Generated Specification Document

INTRODUCTION

-- Overview --

sdrc_core is a single-data-rate SDRAM controller that exposes a 32-bit application interface and drives a commodity SDR SDRAM with a configurable bus width (32/16/8). It accepts read/write requests, normalizes them to the SDR bus width, maps linear addresses to bank/row/column (via cfg_col_bits), and splits requests at page boundaries. Internally, per-bank scheduling and open-page control sequence PRE/ACT/READ/WRITE while enforcing device timing (tRP, tRCD, tRAS, tRC) and turnaround rules. An arbitration layer prioritizes initialization and periodic refresh over backend transfers, encodes active-low SDR commands (sdr_cmd={CSn,RASn,CASn,WEn}), and manages acceptance windows (rdok, wrok, act_ok, pre_ok) and CKE power-down when idle. The read path captures DQ on the SDRAM clock, transfers it into the core clock aligned to CAS latency, and only signals application-side read valid when a full 32-bit word is assembled. The write path slices 32-bit words into bus-width beats (LSB-first), applies byte enables via DQM, asserts output enable (sdr_den_n active-low) during valid beats, and only advances application data when a complete 32-bit word is consumed. The controller performs power-up init (PRECHARGE ALL, AUTO-REFRESH, LOAD MODE REGISTER) and periodic refresh, and requires configuration of sdr_width, cfg_col_bits, CAS latency, timing parameters, and mode register to match the SDRAM device. Integration notes: sdr_cmd is active-low; observe rdok/wrok/act_ok/pre_ok before issuing backend commands; respect read/write turnaround rules; and constrain the read CDC path between pad_clk and clk.

-- Key Features --

- 32-bit application to SDR SDRAM (8/16/32-bit) bridge with automatic bus-width conversion.
- Programmable address mapping (bank/row/column via cfg_col_bits); splits bursts at page boundaries; wrap-burst support.
- Width-normalized lengths/addresses for uniform downstream handling across sdr_width modes.
- Per-bank open-page scheduler with admission FIFO; preserves front-end ordering; head-of-line dispatch.
- Per-bank FSM enforces tRP/tRCD/tRAS, tracks open rows, sequences PRE/ACT/RD/WR; page-hit optimization.
- Transfer/timing engine prioritizes init/refresh over read/write; drives CMD/ADDR/CKE; generates DQM and DQ output-enable.
- CAS-latency–aligned read pipeline (rdstart/rdok/rdlast) and write strobes (wrstart/wrnext/wrlast) with turnaround control.
- Precharge permission gating per bank and global ACT spacing guard (tRC); precharge only when tRAS satisfied.
- Integrated management: power-up (PRECHARGE ALL, multiple AUTO-REFRESH, LMR), periodic refresh (configurable), refresh activity signaling; idle power-down via CKE.
- Clean handshakes and flow control: front-end req/ack, back-end x2b_ack; readiness windows x2b_rdok/wrok/act_ok; transaction ID/last/wrap propagation.
- Safe read-data capture across pad_clk to clk via two-stage sync aligned to CAS latency; suited for related clocks.

- Bus-width conversion details: write slicing with byte-enable mapping; read aggregation emits full 32-bit words; pass-through in 32-bit mode.
- Configurability: sdr_width, cfg_col_bits, CAS latency, tRP, tRC/tRCD, refresh period/max rows, mode register, enable.
- Reliability: FIFO under/overflow guards; one-bank-at-a-time issue; active-low sdr_cmd {CS,RAS,CAS,WE} per SDRAM truth table.

-- Block Diagram and Submodule Summary --

- Block Diagram (textual)
- Application-side (32-bit) interface
- Requests: addr/len/id/rd-wr/wrap enter sdrc_req_gen; acceptance via req_ack.
- Data: 32-bit write/read boundary through sdrc_bs_convert; byte enables active-low.
- Front-end request path
- sdrc_req_gen: normalizes widths to SDR beats, maps linear addr to bank/row/column, splits at page boundaries, emits r2b_* bursts.
- sdrc_bank_ctl: admits requests per bank, queues bank indices, arbitrates head-of-queue, multiplexes per-bank metadata/commands to back-end; implements four sdrc_bank_fsm instances.
- Command/transfer back-end
- sdrc_xfr_ctl: arbitrates management (init/refresh) vs bank commands, sequences PRE/ACT/READ/WRITE, enforces timing, aligns read data to CAS. Drives SDRAM pins: sdr_cmd{CSn,RASn,CASn,WEn}, sdr_addr, sdr_ba, sdr_cke, sdr_dqm, sdr_den_n, sdr_dout; captures sdr_din.
- Data width adapter and app handshakes
- sdrc_bs_convert: converts between 32-bit app data and 32/16/8-bit SDR bus beats; handles per-byte enables, serializes writes, assembles reads; coordinates via x2a_*/a2x_* strobes from sdrc_xfr_ctl.
- Read data capture
- pad_sdr_din captured on pad_clk, re-registered on clk, then delivered to sdrc_bs_convert; app_rd_valid asserted when a full 32-bit word assembled.
- Major handshakes and gates
- Front-end: req -> req_ack; r2b_req -> b2r_ack (per-bank/FIFO space).
- Bank/back-end: i2x_req (per-bank ready) -> b2x_req/b2x_* (selected bank) -> x2b_ack.
- Back-end acceptance: x2b_rdok, x2b_wrok, x2b_act_ok, x2b_pre_ok[3:0]; global tRAS gate via b2x_tras_ok.
- Data strobes: x2a_wrstart/wrnext/wrlast and x2a_rdstart/rdok/rdlast bound app_wr_next/app_last_wr and app_rd_valid/app_last_rd.
- Configuration/control
- Data width via sdr_width (32/16/8).
- Address mapping via cfg_col_bits (column width 8/9/10/11).
- Timing via cas_latency and tRP/tRCD/tRAS/tRC/tMRD; refresh via rfsh_time/rfsh_rmax; enable via sdram_enable.

- Submodule Summary
- sdrc_req_gen
- Role: Front-end request splitter and address mapper; normalizes to SDR bus width, maps linear addresses to bank/row/col, prevents page crossing by splitting into up to two sub-requests (r2b_last marks final).
- Key I/O
- Inputs: req, req_addr, req_len, req_id, req_wr_n, req_wrap, cfg_col_bits, sdr_width, b2r_arb_ok.
- Outputs: req_ack; r2b_req, r2b_write, r2b_req_id, r2b_ba, r2b_raddr, r2b_caddr, r2b_len, r2b_last, r2b_wrap.
- Handshake: r2b_req -> b2r_ack (per-bank admission).

