Generated Specification Document

INTRODUCTION

-- Overview --

spiMaster is a memory-mapped SPI/SD master core that bridges a software bus to SPI wire-level transfers and SD-card operations. It supports three transaction classes: single-byte SPI transfers, SD-card initialization over SPI (legacy CMD0/CMD1), and SD single-block read/write (CMD17/CMD24). The core presents a Wishbone-like bus front-end, runs across two clocks (busClk and spiSysClk), and drives external SPI pins (MOSI, MISO, SCK, active-low CS). Architecture layers include a bus interface decoder, a control/status register block with safe clock-domain crossing, a dispatcher FSM in the SPI domain, specialized SD init and block R/W controllers, a command framer (sendCmd), a single-byte TX/RX staging unit, a wire-level SPI shifter with programmable half-period, and dual-clock FIFOs for streaming data across domains. Chip select is arbitrated by active-low AND across submodules, allowing any active transaction to assert CS. Software interacts via an address map: 0x00.. control/status, 0x10.. RX FIFO, 0x20.. TX FIFO; most accesses ack immediately, with a defined 1-cycle wait-state on FIFO data reads at the 0x10/0x20 base. Typical use: program type and parameters (including SPI clock delay), optionally write a direct-access TX byte, issue start, poll status, and read results or stream bytes via FIFOs. Key properties include clean layering, CDC coverage via synchronizers and dual-clock FIFOs, continuous streaming capability, and programmable SPI speed. Notable assumptions/limits: SD init omits CMD8/ACMD41, data CRC is fixed/ignored, SPI timing targets CPOL=0 devices, and the provided FIFO CDC uses simple binary pointer synchronization rather than Gray coding.

-- Key Features --

- Memory-mapped SPI/SD master with clean sublayers: bus/registers, control FSMs, command sender, byte staging, wire-level shifter, and dual-clock FIFOs.
- Wishbone-style bus front-end with 3 address windows (0x00 control/status, 0x10 RX FIFO, 0x20 TX FIFO); immediate ACK on most accesses; single wait-state inserted on FIFO base reads for timing alignment.
- Control/Status block featuring robust CDC (stretched start/reset), selectable transaction types (raw SPI byte, SD init, SD single-block read/write), direct single-byte TX/RX path, 32-bit SD address registers, programmable SPI half-period (clkDelay), module ID and detailed status/error readback.
- Central transaction dispatcher providing one-cycle request pulses, busy/idle reporting, and deterministic chip-select control for raw SPI transfers.
- Top-level chip-select arbitration: active-low CS derived by AND of submodule requests; any active submodule can assert CS low.
- SD-over-SPI support: initialization sequence with slow SPI clock, preamble clocks, CMD0/CMD1 retries/timeouts; single-block read/write via CMD17/CMD24 with token handling, 512-byte streaming through FIFOs, dummy 0xFF clocking, and comprehensive timeout/error reporting.
- Shared command-framing engine: frames preamble + command + 4 data bytes + checksum, polls for R1 response until ready or retry limit, exposes response and timeout flags, arbitrates two request sources.
- Byte staging layer: single-byte TX register with prioritized writers and full flag; single-byte RX register

with ready flag and multiple independent clear sources.
- Wire-level SPI shifter: MSB-first 8-bit transfers, SCK idle low (CPOL=0 behavior), programmable half-period via clkDelay, back-to-back streaming when TX full is maintained, per-byte RX ready pulses, and TX idle indication.
- Dual-clock FIFOs with bus-side interfaces: TX FIFO (bus writes, SPI reads) and RX FIFO (SPI writes, bus reads) with one-shot force-empty flush via toggle-based CDC; exposure of full/empty/count; RX data port with 1-cycle latency and mapped count/flush registers.
- Robust control/data clock-domain crossings: stretched control pulses, three-stage synchronizers, and toggle-based one-shot flushes for FIFOs.
- Programmable SPI speed: global clkDelay register; SD init temporarily overrides to slow preset and restores normal speed after bring-up.
- Sustained throughput: shifter supports gapless back-to-back byte streaming when upstream maintains TX full.
- Software-friendly operation: zero-wait control accesses, explicit wait-state handling for FIFO reads, simple poll/ack transaction flow with clear busy and error reporting.

-- Design Goals --

- Provide a software-friendly, memory-mapped SPI/SD master with a simple bus register map and minimal wait-states (immediate ACK everywhere except a single-cycle wait on RX/TX FIFO base reads).
- Bridge busClk and spiSysClk safely using stretched start/reset pulses and dual-clock FIFOs with coherent force-empty flush; favor robust CDC for predictable behavior.
- Support three transaction classes: raw single-byte SPI, SD-card initialization, and SD single-block read/write (512 bytes).
- Use a layered, modular architecture (bus front-end, dispatcher, command sender, byte staging, wire shifter, async FIFOs) with clear responsibilities and simple handshakes.
- Ensure deterministic chip-select arbitration: CS high when idle; any active submodule can assert CS low; hold CS low across block transfers.
- Provide a programmable SPI clock (clkDelay) and automatically slow the SPI rate during SD initialization to meet timing.
- Enable sustained streaming: back-to-back byte transfers at the wire engine with minimal idle gaps when txDataFull persists; pace 512-byte moves via FIFO full/empty.
- Expose clear, software-visible transaction status and concise error reporting for init, read, and write; include sendCmd response timeout indication.
- Favor portability and resource efficiency: synchronous resets, simple counters, generic dual-port RAM inference; keep CDC/simple pointer sync practical with a path to Gray-coded upgrades.
- Maintain predictable SPI behavior (Mode 0, MSB-first) with timing that meets common devices.
- Provide clean soft reset across both domains, clearing key state; require software to reprogram SD address post-reset.
- Design for extensibility: spiTransType and dispatcher framework allow new transaction types without changing the bus map or disturbing existing layers.

-- Typical Use Cases and Flows --

Typical Use Cases and Flows for spiMaster

- Direct single-byte SPI exchange (type 00)
- Software: optional reset (0x01 bit0), set clock (0x0B); write TX byte (0x06); set type=0 (0x02), start (0x03 bit0); poll busy (0x04); read RX (0x06); check errors (0x05, none expected).
- Hardware: CS low, send 8 bits at rate from clkDelay, sample MISO, mark RX ready, CS high.

- SD Card Initialization (type 01)

- Software: optional reset; set normal clkDelay (0x0B); set type=1 (0x02), start; poll busy (0x04); read init error at 0x05.
- Hardware: force slow clock; preclock ~160×0xFF with CS high; assert CS; issue CMD0 (expect R1=0x01) then CMD1 (expect R1=0x00) until success or timeout; restore clock; set error bits on failure.

