOP), and observing status/interrupts. Internally it integrates a byte controller that sequences 8-bit transfers plus the ACK/NACK phase, and a bit-level engine that generates I2C-compliant waveforms with input synchronization/filtering, busy tracking, clock stretching support, and arbitration-loss detection. The host writes prescaler and control registers, loads TX data, and drives the shared command/status location to initiate operations; completion, ACK/NACK, busy, transfer-in-progress, and arbitration loss are reflected in status while an interrupt is optionally raised and cleared via the command register. Open-drain bus driving is implemented through output-enable controls, and the core honors slave clock stretching. Typical use is: set prescaler, enable the core (and IRQs), issue STA|WR or STA|RD, repeat WR/RD for subsequent bytes, then issue STO; status can be polled or interrupts used, with irq cleared by IACK.

-- Key Features --

- Wishbone-compatible I2C master (classic single-cycle acknowledge) with memory-mapped registers
- Programmable SCL frequency via 16-bit prescaler (PRER[15:0]); same clock drives input sampling/filter
- Simple software flow: enable (CTR.core_en), write TXR, issue CR (STA/RD/WR/STO); per-byte commands auto-clear on completion/arbitration loss
- Shared SR/CR address for efficient status/command access; CR writes accepted only when core enabled
- Rich interrupt support: irq_flag on byte-done or arbitration loss; enable via CTR.ien; IACK to clear; level-latched until cleared
- Multi-master ready: bus busy tracking, START/STOP detection, arbitration loss detection and recovery
- Byte-level controller: handles 8 data bits + 9th ACK/NACK, START/STOP generation, slave ACK capture (rxack), cmd_ack handshake, aborts on reset/arbitration loss
- Bit-level engine: prescaler-timed I2C waveforms; open-drain SCL/SDA via oen controls; double-synchronization + 3-sample majority filter; SCL clock stretching support; data sampled on SCL rising edge; arbitration checked during writes
- ACK/NACK semantics: master drives ACK after read bytes; slave ACK after writes captured and reported (rxack)
- Deterministic reset and auto-clear behavior: defined reset values; CR control bits self/auto-clear; IRQ latched until IACK
- Status reporting: rxack, tip (transfer in progress), i2c_busy, arbitration lost (latched), irq_flag
- Open-drain pad model: scl_o/sda_o fixed 0; scl_oen/sda_oen release lines for ‘1’
- Register map: PRER low/high (0x0/0x1), CTR (0x2), TXR/RXR (0x3), SR/CR (0x4)

-- Standards and Compliance --

Conforms to NXP I2C-bus specification UM10204 at the protocol level: supports START/STOP and repeated START (SDA transitions while SCL is high), 8-bit transfers followed by the 9th ACK/NACK, samples SDA on SCL rising edge, and maintains SDA stable while SCL is high. Supports multi-master operation with wired-AND arbitration (loss when attempting to send '1' while bus reads '0') and detects external STOP; releases the bus on arbitration loss. Honors clock synchronization and slave clock stretching by waiting for SCL to actually go high; prescaler stalls during stretch. Implements bus-busy tracking via filtered START/STOP detection. Provides digital spike suppression using a 3-sample majority filter with sampling derived from the prescaler; meeting Standard/Fast-mode spike requirements depends on system clock and prescaler configuration. Generates SCL timing via a programmable prescaler; Standard-mode (100 kHz) and Fast-mode (400 kHz) are achievable with suitable system clock, prescaler, and I/O; Fast-mode Plus and High-speed mode are not claimed. Does not automatically enforce tBUF or bus-free minimums; firmware must sequence operations to satisfy these limits. Electrical layer uses open-drain style outputs (drive 0 or release only) and requires external pull-ups and proper pad configuration to meet VOL/VOH and rise/fall timing. Addressing and framing are software-driven (7-bit or 10-bit sequences); the core transports bytes and ACK/NACK without enforcing address decoding. Not a full SMBus implementation (no bus-low timeout, clock-low timeout, or PEC); SMBus compliance requires additional system-level measures. Host interface follows Wishbone Classic single-cycle acknowledge for cyc&stb; compatible with standard usage, no burst/pipelined features. Status/interrupt flags align with I2C semantics (byte completion, ACK detect, transfer-in-progress, arbitration lost). Overall compliance depends on correct prescaler programming, system clock selection, and board-level pull-ups/pad characteristics.

-- Block Summary and Use Cases --

Block Summary: i2c_master_top is a Wishbone-attached I2C master with a compact, memory-mapped register set to control START/STOP, byte read/write, and ACK/NACK while reporting bus status, arbitration loss, and interrupts. It composes a byte-level controller (i2c_master_byte_ctrl) for 8-bit data plus the 9th ACK/NACK bit and a bit-level engine (i2c_master_bit_ctrl) for prescaled SCL generation, input filtering, clock stretching compliance, bus busy/START/STOP detection, and multi-master arbitration. PRER sets I2C SCL timing and filter cadence; CTR enables the core and interrupts; TXR/RXR carry transmit/receive bytes; SR (status) and CR (commands) share an address for low-latency reads/writes, with command bits auto-clearing on byte completion or arbitration loss. Open-drain SDA/SCL are driven via enable signals; IRQ flag latches until cleared; AL is latched until a new START or reset. Wishbone is classic single-cycle acknowledge. Use Cases: - Single-master write: configure PRER; set CTR.core_en; write TXR with slave address+W; issue CR.STA|WR; write data bytes via TXR and CR.WR; finish with CR.STO. Monitor SR.rxack for NACK, SR.tip for progress, SR.irq_flag for completion. - Single-master read: write TXR with slave address+R; issue CR.STA|WR; for each byte issue CR.RD; after each read, drive CR.ACK=0 to ACK more bytes and CR.ACK=1 to NACK the final byte; optionally issue CR.STO; read data from RXR when SR.irq_flag asserts. - Combined write-then-read (repeated START): send slave address+W and register address bytes (CR.WR); without STOP, issue CR.STA|WR with address+R; then CR.RD with ACK policy (ACK intermediate, NACK last); finish with CR.STO. - Interrupt-driven operation: set CTR.ien; issue commands; on interrupt or SR.irq_flag, read SR to determine completion/NACK/AL; clear IRQ via CR.IACK; continue or retry as needed. - Polling mode: poll SR.tip and SR.irq_flag to detect end-of-byte; check SR.rxack for NACK and SR.al for arbitration loss; retry when SR.i2c_busy deasserts. - Multi-master arbitration: if SR.al=1, the core releases the bus; AL remains latched until a new START; wait for SR.i2c_busy=0, then reissue the transaction. - Clock stretching: bit core holds SCL high release until the bus actually goes high; software sequences are unchanged. - Speed tuning: adjust PRER to meet standard/fast-mode targets and input filter cadence. - Typical applications: EEPROM/flash programming, sensor configuration/readout, PMIC control, boot-time probing.

-- Dependencies and Assumptions --

Single synchronous clock domain with an asynchronous active-low nReset; internal synchronous rst is used by submodules. Wishbone interface is classic single-cycle: no STALL/ERR, wb_ack_o asserts in the same cycle as wb_cyc_i & wb_stb_i; the host must present valid adr/dat/we and sample ack/dat that cycle. Byte-addressed register map at offsets 0x0..0x4; decode and byte-lane mapping must align so PRER, CTR, TXR/RXR, SR, and CR are at the intended addresses (reads at 0x3 return RXR, writes at 0x3 update TXR; reads at 0x4 return SR, writes at 0x4 update CR). CTR.core_en must be set before any CR command is accepted; IACK is ignored if core_en=0; CTR.ien gates the interrupt output. PRER must be non-zero and large enough to meet I2C timing and the bit-core glitch filter cadence (filter steps at clk_cnt>>2, choose PRER≥4). SDA/SCL must be connected to open■drain pads with external pull■ups; scl_o/sda_o are fixed 0, scl_oen/sda_oen control release; scl_i/sda_i must sense the physical bus. Multi■master operation is assumed: arbitration loss can occur and is latched in SR.al until a new START or reset; SR.i2c_busy reflects the physical bus state; software must handle AL and restart transactions. Command constraints: only one of RD or WR per byte; STA may be combined with RD/WR; STOP optional; CR[7:4] auto■clear on byte done or AL; CR[2:0] self■clear; SR.rxack=0 indicates slave ACK; SR.tip reflects active RD/WR. Clock stretching is honored; byte completion latency is variable, software must not assume fixed timing. Data path assumptions: TXR must be loaded before WR; RXR at 0x3 holds the last received byte; for reads, CR.ACK sets the master’s 9th■bit response (0=ACK, 1=NACK). Reset behavior: on nReset/rst, all cores release SDA/SCL, clear status/commands, and return to idle; arbitration loss also aborts to idle. If the system Wishbone fabric requires wait states, an adapter is needed; this core assumes zero■wait acknowledge. Module dependencies: PRER drives bit■core clk_cnt; CTR.core_en enables subcores; byte_ctrl advances only on bit■core cmd_ack and aborts on rst or i2c_al; bit■core double■syncs and majority■filters inputs, supports clock stretching, detects AL when releasing ‘1’ but reading ‘0’, and sets busy on bus START/STOP. Integration must verify address width/byte■lane mapping and that RXR (not TXR) is returned on reads at 0x3.