- sdrc_bank_ctl
- Role: Per-bank admission, scheduling, and muxing; maintains per-bank queues, arbitrates the active bank, forwards commands/meta to the back-end; exposes global idle and tRAS gating.
- Key I/O
- Inputs: r2b_* from sdrc_req_gen; x2b_ack, x2b_rdok, x2b_wrok, x2b_act_ok, x2b_pre_ok[3:0] from sdrc_xfr_ctl.
- Outputs: b2r_ack to sdrc_req_gen; b2x_req, b2x_cmd, b2x_ba, b2x_raddr, b2x_caddr, b2x_len, b2x_write, b2x_wrap, b2x_last to sdrc_xfr_ctl; b2x_idle, b2x_tras_ok.
- Children: sdrc_bank_fsm[3:0] (one per SDRAM bank).
- sdrc_bank_fsm (instantiated per bank)
- Role: Open-page bank machine; tracks open row, sequences PRE/ACT/READ/WRITE, enforces tRP/tRCD/tRAS, optimizes page hits, handles burst close on last beat.
- Key I/O
- Inputs: r2b_* for this bank; x2b_ack, x2b_rdok, x2b_wrok, x2b_act_ok, x2b_pre_ok[bank]; refresh awareness from sdrc_xfr_ctl.
- Outputs: i2x_req (bank ready), b2x_cmd/meta/addr/len/write/wrap/last to sdrc_xfr_ctl.
- sdrc_xfr_ctl
- Role: Command/transfer engine and SDRAM pin driver; arbitrates management vs backend, sequences SDRAM commands, aligns read data to CAS latency, controls DQ enable and DQM; performs initialization and periodic refresh.
- Key I/O
- Inputs: b2x_* from sdrc_bank_ctl; configuration: cas_latency, timing registers (tRP/tRCD/tRAS/tRC/tMRD), rfsh_time, rfsh_rmax, sdram_enable; pad_sdr_din/pad_clk.
- Outputs: x2b_ack, x2b_rdok, x2b_wrok, x2b_act_ok, x2b_pre_ok[3:0]; x2a_wrstart/wrnext/wrlast, x2a_rdstart/rdok/rdlast; sdr_cmd, sdr_addr, sdr_ba, sdr_cke, sdr_dqm, sdr_den_n, sdr_dout.
- sdrc_bs_convert
- Role: Width adapter between 32-bit app and 32/16/8-bit SDR bus; slices/serializes 32-bit writes to bus beats; converts byte-enables to sdr_dqm; assembles returning narrow beats into 32-bit words and generates app_rd_valid.
- Key I/O
- Inputs: app_wr_data, app_wr_en_n, sdr_width; x2a_rdstart/rdok/rdlast and x2a_rddt from sdrc_xfr_ctl.
- Outputs: a2x_wrdt, a2x_wren_n to sdrc_xfr_ctl; app_wr_next, app_last_wr; app_rd_data, app_rd_valid, app_last_rd.
- Data ordering: 16-bit mode low half then high half; 8-bit mode bytes [7:0]→[15:8]→[23:16]→[31:24]; app_rd_data presents full word on final beat.

-- Standards and Protocol Compliance --

Implements the JEDEC single data rate (SDR) SDRAM standard for command set, signaling, addressing, timing, initialization, refresh, and power management. Commands are driven using active■low CSn/RASn/CASn/WEn per the JEDEC truth table, with BA/A multiplexed addressing (two BA bits) and A10 semantics for PRECHARGE ALL. Supported commands: ACTIVATE, READ, WRITE, PRECHARGE (bank/all), AUTO■REFRESH, LOAD MODE REGISTER, and BURST TERMINATE; auto■precharge is not used (explicit PRECHARGE only). Timing conformance is enforced via configurable parameters for tRP, tRCD, tRAS, tMRD, and tRC, plus activation spacing guards and CAS■latency■aligned read pipelines; read↔write turnarounds are controlled, and integrators must set device■specific constraints (e.g., tWR, tRRD, tRFC, tCCD, tWTR/tRTW) to match the target SDRAM. Initialization follows JEDEC: PRECHARGE ALL, a programmable number of AUTO■REFRESH cycles spaced by tRC, then LOAD MODE REGISTER with tMRD recovery; normal operation begins after an init■done indication. Refresh compliance is achieved via a programmable timer/counter (rfsh_time/rfsh_rmax) that schedules AUTO■REFRESH with required precharge and tRC spacing to meet row retention. Data bus compliance: x8/x16/x32 widths with DQM■based byte/halfword masking

per JEDEC; DQ drivers are enabled only during valid write beats, and read data is captured per the configured CAS latency. Burst behavior adheres to the mode register's burst length/type and CAS latency; BURST TERMINATE is issued when ending transfers early. CKE is used per JEDEC for power■down and active operation. Interoperability limits: designed for SDR SDRAM with two BA lines; not intended for DDR/DDR2+. Correct operation requires accurate mapping of active■low signals, A10 behavior, mode register image, and timing/CDC constraints at the SDRAM interface.

-- SDRAM Command Encoding (Active-Low CS/RAS/CAS/WE) --

sdr_cmd[3:0] is the SDRAM command bus with bit order {CSn, RASn, CASn, WEn}; all signals are active-low (logic 0 asserts). The discrete pins sdr_cs_n/sdr_ras_n/sdr_cas_n/sdr_we_n mirror this bus. Encodings used by the design:
- PRECHARGE: 0010; single-bank when sdr_addr[A10]=0.
- PRECHARGE ALL: 0010 with sdr_addr[A10]=1 (bank value ignored).
- ACTIVATE: 0011.
- READ: 0101.
- WRITE: 0100.
- AUTO-REFRESH: 0001.
- LOAD MODE REGISTER: 0000; mode/address presented on sdr_ba/sdr_addr.
- BURST TERMINATE: 0110.
Address/bank semantics: ACTIVATE uses sdr_ba as bank and sdr_addr as row; READ/WRITE use sdr_ba as bank and sdr_addr as column. Commands are encoded and arbitrated solely by sdrc_xfr_ctl: it maps b2x_cmd (00 PRE, 01 ACT, 10 READ, 11 WRITE) to sdr_cmd and injects management/initialization/refresh commands. The SDRAM samples sdr_addr/sdr_ba only when CSn is asserted (CSn=0). Ensure board pin mapping respects the {CSn, RASn, CASn, WEn} order.

-- Clocking and Reset Domains --

- Clocks: Two domains are used. clk is the core/application clock; all controller logic (request generation, bank control/FSM, transfer control, bus-width conversion, command arbitration, CAS-latency pipeline, init/refresh timers, and sdr_cke control) runs in clk. pad_clk is the SDRAM pad-side clock used to capture incoming DQ during reads.
- Reset: reset_n is a global, active-low reset distributed to all submodules. Treat reset_n as synchronous to clk unless otherwise indicated by RTL. Upon reset, all FSMs, counters, FIFOs, pipelines, and accumulators are cleared. sdrc_xfr_ctl additionally clears initialization/refresh machinery and holds sdr_init_done low until the sdram_enable-driven initialization sequence completes.
- Domain assignment: sdrc_req_gen, sdrc_bank_ctl, sdrc_bank_fsm, sdrc_xfr_ctl, and sdrc_bs_convert are clocked by clk and use reset_n. Per-bank timing gates (tRP/tRCD/tRAS), row tracking, ACT spacing (x2b_act_ok), turnaround gating (rd_pipe_mt), and precharge permission (b2x_tras_ok, cb_pre_ok) are generated and consumed within clk.
- Read CDC: The read data return path crosses from pad_clk to clk via a two-stage capture: pad_sdr_din1 is registered on pad_clk; pad_sdr_din2 is registered on clk. There is no handshake on this path; clk and pad_clk must be related (identical or phase-aligned/derived). Both capture stages are reset by reset_n. CAS-latency alignment and app-side validity strobes are generated after the crossing in clk.
- SDRAM outputs: SDRAM command/address/data outputs are generated by clk logic. If an external PHY or I/O wrapper re-registers these signals on pad_clk, ensure appropriate output timing; otherwise, constrain them directly from clk to the SDRAM.
- Constraints and usage: Declare pad_clk as related/derived from clk and constrain the pad_clk→clk read CDC; apply appropriate exceptions (e.g., multicycle or false-path) to match implementation. Do not use the two-flop CDC with fully asynchronous, unrelated clocks without adding synchronization. If pad_clk == clk, the design operates as single-clock with registered input capture. If pad_clk is a phase-related derivative, maintain the two-flop CDC and timing constraints.

- Reset effects (examples): sdrc_bs_convert clears wr_xfr_count/rd_xfr_count and saved_rd_data; sdrc_bank_fsm clears bank_valid and tRP/tRCD/tRAS timers and row tracking; sdrc_bank_ctl resets rank FIFO depth/contents and selection; sdrc_xfr_ctl clears the read pipeline, timers (tmr0, cntr1, rfsh_timer), and returns sdr_cke to the idle policy. CAS-latency pipeline flops are reset to a known state.
- Application interface: All app-side handshakes (req/ack, app_wr_next, app_rd_valid) are in clk; no CDC is required on the application interface.