- SD Single-Block Read (type 10)
- Software: ensure initialized; write block address bytes 0x07..0x0A; set type=2; start; poll busy; if readError=0 (0x05), read RX FIFO at 0x10 while count (0x12/0x13) > 0; note 1-cycle wait on 0x10 base reads; optional flush RX (0x14 bit0).
- Hardware: CS low; CMD17 with address; wait for token 0xFE; stream 512 bytes into RX FIFO; send trailing CRC bytes (ignored); CS high; set readError on token/command timeouts.

- SD Single-Block Write (type 11)
- Software: ensure initialized; fill TX FIFO by writing to 0x20; optional flush TX (0x24 bit0); set type=3; start; poll busy; read writeError (0x05).
- Hardware: CS low; CMD24 with address; gap 0xFF, send token 0xFE; stream 512 bytes from TX FIFO; send 2×CRC 0xFF; expect data-response 0x05; poll busy release by sending 0xFF until MISO nonzero; CS high; set writeError on response/timeout/busy failures.

- Continuous streaming
- SPI engine supports back-to-back bytes without gaps if txDataFull is asserted in time; used by SD block transfers to keep SCK continuous while CS low.

- Bus protocol and timing
- Address windows: 0x00.. control/status, 0x10.. RX FIFO, 0x20.. TX FIFO.
- Acknowledge: zero-wait for most ops; base reads of 0x10 and 0x20 insert a 1-cycle wait (hold strobe across the wait).
- RX FIFO: read at 0x10 returns data and pops; occupancy at 0x12/0x13; force-empty at 0x14 bit0.
- TX FIFO: write at 0x20 enqueues; force-empty at 0x24 bit0.

- Chip-select behavior
- spiCS_n = spiCS_nFromInitSD & spiCS_nFromRWSDBlock & spiCS_nFromSpiCtrl; any active submodule drives CS low; idle high.

- Error/status observation
- 0x04: busy latch set on start, cleared on SPI-side completion.
- 0x05: consolidated init/read/write error flags.
- 0x00: ID=0x12; 0x02/0x03: type/control mirrors; 0x0B: current clkDelay.

- Typical top-level use
- Program desired SCK via 0x0B; use type 00 for raw SPI bytes, 01 for SD init, 10/11 for block I/O.
- Move bulk data via dual-clock FIFOs; use force-empty to recover from aborts; ensure TX FIFO is filled before starting a write.
- Respect FIFO base read wait-state behavior and RX pop semantics.

- Safety and CDC practices
- ctrlStsRegBI provides clean start/reset pulses across clock domains; busy cleared on SPI-side completion.
- Dual-clock FIFOs bridge bus/spi clocks; force-empty delivered as one-shot toggle pulses in both domains.
- Pace force-empty commands and avoid flooding command sources; FIFO pointer sync is simple (not fully robust), rely on reasonable asynchronous behavior.

-- Assumptions and Limitations --

- Clocks and CDC
- Two independent domains (busClk, spiSysClk). Control strobes/reset crosses via pulse/stretch sync; FIFO flush via toggle CDC.
- FIFO pointers are binary with single-flop cross-domain sampling (no Gray code, no double-sync). Assumes modest clock skew; limitation: possible flag glitches/metastability under fully asynchronous/high-skew conditions.
- Some CDC chains are unreset; limitation: spurious pulses (e.g., force-empty) can occur at power-up.

- Bus interface
- Wishbone-like timing: immediate ack for most accesses; reads to FIFO bases (0x10, 0x20) add one-cycle wait. Assumes bus master holds strobe through wait.
- Address decode uses 16-byte windows (mask 0xF0); only one region active at a time. Reads are combinational; limitation: no registered readback—software must sample during strobe.
- RX FIFO is pop-on-read with 1-cycle latency; limitation: reading while empty returns stale/undefined data unless guarded by count/not-empty.

- SPI wire engine
- Fixed 8-bit, MSB-first, mode-0-like (SCK idle low, MOSI valid while low, sample MISO during low). No CPOL/CPHA/bit-order configurability. Limitation: incompatible with devices needing other modes.
- Single chip-select; CS is active-low AND across requesters. Assumes only one requester active; limitation: concurrent requests cause ambiguous CS ownership, CS held low.

- Command/byte staging
- sendCmd ORs two producer sources; bytes must be stable and mutually exclusive over the transaction. Limitation: overlapping producers corrupt fields via bitwise-OR.
- No timeout for missing response in WAIT_RX; only bounded busy-response count (512). Limitation: potential indefinite hang if rxDataRdy never asserts.
- Polling injects 0xFF writes unconditionally; assumption: TX path not full at that time.
- spiTxRxData is single-byte staging; overwrite permitted even when full. Assumes writers respect txDataFull==0; limitation: data clobber possible. RX ready can be suppressed by any clear; ready-set overrides clear in same cycle.

- SD-card flows
- Init uses legacy CMD0/CMD1 only; no CMD8/ACMD41 detection/loop. Limitation: may fail on SD v2.0/SDHC; CRC assumed off after init (fixed 0x95 for CMD0, 0xFF otherwise).
- Single-block operations (CMD17/CMD24) only; no multi-block/DMA. Read ignores CRC; write sends fixed CRC. Limitation: no CRC checking.
- Address field sent verbatim; assumption: caller supplies correct byte vs block address per card type.
- Fixed timeouts/counters; 512-byte transfer relies on 9-bit loop counter wrap. Limitation: mis-sizing breaks termination.

- FIFOs and memory
- Write enable is level-sensitive (stream-capable); read enable is rising-edge pop (must pulse). Limitation: holding read high pops only one item.
- No overflow/underflow protection; assumption: software/logic gates writes when full and reads when not empty.
- FIFO occupancy is eventually consistent across domains (CDC latency); not instantaneous.
- Dual-port RAM has 1-cycle registered read; same-address read/write collision behavior is device-dependent; no reset.

- Reset/configuration
- Soft reset is stretched; most regs clear, but SDAddr bytes do not. Limitation: SDAddr undefined after reset until programmed.
- SPI speed programmable (clkDelay); initSD forces slow (0x3b). Assumes software sets normal clkDelay post-init and avoids racing init overrides.

- Control planes
- ctrlStsRegBI: start pulse stretched/edge-detected into spiSysClk; transaction status cleared by SPI-side falling edge.
- spiCtrl samples spiTransCtrl only in idle; emits one-cycle request pulses; asserts CS low only for raw wire transfers (SD flows manage CS). Assumes single request at a time.

- Event spacing
- Force-empty uses toggle -> multi-flop sync -> one-shot; events must be spaced to avoid coalescing. RX BI read both returns current byte and issues pop.