IO PORTS

-- Clock and Reset --

- Clock domain
- Single synchronous system clock for the Wishbone interface, register file, status/IRQ logic, and the byte/bit controllers. i2c_master_top drives the child controllers’ clk input; wb_ack_o and wb_dat_o are synchronous to this clock.
- Bit-level timing is generated by the 16-bit prescaler PRER[15:0]. PRER is forwarded to the bit core as clk_cnt, which produces clk_en strobes to advance internal FSMs and to update the input glitch/majority filter. Prescaler updates take effect on the next reload.
- SCL/SDA inputs are double-synchronized and majority-filtered (3-tap). Data sampling occurs on the filtered SCL rising edge.
- Clock stretching is supported: if a slave holds SCL low after release, clk_en stalls and FSM progression pauses until SCL is observed high.
- Reset
- Two resets are supported and propagated to the byte and bit controllers: nReset (asynchronous, active-low) and rst (synchronous). Either reset returns controllers to idle, releases the bus (scl_oen=1, sda_oen=1), clears command handshakes, and deasserts internal controls.

- Top-level reset defaults:
- PRER[15:0] = 0xFFFF (PRER_LO=0xFF, PRER_HI=0xFF).
- CTR = 0x00 (core_en=0, ien=0).
- TXR = 0x00.
- SR flags cleared: rxack=0, i2c_busy=0, al=0, tip=0, irq_flag=0.
- wb_inta_o deasserted; it only asserts when SR.irq_flag=1 and CTR.ien=1.
- Core enable gating: CTR.core_en must be 1 to operate. When core_en=0, CR writes are ignored, bit/byte engines remain idle, and the bus is released. Deasserting core_en halts new commands; keep core_en=0 while programming PRER to avoid unintended bus activity.
- Arbitration loss handling: i2c_al forces an immediate abort to idle in the byte/bit controllers. SR.al remains set until a new START or a reset clears it.
- Command/IRQ clearing: CR command bits auto-clear on completion or arbitration loss. irq_flag is cleared by writing CR.IACK=1 or by reset; reset also clears irq_flag and disables interrupts (CTR.ien=0).

-- Host Bus Interface --

- Protocol: Wishbone classic (B4-style), synchronous, memory-mapped 8-bit register file.
- Clock/Reset: wb_clk_i is the bus clock. wb_rst_i is a synchronous, active-high reset that initializes all registers to their reset defaults.
- Signals used: wb_cyc_i, wb_stb_i, wb_we_i, wb_adr_i, wb_dat_i[7:0], wb_dat_o[7:0], wb_ack_o, wb_inta_o. If present, wb_sel_i is supported only on sel[0]. No use of ERR/RTY/STALL/CTI/BTE.
- Data width and lanes: 8-bit data path. If wb_sel_i exists, only sel[0]=1 is meaningful; other lanes are ignored. Multi-byte accesses are not supported.
- Addressing: Byte-oriented offsets relative to the block base. Defined addresses are 0x0–0x4:
- 0x0 PRER[7:0] (rw)
- 0x1 PRER[15:8] (rw)
- 0x2 CTR (rw)
- 0x3 TXR (write) / RXR (read)
- 0x4 SR (read) / CR (write)
- Handshake and timing: A bus access is requested when wb_cyc_i=1 and wb_stb_i=1. The slave returns exactly one single-cycle wb_ack_o pulse per accepted access; no wait-states are inserted. Read data (wb_dat_o) is valid when wb_ack_o is asserted. Writes take effect on the wb_ack_o cycle.
- Access latency: Zero added wait-state behavior; every valid request is acknowledged without stretching. Back-to-back accesses are supported; each request receives its own single-cycle acknowledge.
- Read path: wb_dat_o is a mux of the internal registers selected by wb_adr_i. Reading 0x3 returns RXR. Reading 0x4 returns SR. Reads have no side effects (e.g., irq_flag is not cleared on read).
- Write path: Writes to 0x0, 0x1, 0x2, and 0x3 update PRER, CTR, and TXR respectively. Writes to 0x4 update CR, but are accepted only if CTR.core_en=1; if core_en=0 the write is acknowledged with no effect. CR command bits exhibit auto-clear behavior as defined in the register section.
- Interrupt: wb_inta_o is active-high and equals (irq_flag & CTR.ien). irq_flag is cleared only by writing CR.IACK=1.
- Unmapped/reserved addresses: Only 0x0–0x4 are defined. Accesses outside this range are acknowledged and have no side effects; reads return 0x00.
- Reset defaults (observed via bus): PRER=16'hFFFF at 0x0/0x1, CTR=8'h00 at 0x2, TXR=8'h00 at 0x3 (write view), SR reflects idle bus at 0x4 (irq_flag=0, tip=0; i2c_busy mirrors bus state).

-- I2C Pad Interface --

- Electrical behavior: SCL/SDA are open-drain. The core never drives a logical '1'; it only drives low (0) or releases the line. External pull-up resistors are required, and pad cells must support bidirectional

tri-state with drive-low capability (no push-pull high).
- Pad ports and roles:
- scl_pad_i, sda_pad_i: raw pad inputs to the core.
- scl_pad_o, sda_pad_o: tied to logical 0 (drive-low data path).
- scl_pad_oen, sda_pad_oen: output-enable controls (0 = actively drive 0, 1 = release/high-Z so pull-ups yield logic 1).
- Top-level mapping: i2c_master_top connects pads to internal bit-level signals as scl_pad_i→scl_i, sda_pad_i→sda_i, scl_pad_o→scl_o (0), sda_pad_o→sda_o (0), scl_pad_oen→scl_oen, sda_pad_oen→sda_oen.
- Input handling: Pad inputs are double-synchronized into the core clock domain and then glitch-filtered using a 3-sample majority. The filter update rate is derived from the prescaler (PRER), ensuring robust START/STOP and data edge detection across clock configurations.
- Timing and edge use: Data is sampled on the filtered SCL rising edge. START is detected as SDA falling while SCL is high; STOP as SDA rising while SCL is high.
- Read/write pad behavior:
- Write bits: sda_oen=0 drives a '0'; sda_oen=1 releases to send a '1'.
- Read bits: SDA is released (sda_oen=1); the slave drives data; the master samples on filtered SCL rising.
- ACK after read (9th bit): master behavior per CR.ACK (0=ACK drives 0 via sda_oen=0; 1=NACK releases via sda_oen=1). After write bytes, the slave ACK is read with SDA released and latched to status (rxack).
- Clock stretching: When SCL is released high but filtered SCL remains low, the core stalls its prescaler and waits until filtered SCL is high, honoring slave clock stretching at the pad level.
- Arbitration and multi-master: During write, if the master releases '1' (sda_oen=1) yet filtered SDA is low, arbitration loss is flagged. A STOP observed that was not initiated by the core also flags loss. On reset or arbitration loss, both lines are released (scl_oen=1, sda_oen=1).
- Bus monitoring: Bus busy is derived from filtered START/STOP detection and is tracked regardless of which master drives the bus.
- Reset/disable behavior: On reset, and when CTR.core_en is disabled, the pad outputs are released (output-enables set to 1), leaving the bus idle and high-Z from the core’s perspective.
- Compliance/timing: PRER sets SCL low/high periods and the input filter sample rate, enabling compliant I2C timing with different system clocks. Ensure board-level pull-ups and appropriate pad cells to meet I2C electrical requirements.

-- Interrupts --

Interrupts are level-driven and sourced by two events: (1) byte-complete (cmd_ack) from the byte controller, including completion after an optional STOP; (2) arbitration lost (i2c_al) from the bit controller. The latched interrupt flag (SR[0] irq_flag) is set when either event occurs and remains set until explicitly cleared. The arbitration-lost status (SR[5] al) is latched separately: it is set on i2c_al and held until a new START (STA) is issued or reset; clearing irq_flag does not clear SR[5]. The interrupt output wb_inta_o equals irq_flag AND CTR[6] ien. When ien=0, wb_inta_o is deasserted but irq_flag still latches for polling; setting ien while irq_flag=1 asserts wb_inta_o immediately. Clear the interrupt by writing CR[0]=1 (IACK) at address 0x4; IACK self-clears on the next cycle, and CR writes (including IACK) are accepted only when CTR[7] core_en=1. IACK may be combined in the same write with new command bits (e.g., STA|WR). Command bits CR[7:4] auto-clear on byte completion or arbitration loss; CR[2:0] clear each cycle. tip (SR[1]), busy, and rxack are informative and do not generate interrupts. On reset, CTR.ien=0 and irq_flag=0. Because wb_inta_o is level-sensitive, additional byte-complete events while irq_flag=1 do not create new edges; software should clear irq_flag between events if edge-triggered handling is required.

-- Signal Definitions --

- Conventions
- All signals are synchronous to wb_clk_i unless noted. Active level is high unless noted. Widths shown as [msb:lsb]. Directions are relative to top-level module i2c_master_top.

- Clock and Reset
- wb_clk_i (in, 1): System/Wishbone clock driving all logic.
- wb_rst_i (in, 1, active-high): Synchronous system reset; initializes registers and clears status.