-- Clock Domain Crossing Strategy and Assumptions --

Clock Domain Crossing Strategy and Assumptions
- Clock domains
- clk: Core/application domain for all controller FSMs and datapath (sdrc_req_gen, sdrc_bank_ctl/sdrc_bank_fsm, sdrc_xfr_ctl, sdrc_bs_convert).
- pad_clk: SDRAM pad/device domain used to capture incoming DQ.

- Read data CDC (pad_clk → clk)
- Two-stage related-clock transfer:
1) pad_sdr_din1 <= pad_sdr_din sampled on pad_clk (pad domain capture).
2) pad_sdr_din2 <= pad_sdr_din1 retimed on clk (core domain).
- Post-crossing read-valid/start/last strobes (e.g., x2a_rdok/x2a_rdstart/x2a_rdlast) are generated in clk, aligned via the CAS-latency pipeline after the retime.
- Assumption: clk and pad_clk are frequency/phase related (same source or known relationship). The multi-bit DQ bus relies on timing closure, not asynchronous synchronizers.

- Write/command path (clk → pad_clk)
- sdr_cmd[CSn,RASn,CASn,WEn], sdr_addr, sdr_ba, sdr_cke, sdr_dout, sdr_dqm, sdr_den_n are driven in clk and assumed to be delivered synchronously to the SDRAM/pad domain (pad_clk) via IO/PHY timing.
- No explicit CDC synchronizers are used; correctness depends on the clk↔pad_clk relationship and meeting SDRAM setup/hold at the device.

- No other CDCs
- All internal handshakes (r2b_*, b2x_*, x2b_*, x2a_*) remain in clk; no cross-domain synchronization required for these.

- Constraints and timing assumptions
- Constrain pad_clk→clk retiming path for DQ as related clocks (define clock relationships or apply appropriate multicycle/latency constraints consistent with CAS/data capture).
- Constrain clk→pad_clk launch for command/address/write data to meet SDRAM setup/hold at the IO boundary (account for clock forwarding/phase).
- Do not treat the multi-bit read bus as asynchronous; timing closure is required across the two-stage capture.

- Integration guidance for unrelated clocks
- If clk and pad_clk are not frequency/phase related, this core’s scheme is not sufficient: add a pad-side PHY for write/command and an asynchronous read data CDC (e.g., FIFO) external to the core. The current design does not include such asynchronous CDC structures.

- Reset considerations
- reset_n usage across domains is not defined here; ensure domain-appropriate, synchronously deasserted resets in each clock domain at the system level if a common reset spans clk and pad_clk.

- Read pipeline alignment
- CAS-latency alignment logic in clk assumes a deterministic clk↔pad_clk relationship so x2a_rdok aligns with valid pad_sdr_din2 data after the retiming stage.

-- Design Assumptions and Operating Conditions --

- Target device: JEDEC-compliant single data rate (SDR) SDRAM with 4 banks (BA[1:0]). Commands use active-low pins {CS_n, RAS_n, CAS_n, WE_n}. PRECHARGE ALL requires A10=1; per-bank precharge uses A10=0.
- Clocks: Two related domains are required: pad_clk (SDRAM I/O/PHY) and clk (core/app). They must be frequency-related (mesochronous/synchronous) and constrained for CDC. Read data crosses pad_clk->clk via two-stage capture; ensure STA exceptions or phase alignment as needed.
- Reset and enables: reset_n is active-low and must asynchronously/synchronously place the controller and SDRAM in a known state. sdram_enable must be asserted to run/maintain initialization; configuration changes require re-initialization.
- Data widths: Application interface is fixed 32-bit. SDRAM DQ bus width is configurable to 32/16/8 bits (sdr_width). Bus-width conversion serializes/deserializes across narrow beats; app must honor app_wr_next (advance only when high) and app_rd_valid (consume data only when high).
- Address mapping: Linear app addresses map to {ROW, BA, COL}; cfg_col_bits selects column width of 8/9/10/11 bits (page size 256/512/1024/2048 x8-beats). BA are the two bits above the column field; ROW is above BA. Requests are normalized to x8-beat units based on sdr_width.
- Request splitting: Transfers that would cross a page boundary are split into at most two sub-requests unless wrap=1. Splits respect remaining space in the current page.
- SDRAM timings: tRP, tRCD, tRAS, tRC (trcar_delay), and tMRD must be programmed per device datasheet at the intended frequency/voltage/temperature. x2b_act_ok enforces ACTIVATE spacing; precharge_ok requires cb_pre_ok & b2x_tras_ok to honor tRAS.
- CAS latency: cas_latency must match the SDRAM mode register; it sets read-return pipeline tap points. Misconfiguration will misalign read timing (rd_ok/rdstart/rdlast) and is not tolerated.
- Initialization: Requires PRECHARGE ALL, a programmed number of AUTO-REFRESH cycles spaced by tRC, then LOAD MODE REGISTER with sdram_mode_reg. sdr_init_done must assert before backend acceptance.
- Refresh: A free-running accumulator issues AUTO-REFRESH when rfsh_row_cnt reaches rfsh_rmax, with tRC spacing. rfsh_time and rfsh_rmax must meet the SDRAM tREF requirement over all rows and operating conditions.
- Arbitration/backpressure: Initialization/refresh have priority over backend reads/writes. Issue READ/WRITE only when x2b_rdok/x2b_wrok are asserted and x2b_act_ok permits. Admission respects per-bank queues (rank FIFO depth up to 4); b2r_arb_ok indicates capacity.
- Turnaround policy: Writes are blocked while the read-return pipeline is non-empty (rd_pipe_mt=0) to avoid bus turnaround hazards. Precharge permissions (x2b_pre_ok) are suppressed during active streaming.
- Burst behavior: Non-wrap transfers may be terminated using BURST TERMINATE at sub-burst boundaries; last/wrap metadata must be consistent with desired termination and page limits.
- Power-down: sdr_cke may be deasserted only when the controller is fully idle (no init/refresh/backend/read-return activity). The SDRAM must support CKE power-down; wake-up timing must be satisfied by configuration.
- Board-level mapping: External wiring must match SDRAM truth table for command bit order and active-low polarity. DQM width/lane mapping must correspond to sdr_width and PCB byte-lane wiring. Byte-enables/masks are active-low and mirror app_wr_en_n during writes.
- Configuration stability: Operating parameters (sdr_width, cfg_col_bits, cas_latency, timing delays, sdram_mode_reg) are expected stable during normal operation. Any change requires a full re-initialization sequence.

- Electrical/timing conditions: pad_clk-driven signals must meet SDRAM setup/hold at the target frequency. Constrain pad_clk↔clk CDC paths (read capture) appropriately. Ensure supply, temperature, and clock quality stay within the SDRAM datasheet operating ranges used to derive timing parameters.

IO PORTS

-- Port Summary --

- Clocks and Reset
- clk (input): Core/application clock.
- pad_clk (input): SDRAM pad/DQ capture clock.
- reset_n (input, active-low): Asynchronous reset.

- Application Request Interface
- app_req (input): New request strobe.
- app_req_addr[...] (input): Linear address (mapped to bank/row/column internally).
- app_req_len[...] (input): Burst length in 32-bit words (application side).
- app_req_wr_n (input): 0=write, 1=read.
- app_req_id[...] (input): Transaction ID tag.
- app_req_wrap (input): Allow wrap within page/burst.
- app_req_ack (output): Acceptance strobe for app_req.
- app_req_idle (output): High when controller can accept a new request.

- Application Write Data Interface
- app_wr_data[31:0] (input): Write data word.
- app_wr_en_n[3:0] (input, active-low per byte): Byte enables for app_wr_data.
- app_wr_next (output): Advance to next 32-bit word after all narrow beats for the current word are issued on SDR bus.
- app_last_wr (output): Last-beat indicator forwarded from SDR side.