IO PORTS

-- Clock and Reset Signals --

Clocks: Two internal clock domains. busClk is the primary bus clock and drives spiMasterWishBoneBI, the bus side of ctrlStsRegBI, sm_TxFifo write side (wrClk), and sm_RxFifo read side (rdClk). spiSysClk is the SPI system clock and drives spiCtrl, initSD, readWriteSDBlock, sendCmd, spiTxRxData, readWriteSPIWireData, and the opposite sides of the FIFOs (sm_TxFifo rdClk, sm_RxFifo wrClk). SCK (spiClkOut) is generated inside readWriteSPIWireData by dividing spiSysClk with a programmable half■period (clkDelay); it is an output to the SPI device (CPOL=0, idles low), not a separate internal domain. During SD card initialization, initSD forces a slower clkDelay, then reverts to the programmed value. Resets: Two sources. An external asynchronous rstFromWire and a software reset issued by writing ctrlStsRegBI at address 0x01 bit0. The bus■domain reset is created by ctrlStsRegBI as rstSyncToBusClkOut, a synchronous, active■high reset in busClk produced via a stretch/deassert scheme. rstSyncToBusClkOut is double■synchronized into spiSysClk to form rstSyncToSpiClkOut; all SPI■side modules use synchronous, active■high reset with this signal. Module behavior on reset: spiCtrl/initSD/readWriteSDBlock/sendCmd/spiTxRxData/readWriteSPIWireData clear internal FSMs/flags; chip■selects return high. sm_TxFifo and sm_RxFifo implement independent synchronous resets per domain (rstSyncToWrClk and rstSyncToRdClk); forceEmpty provides a domain■local soft flush that generates one■cycle pulses in both busClk and spiSysClk via a toggle synchronizer. spiMasterWishBoneBI has busClk■registered logic; the ack_delayed flip■flop is not explicitly reset but is functionally benign under normal bus usage. CDC related to resets/control: Start strobe from bus (ctrlStsRegBI 0x03) is stretched in busClk and re■synchronized into spiSysClk with a 3■flop chain to produce a single■cycle spiTransCtrl pulse. SPI■side status (spiTransStatus) is triple■sampled in busClk; a falling edge clears the bus■side in■progress flag. Notes/caveats: FIFO forceEmpty toggle synchronizers lack explicit reset on SPI■side stages and may produce a spurious initial pulse depending on power■up; FIFO pointer synchronization uses single■flop binary sampling, adequate for simple asynchronous crossings but not the most robust scheme.

-- Bus Interface Signals --

- Type: 8-bit, memory-mapped, Wishbone-like bus interface using strobe/ack handshake.
- Clock/reset domain:
- busClk (in): bus clock; all bus signals synchronous to busClk.
- rstSyncToBusClk (in): active-high synchronous reset for bus domain (deassertion is stretched by internal logic).
- Addressing:
- address[7:0] (in): byte address. Upper bits (if present) are ignored. Decoding uses address[7:4] (i.e., address & 8'hF0) to select 16-byte regions.
- Transfer control:
- writeEn (in): 1=write, 0=read.
- strobe_i (in): transfer qualifier/request; must remain asserted until ack_o is observed high. One transfer completes per ack_o.
- Data:
- busDataIn[7:0] (in): write data.
- busDataOut[7:0] (out): read data; driven from the selected block. Valid when ack_o is high.
- Handshake/acknowledge:
- ack_o (out): acknowledge for the current transfer.
- Default behavior: ack_o asserts combinationally with strobe_i (zero wait state) for all writes and most reads.
- Special-case read latency: reads to FIFO base data ports (address 0x10 RX FIFO pop; 0x20 TX FIFO data port if read) insert a one busClk cycle wait. Master must hold strobe_i and control signals stable across the extra cycle. busDataOut is valid in the cycle ack_o asserts.
- Unmapped addresses: busDataOut=8'h00; ack_o asserts with zero wait (same as default). No byte enables; bus is strictly 8-bit.

-- SPI Interface Pins --

SPI Interface Pins: SCK (spiClkOut, output) — generated by readWriteSPIWireData; idles low (CPOL=0-like); half-period programmable via 8-bit clkDelay at ctrl/status addr 0x0B; forced slow (~0x3B) during SD init and then returns to programmed rate; bits counted on SCK rising edges. MOSI (spiDataOut, output) — MSB-first; data updates while SCK is low; supports continuous streaming with no inter-byte gaps when data is available. MISO (spiDataIn, input) — sampled once per bit while SCK is low; received byte presented at each byte boundary. CS_n (spiCS_n, output, active-low) — aggregated arbitration: CS_n = initSD & readWriteSDBlock & spiCtrl (AND of active-low requests); any active submodule pulls CS low, otherwise CS remains high; spiCtrl asserts CS low for a single-byte TX/RX and releases after RX completes; initSD manages CS during command polling and clocks 0xFF with CS high pre-init; readWriteSDBlock holds CS low across CMD17/CMD24 and the 512-byte data phase. Reset defaults: SCK=0, MOSI=0, CS_n=1. Note: timing approximates SPI Mode 0 (idle low, MSB-first), but MISO is sampled during the low phase rather than exactly on the rising edge—verify target device compatibility.

-- Chip-Select and Arbitration Signals --

Chip-select generation and arbitration use active-low semantics. The top-level SPI chip-select output (spiCS_n) is computed as a logical AND of three internal, active-low contributors: spiCS_nFromSpiCtrl (raw SPI), spiCS_nFromInitSD (SD init), and spiCS_nFromRWSDBlock (SD block I/O). Equation: spiCS_n = spiCS_nFromInitSD & spiCS_nFromRWSDBlock & spiCS_nFromSpiCtrl. Any contributor driving 0 asserts the card select; only when all contributors drive 1 does spiCS_n deassert (1). There is no priority or masking; the AND implements a low-dominant arbitration so any active submodule keeps CS asserted. Reset/idle behavior: all contributors drive 1, so spiCS_n defaults high. Ownership by function: spiCtrl (type 00 raw byte) asserts CS for the duration of a single-byte transfer and deasserts immediately after rxDataRdy is observed and cleared; initSD holds CS high during preamble clocking and asserts/deasserts CS only around each command frame and response polling performed via sendCmd, releasing CS when the command engine completes; readWriteSDBlock asserts CS at request acceptance and holds it across the entire block transaction (command, token wait, 512-byte data, CRC, busy-release polling), deasserting at completion. Concurrency: control sequencing ensures only one submodule is active, but if overlap occurs any low keeps the card selected; software/flow must prevent contention. Non-participants: sendCmd does not drive CS and assumes its caller has asserted it; the SPI shifter, byte staging, FIFOs, bus, and registers do not source CS. Clock-rate changes (e.g., clkDelay adjustments by initSD) are orthogonal to CS arbitration and do not affect the logic above.