- Wishbone Slave Interface (8-bit data)
- wb_adr_i (in, N): Address bus; lower bits select internal registers at 0x0..0x4.
- wb_dat_i[7:0] (in): Write data bus.
- wb_dat_o[7:0] (out): Read data bus.
- wb_we_i (in, 1): Write enable (1=write, 0=read).
- wb_cyc_i (in, 1): Bus cycle valid.
- wb_stb_i (in, 1): Strobe/transfer request.
- wb_ack_o (out, 1): Transfer acknowledge; asserted for exactly one cycle per accepted access when wb_cyc_i & wb_stb_i are high.
- wb_inta_o (out, 1): Level interrupt request; equals (irq_flag & CTR.ien).

- I2C Pad Interface (open-drain style)
- scl_i (in, 1): Sampled SCL line from pad.
- sda_i (in, 1): Sampled SDA line from pad.
- scl_o (out, 1): Constant 0 (for open-drain drive); use scl_oen to control line.
- sda_o (out, 1): Constant 0 (for open-drain drive); use sda_oen to control line.
- scl_oen (out, 1): SCL output enable; 0=drive 0 (pull low), 1=release (external pull-up drives high).
- sda_oen (out, 1): SDA output enable; 0=drive 0 (pull low), 1=release (external pull-up drives high).

- Register/Datapath Internals
- prescale[15:0] (int): PRER replicated prescaler value to bit controller (SCL timing and input filter rate).
- txr[7:0] (int): Transmit data byte (from TXR).
- rxr[7:0] (int): Received data byte (to RXR).

- Status/Interrupt Internals (reflected in SR and interrupt)
- rxack (int, 1): Last received ACK from slave (0=ACK, 1=NACK); maps to SR[7].
- i2c_busy (int, 1): Bus busy between START and STOP; maps to SR[6].
- al (int, 1): Arbitration lost (latched until START or reset); maps to SR[5].
- tip (int, 1): Transfer in progress during RD/WR; maps to SR[1].
- irq_flag (int, 1): Latched interrupt flag; set on byte done or arbitration lost; cleared by CR.IACK; maps to SR[0].

- Command/Control Strobes (derived from CR when CTR.core_en=1)
- sta (int, 1): Request START condition (CR.STA).
- stp (int, 1): Request STOP condition (CR.STO).
- rd (int, 1): Request read of one byte (CR.RD).
- wr (int, 1): Request write of one byte (CR.WR).
- ack_in (int, 1): ACK value driven after read (0=ACK, 1=NACK) from CR.ACK.
- iack (int, 1): Interrupt acknowledge pulse when CR.IACK is written.

- Byte Controller Submodule Interface (i2c_master_byte_ctrl)
- clk (in): wb_clk_i.
- nReset (in, active-low async): Optional asynchronous reset from top-level reset infrastructure.
- rst (in, active-high): Synchronous reset (from wb_rst_i).

- i2c_al (in, 1): Arbitration lost from bit controller.
- start (in, 1): sta.
- stop (in, 1): stp.
- read (in, 1): rd.
- write (in, 1): wr.
- din[7:0] (in): txr.
- ack_in (in, 1): ack_in.
- core_ack (in, 1): cmd_ack from bit controller (bit-level operation done).
- core_rxd (in, 1): dout from bit controller (serial input data bit).
- core_cmd[3:0] (out): Encoded START/STOP/WRITE/READ command to bit controller.
- core_txd (out, 1): Data bit to bit controller (serial output data bit).
- cmd_ack (out, 1): Byte/stop operation done pulse to top-level (sets irq_flag, clears CR ops).
- dout[7:0] (out): rxr.
- ack_out (out, 1): rxack.

- Bit Controller Submodule Interface (i2c_master_bit_ctrl)
- clk (in): wb_clk_i.
- nReset (in, active-low async): Optional asynchronous reset.
- rst (in, active-high): Synchronous reset.
- ena (in, 1): Core enable (CTR.core_en).
- clk_cnt[15:0] (in): prescale (PRER) for SCL timing/filtering.
- scl_i (in, 1): Sampled SCL.
- sda_i (in, 1): Sampled SDA.
- cmd[3:0] (in): core_cmd from byte controller.
- din (in, 1): core_txd from byte controller.
- dout (out, 1): core_rxd to byte controller.
- cmd_ack (out, 1): core_ack to byte controller.
- busy (out, 1): i2c_busy to status.
- al (out, 1): Arbitration lost to status/byte controller.
- scl_o (out, 1): Constant 0 (open-drain drive reference to pad mux).
- sda_o (out, 1): Constant 0 (open-drain drive reference to pad mux).
- scl_oen (out, 1): As defined above for pad control.
- sda_oen (out, 1): As defined above for pad control.

ARCHITECTURE

-- Top-Level Block Diagram --

Blocks and interconnect
- Wishbone Slave + Register Bank
- External ports: wb_clk_i, wb_rst_i, wb_adr_i[2:0], wb_dat_i[7:0], wb_dat_o[7:0], wb_we_i, wb_stb_i, wb_cyc_i, wb_ack_o, wb_inta_o.
- Address map and mux on wb_dat_o:
- 0x0: PRER[7:0]
- 0x1: PRER[15:8]
- 0x2: CTR (bits: ien, core_en)
- 0x3: TXR (write) / RXR (read)

- 0x4: SR (read) / CR (write)
- wb_ack_o: classic single-cycle acknowledge when wb_cyc_i & wb_stb_i are asserted.
- Write gating: CR writes are accepted only when CTR.core_en=1.
- Command Decode + Status/IRQ Logic
- Inputs: CR write data (STA, STO, RD, WR, ACK, IACK), CTR.core_en.
- Outputs to byte controller: sta, stp, rd, wr, ack_in; iack used to clear IRQ.
- Status inputs: rxack (from byte controller), i2c_busy and al (from bit controller), cmd_ack (byte done).
- SR fields driven:
- rxack <= byte controller ack_out
- i2c_busy <= bit controller busy
- al <= bit controller arbitration loss
- tip <= rd | wr (held until cmd_ack)
- IRQ: irq_flag set on (cmd_ack | al); cleared by CR.IACK; wb_inta_o <= irq_flag & CTR.ien.
- CR command bits auto-clear on byte done (cmd_ack) or arbitration loss (al).
- Byte-Level Controller (i2c_master_byte_ctrl)
- Upward interface: start/stop/read/write/ack_in from command logic; cmd_ack (done) back to status; dout[7:0] to RXR; ack_out to SR.rxack.
- Data path: din[7:0] from TXR; dout[7:0] to RXR (read-back at 0x3).
- Downward interface to bit controller: core_cmd[3:0], core_txd; receives core_ack and core_rxd.
- Bit-Level Controller (i2c_master_bit_ctrl)
- Prescaler input: clk_cnt[15:0] <= PRER[15:0] for SCL period generation and input filter sampling.
- Handshakes: core_cmd[3:0], core_txd from byte controller; returns core_ack, core_rxd.
- Status outputs: busy (i2c_busy), al (arbitration loss) to status logic.
- I2C pads: scl_i, sda_i (filtered/synchronized inputs from pins); scl_oen, sda_oen (open-drain enable outputs to pins).
- I2C Pad Interface
- External pins: SCL and SDA.
- Inputs to bit controller: scl_i <= SCL pin, sda_i <= SDA pin (after optional filtering in bit controller).
- Open-drain drives: scl_o and sda_o are held low; scl_oen/sda_oen control release/driving of lines to implement open-drain behavior.

Principal signal flow

- Prescaler path: PRER[15:0] -> bit controller clk_cnt -> SCL timing and input filtering.
- Host data path: TXR -> byte controller din; byte controller dout -> RXR (readable at 0x3).
- Command path: CR[STA,STO,RD,WR,ACK,IACK] -> command decode (gated by CTR.core_en) -> byte controller control inputs; CR bits auto-clear on cmd_ack or al.
- Byte-to-bit path: byte controller core_cmd/core_txd -> bit controller; bit controller core_ack/core_rxd -> byte controller.
- Status/IRQ path: rxack, i2c_busy, al, tip aggregated into SR at 0x4; irq_flag set on cmd_ack or al; wb_inta_o asserted when irq_flag & CTR.ien.

Reset and enable

- wb_rst_i resets registers and internal state; CTR.core_en gates command acceptance; arbitration loss and resets return controllers to idle and release I2C lines.

-- Module Hierarchy --

- Hierarchy chain: i2c_master_top → i2c_master_byte_ctrl → i2c_master_bit_ctrl (no deeper submodules).
- i2c_master_top (top-level)
- Role: Wishbone-attached I2C master; exposes PRER[15:0], CTR, TXR, SR/CR; arbitrates host START/STOP/READ/WRITE, manages status/interrupts.
- Instantiates: i2c_master_byte_ctrl.
- External interfaces: Wishbone bus (wb_adr_i, wb_dat_i/o, wb_ack_o, wb_inta_o); I2C pads via

bit-level outputs (scl_i/sda_i, scl_o fixed 0, sda_o fixed 0, scl_oen/sda_oen for open-drain control).
- Provides prescaler (PRER) and core enable (CTR.core_en) downstream.
- i2c_master_byte_ctrl (child of top)
- Role: Byte-level sequencer; issues START/STOP and 8 data bits + 9th ACK/NACK; bridges TXR→din and dout→RXR; handshakes completion.
- Instantiates: i2c_master_bit_ctrl.
- Driven by top: start (STA), read (RD), write (WR), stop (STO), ack_in (ACK), din[7:0] (TXR), enable, prescale, reset.
- Returns to top: cmd_ack (byte done), irxack (rxack), dout[7:0] (RXR), i2c_busy, i2c_al.
- i2c_master_bit_ctrl (child of byte ctrl)
- Role: Bit-level engine; generates I2C-compliant SCL/SDA timing using prescaler; input sync/filter, clock stretching, bus busy and arbitration loss; drives open-drain enables (scl_oen/sda_oen).
- Driven by byte ctrl: core_cmd[3:0] (START/STOP/WRITE/READ), core_txd (write/ACK/NACK bit), enable, prescale.
- Returns to byte ctrl: core_ack (bit operation complete), core_rxd (sampled bit), busy, al.
- Structural notes
- SR/CR share address at top; CR command bits auto-clear on byte completion/arbitration loss; CR gating by CTR.core_en.
- Open-drain pads are implemented via oen signals; scl_o/sda_o are hard-wired low; release = oen=1, drive-low = oen=0.