- Application Read Data Interface
- app_rd_data[31:0] (output): Assembled 32-bit read data.
- app_rd_valid (output): Asserted when app_rd_data contains a full valid word.
- app_last_rd (output): Last-beat indicator forwarded from SDR side.
- app_rd_id[...] (output): Transaction ID associated with valid read data.

- SDRAM Control/Address Pins
- sdr_cmd[3:0] (output, active-low): Command bus {CS_n, RAS_n, CAS_n, WE_n}.
- sdr_addr[...] (output): SDRAM address bus; drives row on ACT, column on READ/WRITE; PRECHARGE uses row with A10 per target (bank/all).
- sdr_ba[1:0] (output): Bank address.
- sdr_cke (output): SDRAM clock enable.
- sdr_dqm[...] (output): Data mask/byte-lane enables (width depends on SDR data width).

- SDRAM Data Pins
- sdr_dout[...] (output): Write data to SDRAM (8/16/32 bits selected by sdr_width).

- sdr_den_n (output, active-low): DQ output enable (assert only during valid write beats).
- pad_sdr_din[...] (input): Read data from SDRAM pads (captured on pad_clk; width per sdr_width).

- Configuration and Timing
- sdr_width[1:0] (input): SDR data width select; 00=32-bit, 01=16-bit, others=8-bit.
- cfg_col_bits[1:0] (input): Column address width selector (8/9/10/11 bits).
- cas_latency[...] (input): CAS latency used to align read-valid strobes.
- sdram_mode_reg[...] (input): Mode register value for initialization.
- sdram_enable (input): Enables SDRAM, triggers power-up initialization.
- trp_delay[...] (input): PRECHARGE timing (tRP).
- trcar_delay[...] (input): tRC spacing parameter.
- rfsh_time[...] (input): Refresh interval.
- rfsh_rmax[...] (input): Maximum accumulated refresh count.

- Notes
- Widths marked [...] depend on device geometry or application protocol (e.g., sdr_addr, cas_latency, timing fields, app_req_id). The app side data width is fixed at 32 bits with 4 byte enables; SDR-side data and DQM widths depend on sdr_width. Read data crosses pad_clk→clk domains; constrain CDC accordingly.

-- Clock and Reset Signals --

Clocks: (1) clk — primary logic/application clock. All internal control, arbitration, bank/transfer FSMs, bus-width conversion, and inter-module handshakes (r2b_*, b2x_*, x2b_*, x2a_*) are synchronous to clk. (2) pad_clk — SDRAM pad/PHY clock. Used to time SDRAM command/address/data/mask outputs (sdr_cmd, sdr_addr, sdr_ba, sdr_dqm, sdr_den_n, sdr_dout) and to capture incoming read data at the pads.
Clock relationship: clk and pad_clk must be phase/frequency related. If clk == pad_clk, signals are naturally aligned. If different but related, define explicit launch/capture relationships in timing constraints (SDC/XDC). Read data crosses pad_clk→clk via a two-stage path: sdr_din → pad_sdr_din1 (pad_clk) → pad_sdr_din2 (clk). Treat this as a timed crossing between related clocks (no asynchronous synchronizer); constrain setup/hold or use multicycle as appropriate.
Reset: reset_n is active-low, asynchronously assertable, and must be synchronously deasserted in both clk and pad_clk domains to prevent partial-reset hazards. It resets all FSMs, counters, FIFOs, pipelines, and capture flops across domains, including: sdrc_req_gen (request-splitting FSM, address/length trackers), sdrc_bank_ctl/bank_fsm (per-bank state, timing trackers: tRP/tRCD/tRAS), sdrc_xfr_ctl (transfer/init FSMs, refresh counters, read-return pipeline; sdr_cke deasserted until init), sdrc_bs_convert (wr/rd transfer counters, saved read data), and pad-side registers (pad_sdr_din1 in pad_clk, pad_sdr_din2 in clk).
Constraints: Provide IO timing constraints on SDRAM pins relative to pad_clk and define the clk↔pad_clk relationship. Constrain the pad_clk→clk read-data capture path; apply CDC methodology consistent with related clocks.
Operational note: Hold application requests/handshakes until sdr_init_done is asserted after reset (initialization is triggered by sdram_enable).

-- Application Request Interface --

Overview
- The Application Request Interface is a 32-bit, synchronous front-end through which the application issues read and write bursts to SDR SDRAM. It exposes a control/request channel (address, length, attributes, handshake) and a 32-bit data channel with per-byte enables for writes and valid strobes for reads. All interface signals are synchronous to clk.

Control/request channel
- Signals:
- app_req: Assert to present a request; hold high with fields stable until acknowledged.
- app_req_ack: One-cycle acknowledge when the controller accepts the request.
- app_req_addr: Linear word address (unit = 32-bit word). No additional alignment required by the app.
- app_req_len: Burst length in 32-bit words.
- app_req_wr_n: Direction (0 = write, 1 = read).
- app_req_id: Application-supplied transaction ID (width user-defined); latched on acceptance and propagated to the data phase.
- app_req_wrap: Hint that the burst is wrapped; suppresses forced page split at a row/page boundary.
- app_req_idle: Optional status indicating the controller is idle/ready to accept a new request.
- Handshake/use:
- The controller asserts app_req_ack only when internal queues/timing allow; the app must hold app_req and all request fields stable until app_req_ack.
- Exactly one app_req_ack is returned per original request, even if the controller internally splits the burst at a page boundary.
- Addressing and burst/page behavior:
- Addresses and lengths are specified in 32-bit word units and are normalized internally to the SDR bus width (x32/x16/x8).
- At page (row/column) boundaries, the controller may split a request into at most two SDR-aligned sub-requests; admission order is preserved. app_req_wrap=1 hints that the app intends a wrapped burst and suppresses forced page splitting.
- Ordering and management:
- Accepted requests are preserved in FIFO order across banks; service may interleave by bank readiness and timing.
- During initialization/refresh or turnaround constraints, acceptance and command issuance may be throttled; backpressure is reflected via app_req_ack deassertion.

Write data channel (application side, 32-bit)
- Signals:
- app_wr_data[31:0]: Write data word.
- app_wr_en_n[3:0]: Active-low per-byte enables for app_wr_data.
- app_wr_next: Advance strobe; pulses when the controller has consumed the SDR beats corresponding to one 32-bit word. The app must only present the next word when app_wr_next is asserted.
- app_last_wr: Assert by the app to mark the last word of the current write burst; used to frame burst completion.
- Width conversion and ordering:
- sdr_width determines beats-per-word: x32=1 beat, x16=2 beats, x8=4 beats.
- Byte enables are narrowed per SDR beat; beat ordering is increasing significance (LSB to MSB).
- Usage:
- Hold app_wr_data/app_wr_en_n stable and advance only on app_wr_next.
- app_last_wr brackets burst completion and resets internal beat aggregation counters.

Read data channel (application side, 32-bit)
- Signals:
- app_rd_data[31:0]: Read data word.
- app_rd_valid: Pulses when a full 32-bit word has been assembled from SDR beats; sample app_rd_data only when asserted.
- app_last_rd: Asserted alongside the final word of the current read burst to bracket completion.
- Returned ID: The transaction ID associated with the request is forwarded in-phase with read

start/valid strobes to support matching (width matches app_req_id).
- Width conversion and timing:
- sdr_width governs aggregation: x32=1 beat, x16=2 beats, x8=4 beats.
- Read-valid timing is aligned to the configured CAS latency and the internal capture pipeline; latency is variable, but app_rd_valid provides precise data timing.

Configuration and timing
- sdr_width: Selects SDR data bus width (32/16/8) and governs beat aggregation/serialization and app_wr_next/app_rd_valid generation.
- cfg_col_bits: Determines page/column width and influences where bursts may split across page boundaries.
- cas_latency: Configures read-return alignment to the SDRAM; the app must tolerate variable latency and rely on app_rd_valid.
- Clocks and capture:
- All app-side signals are synchronous to clk.
- Read data are captured from the SDRAM pad clock domain via a two-stage transfer; treat as a CDC unless clocks are phase-related.