-- Control and Status Signals --

Bus-visible control and status: ID (0x00, RO=0x12); Soft reset (0x01, WO bit0=1) generates stretched, synchronous reset in busClk and a double-synchronized reset into spiSysClk; Transaction type (0x02, RW bits[1:0]: 00=raw byte, 01=SD init, 10=SD block read, 11=SD block write); Start strobe (0x03, WO bit0=1) issues a one-cycle pulse in the SPI domain; Busy (0x04, RO level 1=in progress) sets when a start is accepted and auto-clears when the SPI-side FSM returns to idle; Error flags (0x05, RO aggregated bits for init/read/write) indicate occurrence of an error, cleared by reset; Direct-access data (0x06, RW): write loads a TX byte for raw SPI, read returns last captured RX byte after a completed raw SPI transaction; SD 32-bit address (0x07..0x0A, RW, little-endian by address) used by block transactions; SPI clock delay (0x0B, RW 8-bit half-period in busClk cycles), temporarily overridden to a slow value during SD init. FIFO control/status via windows: RX FIFO @0x10 window—data read/pop (0x10, RO; BI inserts exactly one wait cycle on reads), count high/low (0x12/0x13, RO [15:8]/[7:0]), force-empty (0x14, WO bit0=1) producing one-shot pulses in both busClk and spiSysClk; TX FIFO @0x20 window—data enqueue (0x20, WO; immediate ack on writes; reads insert one wait cycle but are typically unused), force-empty (0x24, WO bit0=1) producing one-shot pulses into both domains. Wishbone-like BI ack behavior: ack is immediate for most accesses; reads to 0x10 or 0x20 base insert exactly one wait cycle and require strobe to be held across the wait. External SPI signals: spiCS_n is active-low and equals the AND of submodule CS_n terms (any active submodule asserts CS low), spiClkOut (SCK) idles low with period set by clkDelay, spiDataOut (MOSI) and spiDataIn (MISO) carry byte-wise transfers with RX bytes flagged internally and captured every 8 bits. Reset/defaults: CS high, requests low, busy low, error flags cleared, clkDelay cleared to default until programmed; resets may be issued via external rstFromWire or the soft reset register. Internal control/status (pulse/level semantics in spiSysClk unless stated): Start strobe creates one spiSysClk pulse (spiTransCtrl); spiCtrl emits one-cycle request pulses (txDataWen, SDInitReq, readWriteSDBlockReq) and rxDataRdyClr pulses; sendCmdReq is a one-cycle pulse, sendCmdRdy is a level (1=idle/ready); byte engine and shifter expose txDataEmpty (level), accept a byte when txDataFull is set, acknowledge with txDataFullClr (pulse), assert rxDataRdySet (pulse) at each received byte boundary; FIFO force-empty controls generate one-shot pulses in both domains via toggle-based CDC. Error reporting: initSD sets initError on failure; readWriteSDBlock sets readError or writeError on failure; bus-visible error bits reflect error presence only.

-- CDC and Handshake Signals --

- Clock domains and scope
- busClk: software/Wishbone interface, bus-side of RX/TX FIFOs.
- spiSysClk: transaction control (spiCtrl), SD init (initSD), block I/O (readWriteSDBlock), command sender (sendCmd), byte staging (spiTxRxData), wire engine (readWriteSPIWireData), SPI-side of RX/TX FIFOs.

- CDC paths and mechanisms
- Reset: software reset bit is stretched in busClk and double-synchronized into spiSysClk (rstSyncToSpiClkOut). A corresponding bus-side stretched reset (rstSyncToBusClkOut) exists for local

use.
- Start/Type: spiTransCtrlSTB is stretched in busClk and edge-detected in spiSysClk to create a 1-cycle spiTransCtrl pulse. spiTransTypeSTB is sampled in spiSysClk to produce spiTransType.
- Status return: spiTransStatus (spiSysClk) crosses to busClk via a 3-flop synchronizer; the falling edge is used to clear the bus-side in-progress flag.
- Dual-clock FIFOs (sm_TxFifo, sm_RxFifo): independent wrClk/rdClk with 1-cycle read latency. Opposite pointers are single-flop synchronized in binary form to derive full/empty/count (adequate for simple ratios; not fully robust for arbitrary async ratios). Force-empty uses a toggle-based CDC generating one clean, 1-cycle pulse in each domain (forceEmptySyncToBusClk, forceEmptySyncToSpiClk).
- Bus read alignment: Wishbone READs of FIFO base addresses (RX 0x10, TX 0x20) insert a 1-cycle ack delay to match FIFO read latency and ensure data readiness.
- SPI clock selection: during initSD, spiClkDelayOut overrides the normal bus-programmed clkDelay; the value is consumed in spiSysClk by the wire engine.

- Handshake conventions
- All request signals in spiSysClk are 1-cycle pulses; ready/status signals are level and deassert when idle/complete.
- FIFO enables: write enables are level-sensitive in the local clock; read enables are rising-edge sensitive.
- tx/rx byte staging: txDataWen sets txDataFull; wire engine asserts txDataFullClr when it accepts a byte. RX side asserts rxDataRdySet at byte boundaries; any consumer may pulse rxDataRdyClr to re-arm.

- Handshake networks and signal inventory
- spiCtrl
- Inputs: spiTransCtrl (pulse), spiTransType.
- Outputs: SDInitReq (pulse), readWriteSDBlockReq[1:0] (pulses), txDataWen (pulse), rxDataRdyClr (pulse), spiCS_nFromSpiCtrl.
- Waits on: SDInitRdy, readWriteSDBlockRdy, rxDataRdy; drives spiTransStatus while busy.
- initSD
- Handshake: SDInitReq (in), SDInitRdy (out); uses sendCmdReq/sendCmdRdy; consumes txDataEmpty; may assert spiCS_nFromInitSD.
- Controls spiClkDelayOut during init.
- readWriteSDBlock
- Handshake: readWriteSDBlockReq[1:0] (in), readWriteSDBlockRdy (out); uses sendCmdReq/sendCmdRdy; monitors txDataEmpty.
- Streaming write: txFifoRen pulses to fetch bytes; pushes via txDataWen when txDataFull==0.
- Streaming read: emits 0xFF clocks; on rxDataRdy, captures rxDataIn and pulses rxFifoWen; counts 512 bytes; clocks CRC; uses rxDataRdyClr.
- May assert spiCS_nFromRWSDBlock.
- sendCmd
- Handshake: sendCmdReq (OR of sources) starts frame; sendCmdRdy=1 indicates idle/complete.
- TX path: issues txDataWen when txDataFull==0; waits for txDataEmpty before state advances.
- RX polling: emits 0xFF, pulses rxDataRdyClr, waits on rxDataRdy; evaluates respByte[7] (0=done, 1=busy); respTout indicates retry limit reached.
- spiTxRxData
- Inputs: txDataWen from up to four producers; rxDataRdySet from wire engine.
- Outputs: txDataFull (level), txDataFullClr (pulse from wire engine), rxDataRdy (level), rxDataRdyClr (consumers).
- readWriteSPIWireData
- Consumes: txDataFull; produces: txDataFullClr, rxDataRdySet, txDataEmpty; drives spiClkOut and MOSI/MISO; supports back-to-back bytes.