-- Datapath Overview --

- Wishbone data paths: Host writes map to PRER[15:0], CTR, TXR, and CR; reads return PRER, CTR, RXR, and SR (SR on reads from the shared SR/CR address). CR writes are ignored when CTR.core_en=0.
- Prescaler: PRER feeds the bit-level prescaler down-counter to generate clk_en (bit timing) and sets the input filter cadence; slave_wait pauses the prescaler during clock stretching.
- Byte-level datapath: TXR loads the shift register sr[7:0] on ld; sr shifts each core_ack. On READs, core_rxd (sampled sSDA) enters sr bit0; on WRITEs, core_txd drives the SDA level from sr[7]. dcnt[2:0] counts the 8 data bits; cnt_done triggers the 9th ACK/NACK phase. ack_out captures the slave's ACK after writes; ack_in (CR.ACK) supplies the master's post-read ACK/NACK bit. At byte completion, dout mirrors sr to RXR.
- Bit-level datapath: scl_i/sda_i pass through double-flop sync and a 3-tap majority filter to produce sSCL/sSDA. sSDA is sampled on the sSCL rising edge to form core_rxd. Open-drain outputs use fixed 0 drivers with enables (scl_oen/sda_oen) to release or pull low; sda_oen releases on READ, follows data on WRITE and ACK bits.
- Status and bus state: SR aggregates rxack, i2c_busy (from START/STOP detection, including other masters), al (arbitration loss latch), tip (active rd|wr), and irq_flag. al is latched until a new START or reset. tip reflects requested activity and clears on byte done; actual progress is governed by core_ack.
- Interrupt datapath: irq_flag sets on cmd_ack (byte done, optional STOP) or al, is masked by CTR.ien to form wb_inta_o, and is cleared by CR.IACK.
- Top-level muxing: wb_dat_o selects PRER, CTR, TXR/RXR, or SR for reads; write-side updates flow into the corresponding registers and through the byte/bit cores as above.

-- Control Path and FSMs --

Three-tier control path and FSMs
- Top-level register/IRQ glue issues byte commands; a byte-level FSM sequences bytes and ACK phases; a bit-level FSM generates I2C waveforms with prescaled timing, filtering, clock stretching and arbitration handling.

Top-level control path (i2c_master_top)

- Command decode: Writes to Control Register (CR) are accepted only when CTR.core_en=1. CR bits: STA (start), STO (stop), RD, WR, ACK (ack_in after reads), IACK (interrupt acknowledge). STA may be combined with RD or WR in the same write to issue a START or repeated START.
- Command lifetime: CR[7:4] (STA, STO, RD, WR) are single-shot strobes that auto-clear on byte completion (done) or arbitration loss (i2c_al). CR.ACK and CR.IACK self-clear each cycle; IACK clears the latched interrupt.
- Status/flags: TIP reflects an active transfer command (rd|wr until auto-clear). rxack mirrors the last slave response after a write (0=ACK, 1=NACK). i2c_busy (from bit core) indicates actual bus occupancy between START and STOP. Arbitration loss latch: al <= i2c_al | (al & ~sta); al holds until a new START is issued or reset.
- Interrupts: irq_flag <= (done | i2c_al | irq_flag) & ~iack; wb_inta_o = irq_flag & CTR.ien. done is asserted by the byte FSM when a byte (and optional STOP) completes.
- Datapath handshakes: TXR provides the transmit byte to the byte FSM; RXR is updated with the received byte on reads. The byte FSM's ack_out (irxack) drives SR.rxack.
- WB timing/reset: wb_ack_o is a single-cycle acknowledge for any valid access. Synchronous/asynchronous resets return all logic to idle and release SCL/SDA.

Byte-level controller FSM (i2c_master_byte_ctrl)
- Purpose: Convert CR-level byte operations into bit-level START/STOP/READ/WRITE sequences with correct 9th-bit ACK/NACK behavior.
- States: IDLE, START, WRITE, READ, ACK, STOP.
- IDLE: Waits for a go request (RD, WR, or STO) optionally with STA. Loads shift register/counter as needed. Branches to START if requested; otherwise to READ, WRITE, or STOP.
- START: Issues core_cmd=START; on core_ack, branches to READ or WRITE according to request and initializes the data phase.
- WRITE: Iterates core_cmd=WRITE for 8 data bits; shifts out on each core_ack. When the bit counter completes, transitions to ACK and issues core_cmd=READ to sample the slave's ACK.
- READ: Iterates core_cmd=READ for 8 data bits; shifts in on each core_ack. When the bit counter completes, transitions to ACK and issues core_cmd=WRITE to drive the master's ACK/NACK (ack_in) as the 9th bit.
- ACK (9th bit): Drives or releases SDA as required. On core_ack, latches ack_out (slave's ACK after a write), asserts cmd_ack (byte done), and either proceeds to STOP if requested or returns to IDLE.
- STOP: Issues core_cmd=STOP; on core_ack, asserts cmd_ack and returns to IDLE.
- Handshakes and aborts: cmd_ack pulses exactly once per completed byte (and optional STOP). Any reset or i2c_al aborts to IDLE, clears handshakes, and releases the bus.

Bit-level waveform FSM (i2c_master_bit_ctrl)
- Purpose: Generate I2C-compliant SCL/SDA waveforms for START, STOP, and per-bit READ/WRITE with prescaled timing, input synchronization/filtering, clock stretching support, and arbitration/bus-busy handling.
- Timing/filtering: Inputs are double-synchronized and fed through a 3-tap majority filter to form sSCL/sSDA. A prescaler (PRER[15:0]) generates clk_en for sub-state advancement and sets the input filter's effective sampling. slave_wait halts clk_en when SCL is stretched low by a slave.
- Command micro-sequences (advance only when clk_en=1 and not slave_wait):
- START: Create a legal START (SDA falling while SCL high), then pull SCL low; cmd_ack asserted at completion.
- STOP: With SCL released high, release SDA high to create STOP; cmd_ack asserted at completion.
- READ bit: Release SDA, raise and hold SCL to sample; capture dout on SCL high; cmd_ack when SCL returns low.
- WRITE bit: Drive SDA with din while toggling SCL; sda_chk active during the high phase to detect arbitration; cmd_ack when SCL returns low.
- Bus state and arbitration: busy is set on a filtered START and cleared on a filtered STOP. Arbitration

loss is detected if the master releases SDA high but sSDA reads low during WRITE, or if an unexpected STOP occurs; on al, lines are released and the FSM returns to idle.
- Open-drain pads: scl_o and sda_o are tied low; scl_oen/sda_oen control drive (0) vs release (1) to implement open-drain behavior.

-- Prescaler and Timing Generation --

The top-level exposes a 16-bit prescaler PRER[15:0] at addresses 0x0 (low) and 0x1 (high), reset to 0xFFFF. Software programs PRER to set the I2C bit timing; larger values slow the bus and increase input deglitching. PRER is wired to i2c_master_bit_ctrl.clk_cnt and drives a down-counter (cnt) that generates clk_en pulses: cnt reloads from clk_cnt and counts down; when it reaches zero (or on reset/!ena/sync), it reloads and asserts clk_en. The bit-level FSM advances only on clk_en, so PRER defines the fundamental timing quantum for SCL/SDA waveform generation. SCL and SDA are open-drain (scl_o/sda_o drive 0); timing is effected by toggling scl_oen/sda_oen in FSM micro-states. Data is sampled on the filtered SCL rising edge (sSCL↑), and START/STOP detection uses filtered lines. The input filter uses a 3-sample majority on synchronized SCL/SDA, sampled every (clk_cnt >> 2) cycles to produce sSCL/sSDA, tying deglitching cadence to PRER. Clock stretching is honored by freezing timing when SCL is released but remains low: if scl_oen transitions to released-high while sSCL stays low, slave_wait holds cnt (no clk_en) until sSCL rises, extending the low phase without advancing the FSM. Enable gating: the top-level CTR.core_en controls acceptance of command writes; within the bit core, ena gates the prescaler-driven timing. When disabled (!ena), cnt is reloaded and clk_en occurs only for reset/sync, halting timing progression. Result: PRER provides a deterministic division from the system clock to both I2C bit timing and filter sampling. Bus speed scales inversely with PRER; absolute SCL timing also depends on the system clock, FSM sequencing, and external pull-up/RC characteristics during released-high phases.