Usage rules
- Issue one request at a time; wait for app_req_ack before issuing the next. Hold request fields stable until ack.
- For writes, present data/byte enables and advance only on app_wr_next; use app_last_wr to mark burst end.
- For reads, consume data only when app_rd_valid; app_last_rd marks burst end.
- IDs should be unique across outstanding requests if the application relies on ID-based matching.

Behavioral guarantees
- Admission order is maintained across banks; internal arbitration may interleave service by bank readiness.
- The app sees a single app_req_ack per original request, irrespective of internal page splits.
- Backpressure can occur due to queue limits, initialization/refresh, or turnaround hazards; handshake/data strobes reflect readiness.

-- Application Write Data Interface --

Application Write Data Interface

Purpose
- Provides a 32-bit, word-oriented data path from the application to the SDRAM controller. Internally serialized to match the configured SDRAM data bus width (32/16/8 bits) while preserving per-byte write enables.

Signals
- app_wr_data[31:0] (input): 32-bit write data word from the application.
- app_wr_en_n[3:0] (input): Active-low per-byte write enables for app_wr_data bytes [7:0], [15:8], [23:16], [31:24].
- app_wr_next (output, pulse): Asserted for one cycle when the controller has consumed exactly one full 32-bit application word; advance the application write buffer only on this pulse.
- app_last_wr (output, pulse): Asserted on the last SDRAM data beat of the active write burst (end-of-burst indicator).
- sdr_width (configuration): Selects SDR data bus width: 00=32-bit, 01=16-bit, others=8-bit; determines sub-beat serialization described below.

Handshake and timing
- The application must hold app_wr_data and app_wr_en_n stable until app_wr_next pulses. There is no separate ready/valid on the app side.
- app_wr_next pulses once per 32-bit application word, coincident with the final sub-beat that completes that word on the SDR bus.
- Inter-beat spacing and sub-beat pacing are controlled by the controller and may include stalls; the application observes only app_wr_next for word-level flow control.

Width adaptation and ordering
- sdr_width=32-bit: Pass-through. One SDR beat per app word; app_wr_next pulses every beat.
- sdr_width=16-bit: Two sub-beats per app word.
- Sub-beat 0: app_wr_data[15:0] with enables app_wr_en_n[1:0].
- Sub-beat 1: app_wr_data[31:16] with enables app_wr_en_n[3:2].
- app_wr_next pulses on sub-beat 1.
- sdr_width=8-bit: Four sub-beats per app word.
- Sub-beats 0/1/2/3 carry bytes [7:0]/[15:8]/[23:16]/[31:24] respectively; app_wr_en_n bit for the active byte is applied.
- app_wr_next pulses on sub-beat 3.
- Narrow transfers emit lower-significance bytes/half-words first (LSB-first) toward the SDR bus.

Byte enables
- app_wr_en_n is active-low; a high value masks (disables) the corresponding byte on the SDR bus. Ensure polarity matches system convention.

Burst and boundary behavior
- app_last_wr pulses on the final SDRAM data beat of the current write burst (may be mid-word for narrow buses if a burst ends early due to scheduling or page boundary splits).
- Internal counters realign to sub-beat 0 at reset and on app_last_wr, ensuring the next app word starts at the first sub-beat.

Integration guidance
- Keep app_wr_data/app_wr_en_n stable until app_wr_next; update to the next word only when app_wr_next=1.
- For simpler buffering, size bursts to whole 32-bit words; partial-word endings are handled but may require careful application-side bookkeeping.
- Expect occasional stalls between sub-beats due to SDRAM timing or read/write turnarounds; no additional app-side ready is provided beyond app_wr_next.

-- Application Read Data Interface --

Provides a 32-bit application-side read data stream independent of the physical SDR SDRAM bus width (32/16/8). Interface signals: app_rd_data[31:0] carries the 32-bit read word; app_rd_valid pulses high only when app_rd_data holds a complete 32-bit word (sample on this pulse; no ready/backpressure is provided); app_last_rd pulses on the last 32-bit word of the active read burst (page-split bursts produce multiple app_last_rd pulses). A transaction ID sideband (x2a_id) is forwarded and remains constant for the duration of the burst to correlate responses with requests.

Width conversion: In 32-bit SDR mode, data and valid pass through one-for-one. In 16-bit SDR mode, two 16-bit beats are assembled per 32-bit word; app_rd_valid asserts on every second narrow beat; byte order is [15:0] then [31:16] (final beat forms the MS half-word). In 8-bit SDR mode, four 8-bit beats are assembled per 32-bit word; app_rd_valid asserts on every fourth narrow beat; byte order is [7:0] $\rightarrow$ [15:8] $\rightarrow$ [23:16] $\rightarrow$ [31:24] (final beat forms the MS byte). Data remains in-order across all modes.

Timing and latency: app_rd_valid is aligned to the SDRAM CAS latency via an internal pipeline in the transfer controller; configuration must match the SDRAM mode register to avoid misalignment. The narrow-beat valid drives the width assembly. Read/write turnaround is managed internally to preserve read timing and ordering.

Clocking and CDC: SDR pad data is captured on the pad clock, synchronized into the core/application clock, then width-converted before presenting app_rd_data/app_rd_valid. Treat this path as a CDC in timing constraints unless clocks are phase-related.

Reset: Internal assembly counters and pipelines reset on global reset and at burst boundaries; app_rd_valid is deasserted during reset. For 8/16-bit SDR widths, issue read lengths aligned to 32-bit words to avoid partial-word tails (assembly state resets at burst end).

-- SDRAM Device Interface (Command/Address/Data) --

The SDR SDRAM device is driven by an active-low command bus sdr_cmd[3:0]={CSn,RASn,CASn,WEn}, an address bus sdr_addr, bank selects sdr_ba[1:0], and a bidirectional DQ bus with DQM masking. Commands (active-low encodings) are: PRECHARGE=0010, ACTIVATE=0011, READ=0101, WRITE=0100, AUTO-REFRESH=0001, LOAD MODE REGISTER=0000, BURST TERMINATE=0110. sdrc_xfr_ctl issues commands with priority mgmt/init/refresh > backend transfers > internal burst continuation. Addressing: ACTIVATE drives the row on sdr_addr for the selected sdr_ba; READ/WRITE drive the column. For PRECHARGE, A10 on sdr_addr selects scope (0=bank-only, 1=all-banks). sdr_addr and sdr_ba are valid and latched while CSn=0 in the issuing cycle. Upstream sdrc_req_gen maps application addresses to {bank,row,column} based on cfg_col_bits and prevents page crossing. Data/DQM: The device DQ width is configurable (8/16/32 bits). sdrc_bs_convert adapts 32-bit application data to device beat order and byte-lane mapping. On writes, sdr_dout drives data; sdr_den_n (active-low) is asserted only on valid write beats to enable DQ outputs; sdr_dqm carries per-byte write masks (mirroring app-side enables) and is 0 during reads. On reads, DQ is tri-stated by the core; data is captured on the SDRAM clock and transferred to the core clock via a CAS-latency-aligned two-stage path, with rdstart/rdok/rdlast strobes generated by sdrc_xfr_ctl. Bursting: sdrc_xfr_ctl streams successive READ/WRITE column commands within a page and asserts BURST TERMINATE at the end of non-wrapping bursts.