- sm_TxFifo
- busClk side: fifoWEn (level), forceEmptySyncToBusClk (pulse), status (fifoFull, fifoEmpty, count).
- spiSysClk side: fifoREn (pulse), forceEmptySyncToSpiClk (pulse).
- sm_RxFifo
- spiSysClk side: fifoWEn (level), forceEmptySyncToSpiClk (pulse).
- busClk side: fifoREn (rising-edge on read), forceEmptySyncToBusClk (pulse), numElementsInFifo exposed.
- Bus interface (spiMasterWishBoneBI)
- Region selects: ctrlStsRegSel, rxFifoSel, txFifoSel.
- ack_o immediate except READs at 0x10/0x20 (1-cycle wait).
- Chip-select arbitration
- Top-level spiCS_n = spiCS_nFromInitSD & spiCS_nFromRWSDBlock & spiCS_nFromSpiCtrl (any submodule can assert low; default high when all deassert).

- Constraints and guarantees
- Space force-empty writes so toggle CDC can produce distinct pulses in both domains; each bus write with bit0=1 yields exactly one pulse per domain.
- Producers must gate txDataWen on txDataFull==0 to avoid overwrite; maintain pacing to keep back-to-back streaming.
- Respect FIFO full/empty to avoid overflow/underflow.
- sendCmd has no absolute timeout for no-response; the system can stall in WAIT_RX if rxDataRdy never asserts.
- Pointer CDC in FIFOs uses single-flop binary synchronization; suitable for simple systems, but not guaranteed for all async ratios.

ARCHITECTURE

-- Top-Level Block Overview --

spiMaster is a memory-mapped SPI/SD master that bridges a byte-wide host bus to a byte-serial SPI wire engine and SD-card control flows. It operates across two clock domains (busClk and spiSysClk) with explicit clock-domain crossings for control, status, and FIFO data.

External interfaces
- Bus: Wishbone-like byte data, address decode into 16-byte windows, separate read/write strobes, acknowledge. Zero-wait-state for all accesses except a single inserted wait on base reads of 0x10 (RX) and 0x20 (TX).
- SPI: MOSI (spiDataOut), MISO (spiDataIn), SCK (spiClkOut with programmable half-period), CS_n (spiCS_n), arbitrated across subfunctions and idles high.

Top-level composition
- Bus front-end (spiMasterWishBoneBI): decodes three regions—0x00..0x0F control/status, 0x10..0x1F RX FIFO, 0x20..0x2F TX FIFO—and muxes read data with the defined wait policy.
- Control/Status registers (ctrlStsRegBI): start/soft-reset pulses, transaction type, direct-access TX byte, 32-bit SD address, clock delay; readbacks include ID, busy, error flags, direct-access RX, address, and delay. All controls/status cross between busClk and spiSysClk.
- SPI controller (spiCtrl): dispatches a start+type into one-cycle request pulses for subfunctions and

holds busy until completion. Supported operations: raw single-byte SPI, SD init, SD single-block read, SD single-block write.
- SD control: initSD performs SPI-mode bring-up at a forced slow SCK; sendCmd frames and polls SPI commands; readWriteSDBlock manages CMD17/CMD24 and streams 512-byte payloads via FIFOs.
- Byte engine and shifter: spiTxRxData arbitrates multiple TX sources and flags RX-ready; readWriteSPIWireData shifts MSB-first bytes and generates SCK to sustain back-to-back streaming.
- Dual-clock FIFOs: sm_TxFifo (bus→SPI) and sm_RxFifo (SPI→bus) with force-empty support and RX count visibility; used for block data paths.
- CS arbitration: top-level CS_n is the logical AND of requests from initSD, readWriteSDBlock, and raw-byte path, ensuring deterministic chip-select control.

Behavioral highlights
- Programmable SPI speed (clkDelay), forced slow during SD init.
- Continuous byte streaming when supply/consume conditions are met.
- Software-visible flush on both FIFOs; one-cycle wait only on FIFO base reads.

-- Data Path Overview (TX/RX) --

End-to-end path: Software writes TX bytes via TX FIFO (0x20 data port; optional flush at 0x24) or single-byte direct access (0x06). Bytes traverse busClk→sm_TxFifo→spiSysClk→spiTxRxData→readWriteSPIWireData→MOSI. Received bits on MISO are assembled per byte in readWriteSPIWireData, latched in spiTxRxData, then consumed by spiCtrl/sendCmd/readWriteSDBlock; block-read bytes are pushed into sm_RxFifo (SPI-side writes) and software reads them back at 0x10 (count at 0x12/0x13; optional flush at 0x14). SPI speed is set by clkDelay (0x0B), with initSD temporarily overriding it during bring-up.

Clock-domain crossings: Dual-clock FIFOs bridge busClk and spiSysClk (TX: bus write/SPI read; RX: SPI write/bus read). Force-empty registers (0x24, 0x14) generate CDC-safe flush pulses. Wishbone inserts a 1-cycle wait only on FIFO data ports (0x10, 0x20); other accesses are zero-wait. RX FIFO pop semantics: each read of 0x10 returns the current element and asserts a pop to fetch the next; gate reads by count>0 to avoid empty pops.

TX staging and wire engine: Multiple writers (single-byte via spiCtrl, sendCmd frame bytes, readWriteSDBlock stream, initSD clocks) push into spiTxRxData; txDataFull advertises availability and txDataFullClr acknowledges consumption. readWriteSPIWireData accepts a staged byte when txDataFull==1, asserts txDataFullClr, shifts MOSI, samples MISO, and asserts rxDataRdySet at each byte boundary. Continuous streaming is supported when the next byte becomes available by the boundary; no idle cycles are inserted.

RX staging and delivery: Each received byte triggers rxDataRdySet; consumers must pulse rxDataRdyClr after capture to accept the next byte. For block read (type 10), readWriteSDBlock waits for start token 0xFE, then for 512 bytes: waits rxDataRdy, writes to sm_RxFifo, and clears rxDataRdy; two trailing CRC bytes are clocked and ignored. For single-byte (type 00), spiCtrl clears rx ready and exposes the byte at 0x06. For block write (type 11), readWriteSDBlock fetches 512 bytes from sm_TxFifo and transmits them after token 0xFE; expects data-response 0x05.

SPI timing and CS: CPOL=0; MOSI updates while SCK is low, sampling in the low phase; bit count advances on SCK rising edge; spiClkOut half-period set by clkDelay. Chip-select spiCS_n is the AND of intents from spiCtrl, initSD, and readWriteSDBlock, ensuring any active submodule holds CS low; CS remains low across an entire block transaction.

Software-visible behavior: TX FIFO writes are combinational strobes; software should check not-full before writing. RX/TX FIFO data port reads incur a 1-cycle wait; reads from 0x10 pop one element and fetch the next. Direct-access byte at 0x06 supports single transfers (write TX, then read RX on completion).

Status and errors: ctrlStsReg exposes busy/idle and SD error flags. sendCmd indicates completion via sendCmdRdy, returns respByte, and asserts respTout on timeout. readWriteSDBlock sets readError/writeError for command, token, or busy-timeout issues.