-- Input Synchronization and Glitch Filtering --

All input synchronization and glitch filtering are implemented in i2c_master_bit_ctrl; i2c_master_top programs PRER[15:0], which sets both SCL timing and the filter sampling cadence. Raw scl_i and sda_i are first passed through two flip-flops (to the system clock) to mitigate metastability. Each line then feeds a 3-sample shift register that is updated at a prescaled cadence derived from PRER; a 2-of-3 majority vote produces the filtered sSCL and sSDA. Spikes shorter than the sampling interval are suppressed, and single-sample disturbances are rejected by the majority logic. START (falling sSDA while sSCL=1) and STOP (rising sSDA while sSCL=1) are detected from the filtered domain using previous samples (dSCL/dSDA); the bus busy flag sets on START and clears on STOP. During reads, data is captured from sSDA on the filtered rising edge of sSCL, ensuring deglitched sampling. Arbitration loss checks compare the intended SDA release against sSDA (filtered), preventing false decisions on noisy inputs. Clock stretching is honored by stalling the prescaler (slave_wait) when SCL is released but sSCL remains low, freezing filter updates and state progression until the line is truly high. Larger PRER values slow the filter cadence and increase spike rejection; smaller values increase responsiveness. Both async (nReset) and sync (rst) resets clear synchronizers and filter registers for a clean restart.

-- Open-Drain Output Control --

Implements true open■drain I2C line driving: the master never drives logic '1'; it either drives '0' or releases the line for external pull■ups.

- Signals
- sda_o, scl_o: hard■wired to 0 (data output permanently low).
- sda_oen, scl_oen: output■enable controls; 1 = release/high■Z (line goes high via pull■up), 0 = actively drive low.

- On reset or arbitration loss, both enables are set to 1 (lines released).

- Bit■level operation
- Idle: sda_oen = 1, scl_oen = 1.
- START: scl_oen = 1 (SCL high), then sda_oen = 0 (pull SDA low), then scl_oen = 0 (pull SCL low).
- STOP: with SCL low, sda_oen = 0; then scl_oen = 1 (SCL high); then sda_oen = 1 (release SDA while SCL high).
- WRITE bit: scl_oen = 0 (SCL low); sda_oen = din (0 → drive low, 1 → release); scl_oen = 1 (data valid with SCL high); hold sda_oen during SCL high; scl_oen = 0 to complete.
- READ bit: scl_oen = 0; sda_oen = 1 (slave drives); scl_oen = 1 and sample SDA on SCL rising/high phase; scl_oen = 0 to complete.
- ACK/NACK after read (9th bit): ack_in = 0 → sda_oen = 0 (ACK low); ack_in = 1 → sda_oen = 1 (NACK release).

- Multi■master and stretching
- Arbitration: during WRITE of a '1' (sda_oen = 1), if the bus reads low, declare arbitration loss and release both lines.
- Clock stretching: after releasing SCL (scl_oen → 1), if SCL remains low, pause timing until it goes high; never drive SCL high against a stretcher.

- Integration guidance
- Use I/O pads with separate data and enable (tri■state) to realize open■drain: tie pad data to 0 and drive pad enable with scl_oen/sda_oen.
- External pull■up resistors on SDA and SCL are required.
- Ensure pad■enable polarity matches 1 = release/high■Z.
- Typical mapping: scl_pad_o/sda_pad_o = 1'b0; scl_padoen_o/sda_padoen_o = scl_oen/sda_oen.
- CTR.core_en gates command acceptance only; when disabled, the bit core remains idle and releases both lines.

-- Bus Busy and Arbitration --

Bus Busy and Arbitration

- Bus Busy (SR[6] i2c_busy)
- Set on any detected START condition and cleared on any detected STOP, regardless of which master generated them.
- START/STOP are detected using debounced, majority-filtered SDA/SCL sampling while SCL is high, making detection robust to noise and clock stretching.
- May be high even when the local controller is idle (tip=0).
- Does not gate command writes: the host may issue CR.STA/WR/RD while i2c_busy=1 (e.g., to perform a repeated START). Arbitration may occur if another master is active.

- Arbitration Loss (SR[5] al)
- Detection conditions:
- Write-bit mismatch: during a WRITE bit with sda_chk active, if the master releases a logical ‘1’ (sda_oen=1) but the filtered bus reads ‘0’, arbitration is lost.
- External STOP: a STOP observed while an operation is active and not initiated by this controller.
- Immediate actions on arbitration loss:
- The bit-level core releases SDA and SCL and returns to idle.
- The byte controller aborts to ST_IDLE.
- The top auto-clears CR[7:4] (STA, STO, RD, WR), terminating the command; tip deasserts once CR clears.

- Latching and clear behavior:
- SR[5] al latches and remains set until a new START is issued (CR.STA=1) or reset.
- Equivalent logic: al_next = i2c_al | (al & ~sta); writing STA clears a previously latched al.
- Interrupt behavior:
- irq_flag is set on arbitration loss; wb_inta_o asserts if CTR.ien=1.
- Clear irq_flag by writing CR.IACK=1 (this does not clear SR[5] al).

- Operational guidance
- Software may read SR[6] to determine if the bus is currently in use by any master. The controller may still attempt a START; if contention occurs and another master wins, SR[5] al sets and the operation is aborted.
- After arbitration loss, retry by issuing a new START when appropriate (immediately for a repeated START or after the bus becomes free).

- Notes
- Multi-master safe: bus busy tracking is global to the bus, and arbitration is performed on write data bits and external STOP events. Filtering and synchronization ensure reliable detection under bus noise.

-- Interrupt Generation and Masking --

The core exposes a single, level-active interrupt output wb_inta_o driven by a sticky irq_flag latch. irq_flag is asserted on either a byte-operation completion (done/cmd_ack from the byte controller) or arbitration lost (i2c_al from the bit controller). No other status (e.g., rxack, tip, i2c_busy) generates interrupts unless accompanied by done or arbitration loss. Masking is controlled solely by CTR[6] ien: wb_inta_o = irq_flag & ien. ien gates only the external line; SR[0] always reflects irq_flag for polling even when interrupts are disabled. irq_flag remains set until cleared by writing CR[0] IACK=1 at address 0x4 (shared SR/CR). IACK self-clears on the following cycle. CR writes, including IACK, are accepted only when CTR[7] core_en=1; if core_en=0, software cannot clear irq_flag. Enabling ien while irq_flag is already set immediately asserts wb_inta_o. Multiple events coalesce; because wb_inta_o is level-sensitive, no events are lost if additional done or arbitration-loss events occur before IACK. Arbitration loss also latches SR[5] al; clearing irq_flag does not clear al. al clears on a new START (CR[7] STA) or reset. Command bits (CR[7:4] STA/STO/RD/WR) auto-clear on the same done/arbitration-loss event that sets irq_flag. irq_flag and wb_inta_o clear on reset. There is no per-source mask or separate interrupt status register; interrupt generation is aggregated into the single irq_flag. All logic is synchronous to the Wishbone clock.

OPERATION

-- Initialization and Configuration --

- Power-up/reset defaults
- PRER[7:0] at 0x0 = 0xFF; PRER[15:8] at 0x1 = 0xFF (prescaler = 0xFFFF)
- CTR at 0x2 = 0x00 (core disabled, interrupts disabled)
- TXR at 0x3 = 0x00
- SR/CR share 0x4: SR is read-only (reflects status); CR is write-only (command bits auto-clear)

- Required initialization sequence (host/Wishbone)

1) If in a multi-master system, wait until SR.i2c_busy == 0 (bus idle) before first START.
2) Program prescaler: write PRER low (0x0) then PRER high (0x1) to set SCL timing and input filter sample rate.
3) Enable the core: write CTR with core_en=1 (bit7). Optionally set ien=1 (bit6) to enable interrupts.
4) Clear any pending interrupt: write CR with IACK=1 (bit0). This has no side effects on other command bits because CR[2:0] self-clear each cycle.

- Configuration guidelines
- Prescaler (PRER[15:0]) drives the bit-level prescaler and the input filter sampling rate. Choose values to meet target SCL frequency and timing margins. Change PRER only when idle: SR.tip == 0 and SR.i2c_busy == 0.
- Interrupts: enable via CTR.ien (bit6). irq_flag (SR[0]) is sticky and asserts wb_inta_o when CTR.ien=1; clear by writing CR.IACK=1. Software may poll SR.irq_flag when interrupts are disabled.
- Command acceptance: writes to CR are honored only when CTR.core_en=1. CR[7:4] (STA, STO, RD, WR) auto-clear on byte completion or arbitration loss; CR[2:0] clear every cycle (IACK self-clears).
- Operational guards during setup: do not issue new CR commands while SR.tip == 1. Avoid changing CTR.core_en while a transfer is active (SR.tip == 1). If SR.al == 1, service/clear interrupt (CR.IACK=1 if used) and restart by issuing a new START when appropriate.

- Readback/verification during configuration
- Read SR at 0x4 to verify: SR.tip == 0 (no active byte), SR.i2c_busy == 0 (bus idle), SR.al == 0 (no arbitration loss), and irq_flag state as expected.
- Address 0x3 read returns RXR; write to 0x3 loads TXR (not typically used during initial configuration).