-- Management and Configuration Inputs --

Management and configuration inputs define SDRAM geometry, timing, initialization, refresh policy, and bus-width adaptation. Inputs and behavior: sdram_enable — Master enable for the controller’s initialization/management FSM; assert to run the power-up sequence and keep asserted for normal operation; sdr_init_done is valid only while high. sdram_mode_reg — Mode Register value loaded during initialization (burst length, CAS latency, write-burst mode, etc.); must be coherent with cas_latency and the target SDRAM. cas_latency — CAS latency used to align the read-return pipeline; must match the CAS configured in sdram_mode_reg for correct data-valid strobes. sdr_width[1:0] — SDRAM data bus width and width-conversion selection: 00=32-bit, 01=16-bit, else=8-bit; drives byte/beat conversion and address/length normalization. cfg_col_bits[1:0] — SDRAM column address width: 00=8, 01=9, 10=10, 11=11; determines page size (256/512/1024/2048 beats) and geometry mapping. trp_delay — tRP in controller clock cycles; minimum PRECHARGE-to-ACTIVATE spacing enforced by bank and management FSMs. trcd_delay — tRCD in cycles; ACTIVATE-to-READ/WRITE spacing enforced by bank FSM. tras_delay — tRAS in cycles; minimum active-row time; precharge permission is gated by both streaming safety and tRAS satisfaction. trcar_delay — Row-cycle/refresh spacing in cycles; used to separate AUTO-REFRESH commands in management sequences. tmrd_delay — tMRD in cycles; wait time after LOAD MODE REGISTER before normal operation.

rfsh_time — Refresh accumulation period in cycles; rfsh_timer timeout interval used to accumulate refresh demand. rfsh_rmax — Refresh accumulator threshold; number of AUTO-REFRESH commands to issue when a refresh sequence starts. a2b_req_depth — Per-bank rank FIFO depth (1..4) for scheduling; sets front-end backpressure window.
Usage and constraints: All timing delays are specified in controller clock cycles and must satisfy the SDRAM datasheet minima at the operating frequency. sdr_width and cfg_col_bits are intended as static configuration; do not change while traffic is active. cas_latency must be consistent with sdram_mode_reg to avoid misaligned read-valid strobes. Refresh policy is defined by rfsh_time and rfsh_rmax: each rfsh_time timeout increments the refresh accumulator; when it reaches rfsh_rmax, that many AUTO-REFRESH commands are issued. Precharge commands are allowed only when both streaming safety and tRAS requirements are met. Provide clock/CDC constraints for pad_clk↔clk paths; the controller assumes related clocks for command/data timing. Initialization order: PRECHARGE ALL (A10=1) → N×AUTO-REFRESH (spaced by trcar_delay) → LOAD MODE REGISTER (wait tmrd_delay) → sdr_init_done.

-- Status and Handshake Signals --

Status and handshake signaling across sdrc_core and submodules:
- Application request (front-end)
- req: app raises a new request; req_ack: one-cycle accept pulse from sdrc_req_gen when b2r_arb_ok is high.
- req_idle: sdrc_req_gen idle (no pending segments); r2x_idle: ready to accept a new request (idle and not mid-split).
- r2b_req: sub-request strobe per page-aligned segment; r2b_last: marks final sub-request of the original app request.
- r2b_write, r2b_req_id, r2b_len, r2b_wrap: metadata latched and forwarded with r2b_req.
- Per-bank admission and selection (sdrc_bank_ctl)
- b2r_arb_ok: backpressure to req_gen (FIFO space available for new requests).
- b2r_ack: accept pulse back to req_gen when addressed bank captures r2b_*.
- r2i_req[3:0]: per-bank strobes to the addressed bank; x2i_ack[3:0]: ack routed only to the selected bank.
- i2x_req[3:0] -> b2x_req: bank "ready to issue" drives global command request; x2b_ack: command-path accept from sdrc_xfr_ctl.
- b2x_idle: rank FIFO empty; xfr_ok[3:0]: one-hot selected bank; b2x_tras_ok: global tRAS permission used with precharge gating.
- Per-bank command FSM (sdrc_bank_fsm)
- b2r_ack: capture of r2b_* metadata in IDLE; b2x_req/x2b_ack: PRE/ACT/RD/WR command request/accept.
- x2b_pre_ok, x2b_act_ok: permission gates for PRECHARGE/ACTIVATE; x2b_rdok, x2b_wrok: acceptance windows for READ/WRITE.
- x2b_refresh: indicates AUTO-REFRESH activity; bank FSM re-open sequencing on refresh.
- b2x_start/b2x_last/b2x_len/b2x_wrap/b2x_id: meta passed during IDLE and latched on accept for subsequent states.
- Transfer/controller path (sdrc_xfr_ctl)
- mgmt_req/mgmt_ack: management init/refresh arbitration (priority over backend); mgmt_idle: no pending mgmt; sdr_init_done: backend enable.
- x2b_ack: accept returned to the selected source; x2b_rdok/x2b_wrok: backend acceptance windows considering pipeline and mgmt.
- x2b_act_ok: ACTIVATE spacing; x2b_pre_ok[3:0]: per-bank precharge permission combined with b2x_tras_ok.
- x2b_refresh: pulses during AUTO-REFRESH; rd_pipe_mt: read pipeline empty flag, gates write acceptance for turnaround safety.
- x2a_wrstart/x2a_wrnext/x2a_wrlast: write beat strobes; x2a_rdstart/x2a_rdok/x2a_rdlast: read beat

strobes aligned to CAS latency.
- sdr_den_n: active-low DQ output enable; asserted low during valid write beats.
- Bus-width adapter (sdrc_bs_convert)
- app_wr_next: asserted on final narrow beat forming one 32-bit app word (gated by x2a_wrnext and width counters).
- app_rd_valid: asserted on final narrow beat assembling a 32-bit word (gated by x2a_rdok and width counters).
- app_last_wr/app_last_rd: forwarded from x2a_wrlast/x2a_rdlast; realign counters at burst boundaries.
- Ordering and usage
- Request path: app req -> req_ack (b2r_arb_ok) -> r2b_req -> b2r_ack -> b2x_req -> x2b_ack -> data beats via x2a_*.
- Read data valid: x2a_rdok per narrow beat; app_rd_valid on final beat per 32-bit word.
- Write accept: x2a_wrnext per narrow beat; app_wr_next on final beat per 32-bit word.
- Command gating: always honor x2b_rdok/x2b_wrok/x2b_act_ok/x2b_pre_ok and b2x_tras_ok; allow writes only when rd_pipe_mt is true.
- Active-level conventions: all req/ack/ok are active-high pulses unless suffixed *_n; sdr_den_n and app byte-enables are active-low.
- Idle detection: req_idle and b2x_idle indicate backend idle; mgmt_idle indicates no init/refresh activity.

-- Signal Polarity and Active Levels --

SDRAM command/control: sdr_cmd[3:0] = {CS, RAS, CAS, WE}, all bits active-low; discrete pins sdr_cs_n, sdr_ras_n, sdr_cas_n, sdr_we_n are active-low (low asserts). Addresses (sdr_addr, sdr_ba) latch when CS is low. PRECHARGE A10 semantics: A10=1 selects PRECHARGE ALL (active-high for ALL), A10=0 selects bank-only.
Clock enable: sdr_cke is active-high; held high during activity, may be driven low only in full idle for power-down.
Data I/O: sdr_den_n is active-low; low enables SDRAM DQ output drivers during write beats. sdr_dqm is an active-high mask (1 masks, 0 enables); mirrors a2x_wren_n which is an active-low byte-enable.
Application/backend handshakes and status: req, req_ack, r2b_req, b2r_ack, i2x_req, b2x_req, x2b_ack, x2b_rdok, x2b_wrok, x2b_pre_ok, x2b_act_ok, b2x_tras_ok are active-high. x2a_wrstart, x2a_wrnext, x2a_wrlast, x2a_rdstart, x2a_rdok, x2a_rdlast are active-high strobes. x2b_refresh pulses active-high during AUTO-REFRESH.
Write/read qualifiers and enables: req_wr_n is active-low (0=write, 1=read); internal r2b_write is active-high for writes. app_wr_en_n[3:0] are active-low per-byte enables; a2x_wren_n is an active-low narrowed enable. app_wr_next and app_rd_valid are active-high strobes.
Reset: reset_n inputs are active-low.