-- Clock Domains and CDC Strategy --

- Clock domains
- busClk domain:
- Bus front-end and register block: spiMasterWishBoneBI, ctrlStsRegBI (includes reset deassertion stretcher).
- FIFO bus interfaces: sm_TxfifoBI (TX write port), sm_RxfifoBI (RX read port).
- spiSysClk domain:
- SPI control/data path: spiCtrl, initSD, readWriteSDBlock, sendCmd, spiTxRxData, readWriteSPIWireData.
- FIFO SPI interfaces: sm_TxFifo (read side), sm_RxFifo (write side).
- ctrlStsRegBI SPI-side synchronizers and shadow registers for control/status.
- Generated SPI clock (spiClkOut):
- Produced inside readWriteSPIWireData by dividing spiSysClk per clkDelay.
- Used only as an external interface clock; no internal logic is clocked by spiClkOut.

- Reset strategy and CDC
- Sources: external rstFromWire and software reset (ctrlStsRegBI at 0x01).
- busClk domain: ctrlStsRegBI stretches reset deassertion via a shift register to create rstSyncToBusClkOut.
- spiSysClk domain: rstSyncToBusClkOut is double-synchronized to generate rstSyncToSpiClkOut.
- FIFOs: accept per-domain synchronous reset; also support a cross-domain force-empty (flush) pulse (see FIFOs below).

- Configuration/control CDC (busClk -> spiSysClk)
- Start pulse (addr 0x03):
- Level-stretched in busClk using a shift register.
- 3-flop synchronized into spiSysClk, with edge detection to produce a single-cycle spiTransCtrl pulse.
- Static configuration fields sampled in spiSysClk without explicit handshake:
- Transaction type (0x02), direct-access TX byte (0x06), SD address (0x07..0x0A), clkDelay (0x0B).
- Software must write and hold these stable before asserting start.

- Status/data CDC (spiSysClk -> busClk)
- Transaction status:
- spiTransStatus is triple-synchronized into busClk.
- A falling edge clears the bus-side in-progress latch (0x04).
- Error flags (init/read/write): sampled each busClk from SPI-side sources.
- Direct-access RX byte (0x06 read): captured in busClk from the SPI byte engine output; relies on producer-ready indication, no explicit handshake.

- Streaming data CDC via asynchronous FIFOs
- sm_TxFifo (bus->SPI): busClk writes; spiSysClk reads.
- sm_RxFifo (SPI->bus): spiSysClk writes; busClk reads.
- Pointer/flag scheme:
- Independent binary read/write pointers.
- Single-stage pointer synchronization into the opposite domain.

- Full/empty derived from synchronized pointer comparisons; element count is eventually consistent across domains.
- Enable semantics:
- Write side: level-sensitive per wrClk.
- Read side: rising-edge detector per rdClk to pop one element per pulse.
- Force-empty (flush):
- Initiated from bus side via a toggle-based CDC.
- Generates one-cycle pulses in both domains to synchronously clear pointers/flags.
- Bus latency alignment:
- Wishbone interface inserts a 1-cycle wait-state for reads of FIFO data ports (0x10/0x20 base) to match FIFO/RAM read latency; bus masters must hold strobe through the wait-state.

- Intra-spiSysClk handshakes
- All SD/sequencer/byte-engine interactions are synchronous to spiSysClk using single-cycle request pulses and ready/empty flags (e.g., txDataFull/Clr, rxDataRdy/Clr, sendCmdReq/Rdy). No CDC involved.

- Design guarantees and cautions
- No internal logic uses spiClkOut as a clock; all SPI timing derives from spiSysClk with programmable delays.
- Event crossings use level-stretch + multi-flop synchronization (start/reset) or toggle-to-pulse (force-empty) to create single-cycle pulses in the destination domain.
- FIFO CDC uses single-flop, binary pointer synchronization; adequate for modest frequency ratios, but Gray-coded pointers with two-flop synchronizers are recommended for more robust asynchronous operation.
- Force-empty toggle synchronizers on the SPI side lack explicit reset; a benign spurious pulse at power-up is possible.
- Multi-bit status sampled in busClk without handshake may be incoherent if changing near a bus edge; mitigated by update-on-byte boundaries and typical software read timing.

-- Bus Front-End and Address Windowing --

The bus front-end is implemented by spiMasterWishBoneBI with a simple Wishbone-like, byte-wide access model. Address windowing uses a 0xF0 mask (address[7:4]) to create three 16-byte regions. Decode and data muxing are combinational; selects are mutually exclusive. Unmapped addresses produce 0x00 read data and no select.

Address windows and selects:
- 0x00–0x0F: Control/Status block (ctrlStsRegBI) -> ctrlStsRegSel
- 0x10–0x1F: RX FIFO block (sm_RxFifo via sm_RxfifoBI) -> rxFifoSel
- 0x20–0x2F: TX FIFO block (sm_TxFifo via sm_TxfifoBI) -> txFifoSel
- Any other address: no select asserted; read returns 0x00

ACK generation and wait-states:
- Default policy: zero-wait-state; ack_o follows strobe_i (immediate ACK) for all writes and for reads except as noted below.
- Special-case reads: a single wait-state is inserted for reads to the base offsets of the FIFO windows (0x10 for RX data, 0x20 for TX data). ack_o = strobe_i & strobe_i_d1, where strobe_i_d1 is strobe_i registered for one busClk cycle. The master must hold strobe_i across the extra cycle. This delay aligns with the registered read latency in the FIFO datapath.

Read-data muxing:

- dataOut is the combinational mux of the selected region’s read data.
- Unmapped addresses drive dataOut = 0x00.

Access gating:
- All sub-block bus enables are qualified by strobe_i and writeEn (for writes); read enables are block- and offset-specific.
- RX FIFO window: reading 0x10 returns the current FIFO output and pops one element via a combinational fifoREn; back-to-back reads drain at one element per bus cycle once primed. Element count is exposed at 0x12/0x13. Force empty at 0x14 generates a synchronized one-shot flush in both busClk and spiSysClk via a toggle-based CDC.
- TX FIFO window: writing 0x20 enqueues data (immediate ACK); software must avoid overflow (not enforced by the front-end). Force empty at 0x24 behaves like RX (toggle-based CDC). Reads in 0x20–0x2F return 0x00.

CDC considerations (front-end related):
- ctrlStsRegBI stretches bus-domain start/reset pulses and synchronizes them into spiSysClk; status/data are sampled back to busClk.
- FIFO force-empty commands use a toggle synchronizer to create single-cycle pulses in both clock domains; spi-side sync flops may produce a benign power-up pulse depending on initialization.

-- Control/Status Register Block --

Control/Status Register Block

Overview
- Byte-wide register window at 0x00..0x0F selected by spiMasterWishBoneBI.
- Zero-wait-state access: ack asserted immediately on reads and writes within this window.
- Provides software control, status, and configuration for SPI/SD transactions across busClk and spiSysClk domains.