- Multi-master considerations at initialization
- At startup, wait for SR.i2c_busy == 0 before issuing START to avoid contention. The core autonomously tracks external START/STOP and clock stretching; no extra configuration is required beyond PRER and CTR.

-- Write Transaction Flow --

Host-visible write transaction flow using i2c_master_top:

- Initialize
- Program PRER[15:0] to set SCL frequency and input filtering.
- Set CTR.core_en=1 to accept CR commands; optionally set CTR.ien=1 to enable interrupts.

- Address phase (START + address write)
- Write TXR with 7-bit slave address in [7:1] and R/W=0 in [0].
- Write CR with STA=1 and WR=1 to issue START and transmit the address byte.
- While the byte is active: SR.tip=1; SR.i2c_busy reflects bus activity.
- On ACK cycle completion: SR.rxack updates (0=ACK, 1=NACK), SR.tip=0, CR[7:4] auto-clear, irq_flag is set (and wb_inta_o asserts if CTR.ien=1).

- Data phase (one or more data bytes)
- For each byte: write TXR with payload, then write CR with WR=1.
- For the final byte, optionally combine STOP by writing CR with WR=1 and STO=1; STOP is issued immediately after that byte's ACK cycle.
- Alternatively, issue STOP separately by writing CR.STO=1 after the last byte completes.

- Completion and status
- Pace the host by polling SR.tip (deasserts when the byte/STOP completes) or by servicing wb_inta_o;

issue the next WR only after SR.tip=0 or after handling the interrupt.
- After each byte, check SR.rxack; if 1 (NACK), terminate by issuing STOP (CR.STO=1) and evaluate retry policy.
- Clear the latched interrupt by writing CR.IACK=1 (self-clearing field).
- SR.i2c_busy stays 1 from START through STOP; repeated START keeps it asserted across segments.

- Error and arbitration handling
- If SR.al=1 (arbitration lost), the byte/command is aborted, CR[7:4] auto-clear, irq_flag is set. Wait until SR.i2c_busy=0, then reissue the transaction.
- Any unexpected external STOP while active or a reset forces the bit core idle and may set SR.al.

- Timing/behavioral guarantees
- Slave clock stretching is handled by the bit core; byte completion latency extends accordingly while SR.tip remains 1.
- START may be combined with WR; STOP may be combined with the final WR as described.

- Register semantics and acceptance rules
- CR writes are honored only when CTR.core_en=1.
- CR[7:4] command bits (STA, STO, RD/WR, ACK) auto-clear on byte completion or arbitration loss; CR[2:0] clear each cycle.
- Relevant SR fields for write: rxack, tip, al, i2c_busy, irq_flag; wb_inta_o = irq_flag AND CTR.ien.
- Data path: TXR sources transmitted bytes; RXR is not used in write transactions.

-- Read Transaction Flow --

- Setup: Program PRER[15:0] to the desired SCL rate and input filter cadence, then set CTR.core_en=1 (optionally CTR.ien=1 to enable interrupts). CR writes are accepted only when core_en=1.
- Address phase: Load TXR with the 7-bit slave address and set R/W=1 in bit0. Issue START and transmit the address by writing CR with STA=1 and WR=1. While SR.tip=1 the transfer is active. When it completes, check SR.rxack (0=ACK, 1=NACK). If rxack=1, abort the read and optionally issue STOP via CR.STO.
- Data phase (read one or more bytes): For each byte, write CR with RD=1 and set CR.ACK to the master response for the 9th bit (ACK=0 to request another byte, ACK=1 to NACK the last byte). On the final byte you may combine STOP by writing CR with RD=1, ACK=1, STO=1. Wait for SR.tip to clear, or if interrupts are enabled, service the interrupt (SR.irq_flag=1) then clear it by writing CR.IACK=1. After each RD completes, read the received byte from RXR (address 0x3) which holds the last captured byte.
- Completion and status: Command bits (STA/WR/RD/STO) auto-clear on byte done or arbitration loss. SR.tip reflects active RD/WR, SR.i2c_busy indicates bus busy, and SR.rxack applies to write phases (e.g., address byte). irq_flag is set on byte completion or arbitration loss and is level-latched until CR.IACK=1 is written.
- Errors and exceptions: If SR.al=1 (arbitration lost), the operation is aborted, active commands are cleared, irq_flag is set, and the bus is released. To retry, start a new transaction with CR.STA=1. Slave clock stretching is handled transparently by the bit engine; transfers resume when SCL is released high.
- Notes: STOP may be sent as a standalone CR.STO write or combined with the final RD (ACK=1, STO=1). Ensure PRER is programmed before enabling CTR.core_en.

-- Repeated START and STOP Handling --

Implements I2C START (S), STOP (P), and Repeated START (Sr) generation in master mode and

detection in slave mode, with correct bus-busy semantics and timing compliance. Definitions: START = SDA falling while SCL high; STOP = SDA rising while SCL high; Repeated START = START issued while bus_busy is already asserted (no intervening STOP). Master generation: accepts commands (START, STOP, REPEATED_START or START when bus_busy=1), gates generation to SCL-high, drives open-drain SDA transitions (high→low for S, low→high for P) and enforces tSU;STA, tHD;STA, tSU;STO via programmable timing counters; treats START when bus_busy=1 as Sr; ensures Sr can be issued immediately after ACK/NACK with SCL high; defers STOP until SCL high if requested mid-bit; collapses redundant commands (e.g., START→START becomes Sr). Slave detection: majority-filtered SDA/SCL sampling; detects START/STOP on SDA edges when SCL is high; flags Sr when START detected while bus_busy=1; on START or Sr, resets bit/byte counters, restarts address phase; on STOP, releases transaction context. Events and status: start_evt, rep_start_evt, stop_evt are single-cycle pulses; bus_busy asserts on first START and remains asserted across Sr; bus_busy deasserts only after STOP and tBUF (bus free time) elapses; provides in_start_gen/in_stop_gen activity flags. Sequencing and precedence: if REPEATED_START and STOP are requested together, Sr takes precedence until a legal STOP point (SCL high) is reached; STOP while idle is a no-op with status flag; START while idle is normal S; START while busy is Sr; back-to-back Sr allowed. Timing and compliance: parameters for tSU;STA, tHD;STA, tSU;STO, tBUF sized per mode (e.g., Standard/Fast/Fm+); SCL low/high period control ensures SDA transitions occur only when SCL is stably high; glitch filter/debounce on SDA/SCL avoids false edge detection. Multi-master: bus state is derived from observed lines; START/STOP detection always resynchronizes bit counters; during generation, arbitration is monitored at SDA while SCL high—on arbitration loss, abort START/STOP generation, release lines, set al_flag, keep bus_busy reflecting observed bus (e.g., remains busy if another master holds it). Interaction with byte/bit FSM: Sr may be requested at defined boundaries (after 9th clock ACK/NACK); STOP is scheduled at legal boundaries and may be deferred until SCL high; provides handshakes to stall/resume data FSM during S/P/Sr phases. Error and corner cases: Sr requested when lines not at required levels waits until SCL high then executes; Sr when bus idle is treated as S; STOP requested before any START is ignored with status; START/STOP pulses are suppressed if edges are not observed due to external bus forcing; all transitions honor open-drain behavior (never drive logic ‘1’; release to pull-up for high).

-- ACK/NACK Behavior --

On I2C, the 9th bit of each byte is the acknowledge: low (0) = ACK, high (1) = NACK. After the master reads 8 data bits, it must drive the 9th bit; CR[3] at 0x4 selects the level for that read-byte’s ACK phase (0 = master drives ACK to request more data, 1 = master drives NACK to end the read). After the master writes 8 bits (address or data), the slave owns the 9th bit and drives ACK/NACK; the byte controller releases SDA (READ-bit to the bit core) and samples the slave’s level into irxack, reported at SR[7] rxack (0 = ACK, 1 = NACK). Note SR[7] reflects only the slave’s response to the most recent write-type frame and does not report the master-driven ACK/NACK after reads. The ACK/NACK always occurs on the 9th SCL pulse; the bit core honors clock stretching and asserts cmd_ack when the bit completes. During a read-byte ACK phase, the byte controller issues a WRITE-bit to the bit core, which drives SDA according to CR[3] (din=0 forces SDA low=ACK; din=1 releases SDA high=NACK). CR[3] is latched for the duration of a RD command and auto-clears when the byte completes or on arbitration loss with other CR[7:4] bits; software must set CR[3] before each RD in multi-byte reads (ACK for all but the last byte, NACK for the final byte followed by STOP or repeated START). If arbitration is lost during the ACK/NACK phase, the operation aborts, command bits clear, SR.al is set, and SR.rxack may remain from the last completed write.

-- Command Handshake and Auto-Clear --

Address 0x4 is shared: reads return SR (status), writes target CR (command). CR writes are accepted only when CTR.core_en=1; otherwise they have no effect. Command fields are: CR[7]=STA, CR[6]=STO, CR[5]=RD, CR[4]=WR, CR[3]=ACK, CR[2:1]=0 (forced by hardware), CR[0]=IACK.