-- Interface Timing Requirements --

- Clocks and CDC
- All application/handshake signals are synchronous to clk.
- Read data from pads is captured on pad_clk and crosses into clk via a two-register path (pad_sdr_din -> pad_sdr_din1 on pad_clk; pad_sdr_din1 -> pad_sdr_din2 on clk). Treat this as a CDC unless clk and pad_clk are phase-related; constrain accordingly (e.g., set_clock_groups or modeled relationship).

- Acknowledgment pulse semantics
- req_ack, b2r_ack, and x2b_ack are single-cycle pulses synchronous to clk. Sources must hold associated address/metadata/control signals stable while the request is asserted and through the ack cycle.

- Application request interface (to sdrc_req_gen)
- Present new requests only when b2r_arb_ok=1 and r2x_idle=1.

- req, addr, len, id, rd/wr, wrap must remain stable while req=1 and until req_ack pulses.
- After req_ack, r2b_req asserts for the first sub-request and must remain asserted, with its associated controls stable, until b2r_ack. If a page split occurs, a second r2b_req follows after the first b2r_ack.

- Bank admission and backend handshakes (sdrc_bank_ctl)
- r2b_* controls presented to a selected bank must remain stable until that bank asserts b2r_ack.
- b2x_req asserts only when the head-of-queue bank i2x_req is high. b2x_cmd, b2x_addr, b2x_len, b2x_id, b2x_start, b2x_last, b2x_wrap must be held stable until x2b_ack.
- x2b_ack is the single-cycle accept; rank FIFO advances only on READ/WRITE accept (b2x_cmd[1]=1 and x2b_ack=1).

- Backend command acceptance (sdrc_xfr_ctl)
- Commands are accepted only when the corresponding permission is true in the same cycle: PRE when x2b_pre_ok & b2x_tras_ok, ACT when x2b_act_ok, READ when x2b_rdok, WRITE when x2b_wrok.
- x2b_ack indicates the command is driven on the SDRAM command bus that cycle; sources must hold b2x_* stable through this cycle.
- Back-to-back ACTIVATEs are forbidden; x2b_act_ok enforces required spacing (with trcd/trp timers).

- SDRAM pin timing (to device)
- sdr_cmd is active-low {CSn,RASn,CASn,WEn}; all command/address/bank signals change synchronously to pad_clk and must meet SDRAM AC timing.
- sdr_addr and sdr_ba are latched when CS is asserted; A10=1 for PRECHARGE ALL, A10=0 for per-bank precharge.
- sdr_cke remains high during active operation and may be lowered only when all paths are idle per controller policy.
- sdr_den_n drives low only on valid write beats; sdr_dout and sdr_dqm must meet setup/hold to pad_clk when writes are accepted.

- Read data timing and CAS latency
- x2a_rdok (read data valid to application/width-converter) asserts cas_latency cycles after the READ beat is scheduled (rd_next).
- x2a_rdstart and x2a_rdlast align to the same CAS pipeline; cas_latency must match the SDRAM mode register CAS.
- Writes are suppressed while the read-return pipeline is non-empty (rd_pipe_mt=0) to honor bus turnaround timing.

- Write data timing
- x2a_wrstart asserts on the first accepted write beat; x2a_wrnext asserts on each accepted beat; x2a_wrlast asserts on the final beat.
- During cycles with wr_next=1, sdr_dout and sdr_dqm must be stable; sdr_den_n must be low only in those cycles.

- Bus-width conversion timing (sdrc_bs_convert)
- 32-bit SDR bus: app_wr_next follows x2a_wrnext; app_rd_valid follows x2a_rdok directly.
- 16-bit SDR bus: hold app_wr_data/app_wr_en_n across two x2a_wrnext pulses; app_wr_next asserts on the 2nd pulse. app_rd_valid asserts on the 2nd x2a_rdok; sample app_rd_data only when app_rd_valid=1.
- 8-bit SDR bus: hold app_wr_data/app_wr_en_n across four x2a_wrnext pulses; app_wr_next asserts on the 4th pulse. app_rd_valid asserts on the 4th x2a_rdok.
- x2a_wrlast/x2a_rdlast mark burst boundaries; width-conversion counters reset on these pulses.

- Arbitration, refresh, and precharge timing effects
- While mgmt_req (init/refresh) is active, x2b_rdok/x2b_wrok deassert; expect stalls in backend acceptance.
- Precharge permission requires cb_pre_ok & b2x_tras_ok; precharge is blocked during active streaming until both allow.

- Address/burst boundary timing
- Next READ/WRITE is emitted at sub-burst boundaries when xfr_caddr_lsb[1:0]==0. Choose b2x_len and b2x_wrap consistent with the SDRAM burst length to avoid premature BURST TERMINATE.

- Programming-dependent timing and initialization
- trp_delay (tRP), trcd_delay (tRCD), tRAS enforcement, trcar_delay (tRC for auto-refresh spacing), and tMRD must be programmed to meet the SDRAM datasheet.
- No backend transfers are accepted until sdr_init_done=1.

ARCHITECTURE

-- Top-Level Architecture --

sdrc_core top-level architecture bridges a 32-bit application interface to an SDR SDRAM device with configurable data width (32/16/8). It is organized around five main blocks operating under a core/control clock (clk) and an SDRAM pad clock (pad_clk), with a dedicated read-data CDC path. External interfaces include: application request/control (req, req_addr, req_len, req_wr_n, req_id, req_wrap, req_ack, req_idle), write data (app_wr_data[31:0], app_wr_en_n[3:0], app_wr_next, app_last_wr), read data (app_rd_data[31:0], app_rd_valid, app_last_rd), and SDRAM pins (sdr_cmd[3:0] active-low, sdr_addr, sdr_ba[1:0], sdr_cke, sdr_dqm, sdr_dout, sdr_den_n active-low, sdr_din). Configuration inputs define data width, page geometry, CAS latency, timing (tRP/tRCD/tRAS), refresh cadence, and mode register programming.

Block roles and connectivity:
- sdrc_req_gen (front-end): accepts application requests, normalizes length to x8-beat units per sdr_width, maps linear address to {bank,row,column} using cfg_col_bits, and splits non-wrapping bursts at page boundaries. It exposes per-bank sub-requests (r2b_*) and handshakes admission with sdrc_bank_ctl via b2r_arb_ok/b2r_ack.
- sdrc_bank_ctl + per-bank sdrc_bank_fsm: bank_ctl queues head-of-line sub-requests per bank and selects one bank at a time, multiplexing command/address/meta toward the back-end (b2x_*), and routing acknowledgments (x2b_ack) back to the active bank. Each bank_fsm enforces tRP/tRCD/tRAS, tracks open-row state (page hit/miss), and sequences PRE/ACT/READ/WRITE for its bank, generating i2x_req and related metadata. Timing guards x2b_act_ok/x2b_pre_ok and b2x_tras_ok gate activates and precharges.
- sdrc_xfr_ctl (SDRAM command/transfer + management): arbitrates between backend transfers and management (initialization/refresh); encodes/outputs SDRAM commands and pins (CS/RAS/CAS/WE, address, bank, CKE, DQM, DQ enable/data). It streams WRITE bursts, aligns READ return to cas_latency, and provides acceptance windows x2b_rdok/x2b_wrok to the bank side. It performs power-up sequencing (PREALL, AUTO-REFRESH cycles, LOAD MODE) and periodic refresh, asserting sdr_init_done when ready, and deasserts sdr_cke in deep idle.
- sdrc_bs_convert (bus-width adapter): serializes 32-bit application writes into 1/2/4 SDR beats (for

32/16/8-bit widths) with narrowed byte enables, and assembles 1/2/4 SDR beats into 32-bit application reads. It paces the application via app_wr_next and app_rd_valid only when full words are consumed/produced, and forwards last-beat indicators.
- Read-data CDC path: SDRAM DQ is captured on pad_clk into a first stage (pad_sdr_din1) and transferred into the clk domain (pad_sdr_din2). sdrc_xfr_ctl schedules READ beats so that x2a_rdok/start/last align to cas_latency taps; writes are blocked while the read-return pipeline is non-empty to avoid turnaround hazards.