Register map (byte addresses relative to 0x00)
- 0x00 (RO): Module ID/version. Returns 0x12.
- 0x01 (WO): Soft reset request. Write bit0=1 to issue a synchronized reset. No readback.
- 0x02 (RW): SPI transaction type (data[1:0]). 00: raw SPI byte; 01: SD init; 10: SD single-block read; 11: SD single-block write. Reads return the last written type in the bus domain.
- 0x03 (WO/RZ): Transaction start strobe. Write bit0=1 to start; generates a one-bus-cycle strobe that becomes a one-cycle pulse in spiSysClk. Read returns only the current-cycle strobe level (transient; not latched).
- 0x04 (RO): Transaction status. Bit0=1 indicates in-progress; set by a write to 0x03 and auto-clears when the SPI side reports completion. Other bits read 0.
- 0x05 (RO): Error flags (summaries). bit0: SD init error (nonzero initSD.initError); bit1: SD read error (readWriteSDBlock.readError != 2'b00); bit2: SD write error (readWriteSDBlock.writeError != 2'b00). Remaining bits read 0.
- 0x06 (RW): Direct-access data. Write: TX byte for raw SPI transfers. Read: last RX byte captured from the SPI engine.
- 0x07..0x0A (RW): 32-bit SD address, little-endian by address. 0x07: [7:0], 0x08: [15:8], 0x09: [23:16], 0x0A: [31:24].
- 0x0B (RW): SPI clock half-period delay (clkDelay). Programs SCK half-period; initSD temporarily overrides to a slower value during bring-up, then normal operation uses the programmed value.
- Unmapped addresses within 0x00..0x0F return 0x00 on read; writes have no effect.

Access semantics and clock-domain crossings

- Writes occur when writeEn && ctrlStsRegSel && strobe_i are asserted; reads are combinational and sampled with ack.
- Start pulse (0x03): bus-cycle strobe is stretched and double-synchronized to generate a one-cycle pulse in spiSysClk; spiCtrl samples this pulse when idle and dispatches per spiTransType.
- Status (0x04): bit0 is latched in the bus domain, set on start, and auto-clears when spiCtrl indicates completion (falling edge of spiTransSts in spiSysClk, synchronized into busClk). Use 0x04 for in-progress polling; 0x03 readback is transient.
- Soft reset (0x01): produces a stretched synchronous reset in busClk and a double-synchronized reset in spiSysClk. Clears control strobes/types/data; SDAddr bytes (0x07..0x0A) are not explicitly reset and should be reprogrammed by software.
- Direct-access data (0x06): TX byte and transaction type (0x02) are sampled into the SPI domain; the RX byte from the SPI engine is mirrored back into the bus domain for read.

Interaction with submodules
- spiCtrl consumes spiTransCtrl (start) and spiTransType, drives spiTransSts, and manages chip-select/timing handshakes.
- initSD and readWriteSDBlock consume SDAddr and clkDelay; they report error conditions summarized at 0x05.
- readWriteSPIWireData uses clkDelay for SCK timing; spiTxRxData carries direct-access TX/RX bytes.

Software usage
- Optional: write 0x01 bit0=1 to reset; wait until 0x04 bit0 reads 0 and reinitialize as needed.
- Program 0x07..0x0A (SDAddr), 0x02 (type), and 0x0B (clkDelay). For raw SPI (type 00), also write TX byte to 0x06.
- Start by writing 0x03 bit0=1.
- Poll 0x04 bit0 until it reads 0 to detect completion.
- Read 0x06 for the RX byte (if applicable) and 0x05 for error flags.

Bus timing notes
- Control/Status window accesses (0x00..0x0F) are zero-wait-state. Special one-cycle read wait-states apply only to RX/TX FIFO base addresses (0x10 and 0x20) and do not affect this block.

-- Transaction Dispatch FSM --

Purpose: Transaction Dispatch FSM (spiCtrl, spiSysClk domain) arbitrates and starts one of four SPI/SD operations, manages raw SPI chip-select, and exposes a busy status until completion.
Interfaces:
- Inputs: spiTransCtrl (1-cycle start pulse), spiTransType[1:0] (00=raw SPI byte, 01=SD init, 10=SD block read, 11=SD block write), rxDataRdy (raw byte complete), SDInitRdy (init engine idle/done=1), readWriteSDBlockRdy (block I/O idle/done=1).
- Outputs: txDataWen (1-cycle start raw byte), rxDataRdyClr (1-cycle clear raw RX-ready), SDInitReq (1-cycle request init), readWriteSDBlockReq[1:0] (1-cycle code for block read/write), spiCS_n_fromSpiCtrl (active-low CS, driven low only for raw SPI), spiTransSts (busy/in-progress).
Behavior:
- Type 00 (raw SPI byte): On spiTransCtrl when idle, set busy, drive CS low, pulse txDataWen, wait rxDataRdy=1, then pulse rxDataRdyClr, release CS high, clear busy.
- Type 01 (SD init): Pulse SDInitReq, wait SDInitRdy=1, clear busy; dispatcher keeps CS high (initSD owns CS).
- Type 10 (SD block read): Pulse read code on readWriteSDBlockReq, wait readWriteSDBlockRdy=1, clear busy; submodule owns CS.
- Type 11 (SD block write): Pulse write code, wait readWriteSDBlockRdy=1, clear busy; submodule

owns CS.
State machine:
- Idle: Accept spiTransCtrl, latch/interpret spiTransType, issue corresponding 1-cycle request, set spiTransSts=1.
- Busy states: RawBusy waits on rxDataRdy then clears; SDInitBusy waits on SDInitRdy; SDBlockBusy waits on readWriteSDBlockRdy; all return to Idle and clear spiTransSts.
Acceptance: New start pulses are accepted only in Idle; pulses during busy are ignored (no queuing).
Chip-select policy: Top-level CS is AND of initSD, readWriteSDBlock, and spiCtrl CS. spiCtrl asserts CS low only for raw SPI; SD submodules manage their own CS windows.
Reset: All outputs to safe defaults (CS high, request pulses low, rxDataRdyClr low, spiTransSts low); FSM enters Idle.
CDC and bus interaction:
- ctrlStsRegBI stretches bus write and generates a clean 1-cycle spiTransCtrl in spiSysClk; spiTransType is sampled in spiSysClk before dispatch.
- spiTransSts is returned to bus with falling-edge clear synchronization.
Software usage:
- Program spiTransType and arguments (e.g., direct-access TX byte, SD address) via ctrlStsRegBI; write start to produce spiTransCtrl; poll status until spiTransSts deasserts. For raw SPI, read RX byte from direct-access register; for SD operations, read error/status from respective submodules.
Notes:
- One-cycle request pulse per operation; busy stays high until ready/idle from the selected submodule.
- Type coding: 00 raw, 01 init, 10 read, 11 write; readWriteSDBlockReq encodes read vs write per readWriteSDBlock convention.
- Error reporting is handled by initSD/readWriteSDBlock and mirrored by ctrlStsRegBI; dispatcher itself does not generate errors.
- SPI speed: initSD may override clkDelay for slow bring-up; other flows use programmed clkDelay.