Handshake: Writing any of CR[7:4] asserts the corresponding request to the byte controller. These command bits remain asserted until the byte controller signals completion (done) after the 8 data bits plus the ACK/NACK phase (and an optional STOP), or until arbitration is lost (i2c_al). During an active RD or WR, SR.tip mirrors the transfer in progress: tip = RD | WR. Software must wait for tip=0 before issuing another RD/WR/STA/STO. Combined commands are supported: STA may be written with RD or WR to start immediately; STO may be combined to append a STOP after the 9th bit. Auto-clear: On done or i2c_al, hardware automatically clears CR[7:4] in the same cycle to release the request; tip deasserts accordingly. On arbitration loss, auto-clear releases the bus; SR.al latches and remains set until a new START (STA) or reset. Interrupts: An internal irq_flag is set on done or i2c_al and drives wb_inta_o when CTR.ien=1. Clear the interrupt by writing CR.IACK=1; IACK self-clears in the same cycle. ACK handling: CR[3] provides the master's ACK/NACK for the 9th bit of a read; it is not part of the [7:4] auto-clear and must be written by software before the read's ACK/NACK phase as needed. CR[2:1] are hard-wired to 0; CR[2:0] are transient (no state is retained beyond the write cycle).

-- Clock Stretching Behavior --

- Compliance: Fully supports I2C slave clock stretching and multi-master clock synchronization; the master advances only on actual filtered SCL transitions.
- Detection: When SCL is released high (scl_oen=1) but filtered/synchronized SCL (sSCL) remains low, a slave_wait condition is asserted.
- Bit-level behavior: While slave_wait is active, the prescaler counter is frozen, clk_en is held low, and the bit-level FSM does not advance. SCL/SDA outputs remain at their intended states. Data sampling (dout) occurs on the true rising edge of filtered SCL after the stretch. Applies to START, STOP, READ, WRITE, and the 9th ACK/NACK bit.
- Byte/top-level behavior: Byte operations stall transparently because cmd_ack/core_ack is delayed until the bit completes. SR.tip stays asserted throughout the stretch; completion (done) is deferred until SCL is released. SR.rxack updates only after byte completion. SR.al remains unaffected unless an actual arbitration loss occurs. Bus busy (SR.i2c_busy) reflects START/STOP detection and is unchanged by stretching.
- Interrupts and command bits: No interrupt is generated by stretching. irq_flag is set only on byte completion (done) or arbitration loss. CR command bits auto-clear only on completion or arbitration loss, not during a stretch.
- Timing/prescaler: PRER defines nominal SCL timing and the input filter cadence. Stretching can extend effective SCL periods; the controller handles this by freezing the prescaler during slave_wait. No hardware timeout is implemented; indefinite slave stretching will stall the master indefinitely.
- Multi-master: Clock synchronization is handled via sSCL gating. Arbitration loss is evaluated on SDA during write-high conflicts; stretching alone does not trigger SR.al.
- Host/bus interface: Wishbone register accesses (wb_ack_o) are independent of I2C timing and respond normally; only I2C operation completion is delayed by stretching.

-- Arbitration Loss and Recovery --

Arbitration Loss and Recovery

Detection (bit-level, i2c_master_bit_ctrl)
- During WRITE bits, if the master releases a logical '1' on SDA (sda_oen=1) but the filtered/synchronized SDA reads low (sSDA=0), arbitration is lost (al=1).
- If a STOP condition is observed on the bus while the controller is active and it is not the controller's own STOP request, assert arbitration lost (al=1).

Immediate reaction to loss
- Bit core (i2c_master_bit_ctrl): On al or rst, return to idle and release both lines (scl_oen=1, sda_oen=1); clear internal checks; cease further bit-level operations. Bus busy is maintained by

START/STOP detection on filtered lines and remains set until a STOP is observed.
- Byte core (i2c_master_byte_ctrl): On i2c_al or rst, abort to ST_IDLE, deassert commands, stop shifting; terminate the current byte sequence without completing the requested operation (no normal cmd_ack pulse).
- Top-level (i2c_master_top):
- SR[5] (al) is latched as: al <= i2c_al | (al & ~STA); it remains set until a new START (CR.STA) or reset.
- CR[7:4] (STA, STO, RD, WR) auto-clear on arbitration loss (and on normal byte done); SR[1] (tip) drops because RD/WR are cleared.
- irq_flag (SR[0]) is set by (done | i2c_al) and remains set until cleared by writing CR.IACK=1; wb_inta_o asserts if CTR.ien=1.
- No STOP is driven after loss; lines are released immediately by the bit core. Any pending STOP request is ineffective post-loss.
- SR[7] (rxack) mirrors the last irxack and may be stale if loss occurred mid-byte.

Host-visible recovery procedure
- Read SR to detect SR.al=1 and optionally acknowledge the interrupt by writing CR.IACK=1 (if enabled).
- Optionally wait for SR.i2c_busy=0 (STOP observed on the bus) before retrying to ensure the bus is free.
- Reprime TXR as needed, then issue a new START by writing CR.STA combined with CR.WR or CR.RD. Issuing START clears the SR.al latch.
- Ensure core_en is set (CTR[7]=1) so CR writes are accepted during recovery.

Scope and compliance notes
- Arbitration loss via data comparison is checked only during WRITE bits; unexpected external STOP triggers loss regardless of current command.
- Multi-master compliance: on loss, the controller relinquishes the bus immediately, reports the event (SR.al and interrupt), and requires the host to restart the transaction when the bus becomes available.

-- Interrupt Handling --

Interrupts are level-driven and sourced by two events: (1) byte-completion (cmd_ack from i2c_master_byte_ctrl at the end of a read/write byte or a STOP-only command) and (2) arbitration loss (i2c_al from i2c_master_bit_ctrl on SDA conflict or unexpected STOP).

The interrupt flag (SR[0] irq_flag) is a sticky latch set when (done | i2c_al) occurs and remains set until explicitly acknowledged. Reading SR does not clear irq_flag. The arbitration-lost status (SR[5] al) is a separate sticky latch set by i2c_al and is cleared only by issuing a new START (STA) or by reset; clearing irq_flag does not affect SR.al. SR[1] tip deasserts when a byte completes and typically coincides with an interrupt on completion.

The external interrupt output wb_inta_o is asserted when (irq_flag & CTR[6] ien). The irq_flag latches regardless of ien; enabling ien while irq_flag=1 immediately asserts wb_inta_o.

Clear/acknowledge is performed by writing CR[0] IACK=1 (register at address 0x4). IACK is a strobe and self-clears. CR writes, including IACK, are accepted only when CTR[7] core_en=1; if core_en=0 the irq_flag cannot be cleared. On byte completion or arbitration loss, CR[7:4] (STA, STO, RD, WR) auto-clear; CR[2:0] clear each cycle.

Reset behavior: irq_flag=0, wb_inta_o deasserted, CTR.ien=0 (masked), SR.al cleared.

Edge condition: irq_flag_next = (done | i2c_al | irq_flag_prev) & ~iack. If IACK and an event occur in the

same cycle, IACK suppresses setting that cycle. Software should acknowledge after servicing to avoid suppressing a masked event.

Typical ISR: read SR to determine cause (al vs. completion, ACK state, TIP), optionally read RX data, then write CR.IACK=1. If arbitration loss occurred, SR.al remains set until a new START is issued.

REGISTERS

-- Address Map --

Byte address offsets from the I2C master base; all registers are 8-bit. Reserved/unused bits must be written as 0 and read as 0. Unmapped addresses are reserved (read as 0, writes ignored). Reset: PRER=0xFFFF (0x00=0xFF, 0x01=0xFF), CTR=0x00, RXR/TXR=0x00, SR fields default to 0.
- 0x00 PRER[7:0] (rw): Prescaler low byte. Reset 0xFF.
- 0x01 PRER[15:8] (rw): Prescaler high byte. Reset 0xFF. PRER[15:0] sets SCL prescale and input filter sample period.
- 0x02 CTR (rw): Control. Reset 0x00. [7] core_en (enable core; gates CR writes), [6] ien (interrupt enable), [5:0] reserved(0).
- 0x03 RXR (ro) / TXR (wo): Receive/Transmit data. Reset 0x00. Read returns last received byte (RXR); write loads next transmit byte (TXR). Do not rely on reading back TXR contents.
- 0x04 SR (ro) / CR (wo): Status/Command.
SR (read): Reset 0x00; fields are dynamic. [7] rxack (0=ACK,1=NACK), [6] i2c_busy, [5] al (arbitration lost; latched until START or reset), [4:2]=0, [1] tip (transfer in progress), [0] irq_flag (latched; cleared by CR.IACK=1).
CR (write): Commands accepted only if CTR.core_en=1. [7] STA (START; may combine with RD/WR), [6] STO (STOP), [5] RD (read byte), [4] WR (write byte), [3] ACK (0=ACK after read,1=NACK), [2:1] forced 0, [0] IACK (write 1 to clear irq_flag). Auto-clear: CR[7:4] on byte done or arbitration lost; CR[2:0] each cycle (IACK self-clears).
- Interrupts: irq_flag sets on byte done or arbitration lost; clear via CR.IACK=1. External interrupt wb_inta_o = irq_flag & CTR.ien.
- Addressing: offsets are byte-based; integration may map these into a wider system address space.