Data/control flow: application requests enter sdrc_req_gen, are bank-queued in sdrc_bank_ctl, timed and sequenced by the selected sdrc_bank_fsm, and executed by sdrc_xfr_ctl which drives SDRAM pins. Write data flows from the application through sdrc_bs_convert to xfr_ctl and out to SDRAM; read data returns from SDRAM through the CDC path to xfr_ctl and then through sdrc_bs_convert to the application. Handshakes and guards (b2r_arb_ok/b2r_ack, x2b_ack, x2b_rdok/wrok, x2b_act_ok/x2b_pre_ok, b2x_tras_ok) ensure legal timing, bank policy, and bus turnaround behavior.

Clocking and conventions: clk is the control/application domain; pad_clk captures SDRAM read data. The two clocks are related but require explicit CDC/timing constraints on the read path. Active-low signals include command bits (sdr_cmd), DQ output enable (sdr_den_n), and byte-enable strobes; DQM polarity must match device truth tables.

Configurability and scaling: page geometry via cfg_col_bits, data width via sdr_width, CAS latency and timing delays, refresh parameters, and mode register contents are parameterized. The design assumes four banks; per-bank FIFO depth in bank_ctl may be configurable. Power-down behavior is managed centrally by sdrc_xfr_ctl via sdr_cke when backend, management, and read pipelines are idle.

-- Request Generation and Address Mapping (sdrc_req_gen) --

Request Generation and Address Mapping (sdrc_req_gen)

Purpose
- Front-end request splitter and address mapper for SDRAM. Converts a generic application request into one or two page-aligned sub-requests, mapped to ROW/BANK/COL, in width-agnostic (x8-beat) units.

Interfaces (conceptual)
- Inputs: clk, reset_n; req (valid), req_addr, req_len, req_wr_n (0=write), req_id, req_wrap; sdr_width; cfg_col_bits; b2r_arb_ok; b2r_ack.
- Outputs: req_ack (accept), req_idle, r2x_idle; r2b_req, r2b_write, r2b_req_id, r2b_ba[1:0], r2b_raddr, r2b_caddr, r2b_len, r2b_last, r2b_wrap.

Width normalization (x8-beat baseline)
- Converts incoming length/address to x8-beat units by shifting based on SDR bus width:
- x8: shift-by-0; x16: shift-by-1; x32: shift-by-2.
- Shifts: length is scaled (len << shift); address LSBs are padded with zeros to preserve alignment.
- Assumes downstream (bank layer) operates in the same x8-beat units.

Address mapping (cfg_col_bits)
- Uses normalized address to derive COL/BA/ROW:
- cfg_col_bits: 00→8 COL bits, 01→9, 10→10, 11→11 (page sizes 256/512/1024/2048 x8-beats).
- BA: the two bits immediately above COL.
- ROW: all bits above BA.
- Exposes: r2b_caddr (zero-extended COL to physical bus width), r2b_ba[1:0], r2b_raddr.

Page-boundary split
- Computes page_size = 2^(COL_BITS) and col_offset from current COL.
- max_len = page_size − col_offset.
- If req_wrap=0 and req_len > max_len, splits into at most two sub-requests:
1) First: length=max_len to end-of-page.
2) Second: remaining length starting at next page.
- If req_wrap=1, no forced split; wrap attribute forwarded (r2b_wrap).
- r2b_len never crosses a page boundary. r2b_last=1 only on the final sub-request for the original request.

Handshakes and sequencing
- Front-end accept: accepts a new request only when b2r_arb_ok=1; asserts req_ack when the request is latched and preprocessed.
- Issue/ack: asserts r2b_req for the current segment; waits for b2r_ack from bank layer to confirm capture.
- Sequencing: Idle → First segment → (optional) Second segment → Idle. If a second segment is required, issues it after first ack.
- Idle indication: req_idle=1 only in Idle; r2x_idle mirrors readiness to accept a new request.

Forwarded attributes
- r2b_write = ~req_wr_n; r2b_req_id = req_id; r2b_wrap = req_wrap.
- r2b_len is in x8-beat units; constrained to current page.
- At most one split per app request (maximum two sub-requests).

Reset/initialization
- On reset_n=0: internal state clears to Idle; req_idle=1; r2b_req=0; no partial request is retained.

Assumptions/requirements
- sdr_width must be decoded consistently system-wide. This block assumes shifts of 0/1/2 for x8/x16/x32; if a different encoding is used elsewhere (e.g., 2'b00=32-bit), a decode must translate to the correct shift amount.
- cfg_col_bits must match the physical SDRAM geometry; incorrect values corrupt BA/ROW/COL decoding and page calculations.
- req_addr/req_len definitions at the application interface must be compatible with the width normalization step so that post-shift units represent x8-beats.
- Extremely long bursts spanning multiple pages are handled as a single app request with at most one internal split here; further spanning requires additional app-level requests or reissue by upstream logic.

-- Per-Bank Queueing and Scheduler (sdrc_bank_ctl) --

sdrc_bank_ctl provides per-bank admission, ordering, and single-issue scheduling for four SDRAM banks. It decouples upstream request acceptance (r2b_*) from per-bank timing (in sdrc_bank_fsm) and from the backend transfer path (sdrc_xfr_ctl). Admission/backpressure: Accepts r2b_* requests only when a small rank FIFO has space; exposes b2r_arb_ok to throttle the upstream generator (sdrc_req_gen). Per-bank dispatch: Decodes the target bank BA[1:0] and presents the request to only that bank via r2i_req[3:0]; aggregates per-bank accepts i2r_ack[3:0] back to b2r_ack (OR). Queueing: On any i2r_ack asserted, pushes the accepted bank ID into a rank FIFO (depth parameter a2b_req_depth, up to 4). The FIFO stores only 2-bit BA entries, preserves arrival order across banks, supports simultaneous push/pop, and tracks occupancy via rank_cnt; internal write-slot select (rank_wr_sel) derives from current depth. Flow control: Deasserts b2r_arb_ok when FIFO is full; guards against overflow/underflow (implementation assertions). Scheduling: Strict head-of-line policy—only the head bank in the FIFO (xfr_ba) may issue; if its per-bank FSM is not ready (i2x_req[head]=0), no other

bank is considered (no look-ahead). This maintains fairness by admission order and may introduce head-of-line blocking by design. Backend command/data muxing: When the head bank asserts i2x_req, sdrc_bank_ctl drives b2x_req along with b2x_ba=xfr_ba, b2x_cmd/b2x_addr and meta (id/start/last/len/wrap) multiplexed from that bank's i2x_* signals. Backend acknowledgment x2b_ack is routed only to the selected bank via one-hot x2i_ack[head]. Completion/pop: The FIFO entry is popped only when the first data-transfer command (READ/WRITE; b2x_cmd[1]=1) for the head bank is accepted by the backend (x2b_ack). PRECHARGE/ACTIVATE acknowledgments do not pop; the head remains until transfer begins. Status: b2x_idle asserts when the FIFO is empty; xfr_ok is a one-hot decode of the selected bank used to gate issue paths. Timing and gating: Per-bank SDRAM timing (tRP/tRCD/tRAS) and row state are enforced in sdrc_bank_fsm; sdrc_bank_ctl neither modifies meta nor timing. It surfaces an all-banks tRAS-OK indicator b2x_tras_ok (AND of per-bank flags) to the backend; x2b_pre_ok/x2b_act_ok/x2b_rdok/x2b_wrok are consumed within the bank FSMs. Banks are fixed at four (BA is 2 bits); ensure upstream address mapping matches. One backend command may be issued per cycle; routing is precise: only the selected bank sees backend ack, and all b2x_* meta follow the head bank selection.

OPERATION

REGISTERS

CONFIGURATION