-- Access Types --

- 0x00 PRER[7:0]: Read/Write. No read side-effects.
- 0x01 PRER[15:8]: Read/Write. No read side-effects.
- 0x02 CTR: Read/Write with bit-level access control. Writable: [7]=core_en, [6]=ien. Reserved [5:0] read as 0 and ignore writes.
- 0x03 TXR/RXR (split at same address): Write -> TXR (write-only transmit byte). Read -> RXR (read-only received byte). No read side-effects.
- 0x04 SR/CR (shared address with asymmetric access): Read -> SR (read-only status). Write -> CR (write-only command). CR writes are bus-acknowledged; they take functional effect only when CTR.core_en=1. CR command bits [7:4] self-clear on byte completion/arbitration loss; CR[2:0] clear each cycle; CR[0] (IACK) is write-1-to-clear.
- Status (SR) is read-only and latched; reads do not clear it. irq_flag clears only via CR.IACK; al remains set until a new START or reset.

-- Reset Values --

- On any reset (asynchronous nReset or synchronous rst), the core returns to idle, clears status/IRQs, and releases the I2C bus.
- Registers:
- PRER[15:0] = 0xFFFF (default prescaler)
- CTR = 0x00 (core_en=0, ien=0)
- TXR = 0x00
- RXR = 0x00
- SR (read-only, derived) = {rxack=0, tip=0, al=0, irq_flag=0, i2c_busy=0 unless the bus is externally active}
- CR (write-only) = all bits 0; command writes are ignored until CTR.core_en=1
- Outputs:
- wb_inta_o = 0 (either irq_flag=0 or ien=0)
- wb_ack_o = 0 (only asserts on valid Wishbone cycles)
- I2C pins: scl_o = 0, sda_o = 0; scl_oen = 1, sda_oen = 1 (bus released/high-Z)
- Submodule states:
- i2c_master_byte_ctrl: FSM=IDLE; cmd_ack=0; dout=0x00; ack_out=0; no pending start/stop/read/write
- i2c_master_bit_ctrl: FSM=idle; cmd_ack=0; busy=0; al=0; dout=0; scl_oen=1; sda_oen=1; prescaler loads from PRER
- Flags/behavior:
- irq_flag is cleared; CR.IACK clears it when written and self-clears each cycle
- al (arbitration lost) is cleared; remains cleared until a new START or another loss event
- SR fields reflect physical bus; i2c_busy may read as 1 immediately after reset if another master holds the bus

-- Register Descriptions --

Register map (addresses relative to module base)
0x0 PRER[7:0] (rw, reset 0xFF): Low byte of 16-bit prescaler that sets SCL timing and input filter sampling in the bit-level core.
0x1 PRER[15:8] (rw, reset 0xFF): High byte of prescaler. Combined PRER = {PRER[15:8], PRER[7:0]}.
0x2 CTR (rw, reset 0x00): Control register.
- [7] core_en: 1=enable I2C core; gates acceptance of CR writes. 0=CR writes ignored.
- [6] ien: 1=enable interrupt output (wb_inta_o) when irq_flag=1.
- [5:0] Reserved (read as 0; write 0).
0x3 TXR/RXR (shared data register, reset 0x00):
- Write (TXR, wo): next transmit byte to send.
- Read (RXR, ro): last received byte from bus (updated after RD completes).
0x4 SR/CR (shared address):
- SR (ro, reset 0x00): Status register bits.
- [7] rxack: last ACK sampled from slave after write (0=ACK, 1=NACK).
- [6] i2c_busy: 1=bus busy (START seen without STOP); 0=idle.
- [5] al: arbitration lost (latched). Cleared by reset or by issuing a new START.
- [4:2] 0.
- [1] tip: transfer in progress (RD or WR active).
- [0] irq_flag: latched interrupt flag.
- CR (wo): Command register (write accepted only if CTR.core_en=1). Bits are one-shot unless noted.
- [7] STA: issue START (may be combined with RD or WR in the same write).
- [6] STO: issue STOP.
- [5] RD: read one byte.
- [4] WR: write one byte (uses TXR).

- [3] ACK: master ACK value after a read (0=ACK, 1=NACK).
- [2:1] forced to 0; writes ignored.
- [0] IACK: write 1 to clear irq_flag.

Behavior
- Auto-clear: On byte completion or arbitration loss, CR[7:4] auto-clear. CR[2:0] self-clear each cycle (IACK self-clears on write).
- Interrupt: irq_flag (SR[0]) sets on byte complete or arbitration lost; cleared by writing CR.IACK=1. wb_inta_o asserts when irq_flag=1 and CTR.ien=1.
- TXR is consumed by the byte controller during WR; RXR is updated after RD completes and returned on reads of 0x3.
- SR signal mapping: rxack <= byte controller; i2c_busy <= bit controller; al reflects bit controller arbitration loss and is latched until a new START.
- CR writes are ignored if CTR.core_en=0; status remains readable.

-- Status Bits --

Status Register (SR @ 0x4, read-only). Bits: [7] rxack = last ACK from the slave (0=ACK, 1=NACK); updates on byte completion; meaningful after address/write phases; source: byte controller ack_out. [6] i2c_busy = bus busy indication (1 between a detected START and STOP on the filtered bus; multi-master aware); live bus state. [5] al = arbitration lost; latched; set by i2c_al; cleared by issuing a new START (CR.STA) or reset. [4:2] reserved = always 0. [1] tip = transfer in progress; 1 while CR.RD or CR.WR is active; auto-clears on byte completion or arbitration loss. [0] irq_flag = latched interrupt flag; set on byte completion or arbitration loss; cleared by writing CR.IACK=1; interrupt output wb_inta_o = irq_flag & CTR.ien. Note: SR is read via address 0x4; writing to 0x4 targets CR. Reset: al, tip, and irq_flag clear; i2c_busy reflects current bus state after reset.

-- Command Bits --

Command Register (CR) at 0x4 (write-only; writes take effect only when CTR.core_en=1). Reads at 0x4 return the Status Register (SR).
- [7] STA (START): Issue START. May be written together with RD or WR to begin a byte immediately. Auto-clears when the current byte completes or on arbitration loss (al).
- [6] STO (STOP): Issue STOP. When set with RD/WR, STOP is appended after the 9th (ACK/NACK) bit. Auto-clears on completion or al.
- [5] RD (Read byte): Master reads one byte (releases SDA for 8 bits, then drives the 9th bit per ACK). Asserts SR.tip while active. Auto-clears on completion or al.
- [4] WR (Write byte): Transmit one byte from TXR. Captures slave ACK to SR.rxack after the 9th bit. Asserts SR.tip while active. Auto-clears on completion or al.
- [3] ACK (ACK/NACK after read): 0=ACK (request more data), 1=NACK (final byte). Ignored for WR. Sampled for read operations and cleared each cycle.
- [2:1] Reserved: Hard-wired 0; writes have no effect.
- [0] IACK (Interrupt acknowledge): Write 1 to clear SR.irq_flag. Self-clears on the next cycle; no effect on bus activity or CR[7:4].

Behavior notes
- RD/WR may be combined with STA to start immediately; STO may be combined with RD/WR to append a STOP after the 9th bit.
- Command bits [7:4] auto-clear on byte-complete (byte controller cmd_ack) or al. ACK clears each cycle; IACK self-clears.
- SR.tip reflects an active RD/WR. SR.rxack reflects the slave ACK after WR. irq_flag is set on byte-complete or al and cleared by IACK; wb_inta_o = irq_flag & CTR.ien.

-- Side Effects and Write Restrictions --

Shared address 0x4: reads return SR (status, read■only); writes update CR (command, write■only). CR cannot be read back; attempts to write SR have no effect. CR write restrictions: writes are accepted only when CTR.core_en=1; if core_en=0, CR writes are ignored and no operation starts. CR[2:1] are forced to 0; writing 1 to these bits is ignored. CR[7:4] (STA, STO, RD, WR) start bus activity and auto■clear on byte completion or arbitration loss. CR[2:0] are cleared every cycle; CR[0] (IACK) self■clears immediately after writing 1 (writing 0 has no effect). Writing CR.ACK (bit3) does not start a transfer; it programs the master’s ACK/NACK for the 9th bit after a read—program ACK in the same write as RD when needed. Command combination rules: STA may be combined with RD/WR to start with a START; STO may be issued to terminate or appended after a byte; do not issue a new RD/WR while SR.tip=1. CTR writes: setting core_en=1 enables the core and permits CR writes; clearing core_en prevents further CR writes (ongoing subcore phases may complete or abort internally). CTR.ien only gates wb_inta_o; reserved CTR[5:0] are ignored and should be written as 0. PRER (0x0/0x1) side effects: prescaler and input filter update immediately; changing PRER while SR.tip=1 or SR.i2c_busy=1 alters ongoing SCL/SDA timing and sampling cadence. Data register 0x3: writes update TXR (no transfer start); reads return RXR (no side effects); RXR is not cleared by reads and TXR is not readable via 0x3. Interrupt/status side effects: writing CR.IACK=1 clears SR.irq_flag; wb_inta_o deasserts when irq_flag=0 or CTR.ien=0. SR.al (arbitration lost) is latched until a new START; issuing CR.STA clears al if no new arbitration loss occurs. SR.tip reflects active RD/WR and deasserts when CR[7:4] auto■clear. On arbitration loss or reset, subcores abort to idle, CR[7:4] clear, SR.al is set (latched), SR.irq_flag asserts, and bus drivers release (open■drain high). Issuing STO is internally captured to avoid false arbitration loss from external STOP observation.