-- SD Initialization Controller --

SD Initialization Controller (initSD)

Purpose
- Sequences an SD card into SPI mode and out of idle at power-up/reset using legacy CMD0→CMD1 flow. Runs the SPI link at a reduced clock during init, generates the required pre-clock SCKs, then restores normal clock on completion. Reports a 2-bit status code.

External interfaces
- Start/ready: input SDInitReq (one-cycle start pulse from spiCtrl when transaction type=01), output SDInitRdy (0 while busy, 1 when complete).
- SPI clock control: input spiClkDelayIn (normal programmed divider), output spiClkDelayOut (forced to 0x3b during init, otherwise passes through spiClkDelayIn).
- Chip select: output spiCS_n driven by initSD; high during pre-clocks, low only for command/response windows. Top-level CS is active-low AND across submodules to allow independent assertion when others are idle.
- TX path (shared): outputs txDataOut[7:0], txDataWen; inputs txDataFull, txDataEmpty. initSD honors txDataFull backpressure and can wait for txDataEmpty.
- Command engine (sendCmd) handshake: outputs sendCmdReq, cmdByte, dataByte1..4, checkSumByte; inputs sendCmdRdy, respByte (R1), respTout.
- RX clear: output rxDataRdyClr present but held low by initSD.
- Status to software: output initError[1:0] surfaced via register block (SDInitErrorSTB). Transaction busy/complete exposed via spiTransStatusSTB.

Operation sequence

1) Pre-clocks: Force slow SPI (spiClkDelayOut=0x3b). With CS high, write 0xA0 (160) bytes of 0xFF into the TX path (1280 SCKs). Wait for txDataEmpty.
2) CMD0 loop: Assert CS low. Issue CMD0 (cmdByte=0x40, arg=0x00000000, checkSumByte=0x95) via sendCmdReq. Wait for sendCmdRdy, deassert CS, then check response: expect R1=0x01 and respTout=0. If not, retry until loopCnt reaches 0xFF; on exhaustion set initError=2’b01 and complete.
3) CMD1 loop: Assert CS low. Issue CMD1 (cmdByte=0x41, arg=0x00000000, checkSumByte=0xFF). Between retries, insert paced delay using nested counters (delCnt2 rolls 0..0xFF per step of delCnt1 up to ~0x177). On sendCmdRdy, deassert CS and check response: expect R1=0x00 and respTout=0. If not, retry until loopCnt reaches 0xFF; on exhaustion set initError=2’b10 and complete. On success, set initError=2’b00.
4) Exit: Raise SDInitRdy, restore spiClkDelayOut to spiClkDelayIn (normal speed), release CS high.

Flow control and timeouts
- TX path backpressure respected (txDataFull). Pre-clock phase waits on txDataEmpty before proceeding.
- Command/response is fully handshaked with sendCmd. respTout reflects an internal busy-poll timeout inside sendCmd (e.g., after 512 polls). If sendCmd never asserts completion (e.g., RX never ready), initSD can stall because it waits on sendCmdRdy.

Software usage
- Program normal SPI clock divider at register address 0x0B (optional). Set transaction type=01 at 0x02, then start via 0x03 bit0. Poll 0x04 (busy) until clear. Read 0x05 (SDInitErrorSTB): 00=OK, 01=CMD0 failed, 10=CMD1 failed. Optional reset via 0x01 bit0.

Assumptions/limitations
- Legacy initialization only: CMD0 then CMD1. Does not issue CMD8 or ACMD41 (CMD55+ACMD41). SD v2.0/HC/XC cards may not complete init. CRCs are fixed (valid 0x95 for CMD0, 0xFF otherwise) under the assumption post-SPI-mode CRCs are not checked by the card.

-- SD Block Read/Write Controller --

SD Block Read/Write Controller

Purpose and scope
- Performs single-block (512-byte) SD-card transfers in SPI mode, covering command issuance (CMD17 read, CMD24 write), chip-select control, SPI byte clocking, token detection, data streaming via dual-clock FIFOs, and completion/error reporting.

Activation and handshakes
- Triggered by spiCtrl via readWriteSDBlockReq: 2'b10 = read (CMD17), 2'b01 = write (CMD24). spiCtrl maps host types 10/11 to these encodings.
- readWriteSDBlockRdy = 1 when idle/ready; goes low during a transaction and returns high on completion.
- 32-bit blockAddr sourced from ctrlStsRegBI registers at addresses 0x07–0x0A.
- Errors reported to ctrlStsRegBI at 0x05; transaction status/readiness at 0x04.

SPI command phase
- Frames command as: 0xFF preamble, cmdByte (0x51 read, 0x58 write), 4 address bytes (blockAddr[31:0]), checksum 0xFF (CRC assumed disabled post-init).
- Expects R1 response = 0x00. A nonzero R1 or response timeout sets command-failure error (01) and aborts.

Write flow (CMD24)
- After R1=0: send two 0xFF gap bytes then start-data token 0xFE.
- Stream 512 data bytes from TX application FIFO (sm_TxFifo) to the SPI byte engine:
- Pull bytes via txFifoRen and push when the SPI byte engine staging register is ready (txDataFull == 0).
- 9-bit loop counter runs 0–511.
- Send two trailing 0xFF CRC bytes (ignored by the card when CRC is off).
- Data-response check: accept only if resp[4:0] == 5'h05; else set writeError = 10.
- Busy release: poll by sending 0xFF until MISO returns nonzero (card not busy). Timeout sets writeError = 11.
- Deassert CS and signal ready (readWriteSDBlockRdy = 1).

Read flow (CMD17)
- After R1=0: poll by sending 0xFF until start-data token 0xFE is received; timeout sets readError = 10.
- Read 512 bytes:
- For each byte, send 0xFF to clock the bus; when rxDataRdy = 1 from the SPI byte engine, capture rxDataIn and write to RX application FIFO (sm_RxFifo) via rxFifoWen.
- 9-bit loop counter runs 0–511.
- Clock out two trailing CRC bytes (ignored).
- Wait for TX path to drain (txDataEmpty = 1), then deassert CS and signal ready.

Chip-select and SPI timing
- readWriteSDBlock holds spiCS_n low for the full transaction; top-level CS arbitration is active-low AND across submodules.
- SPI byte engine (readWriteSPIWireData) generates SCK; MSB-first shifting; SCK idles low.
- Half-period programmable via clkDelay at ctrlStsRegBI address 0x0B. Supports back-to-back byte streaming when upstream keeps the engine fed (txDataFull managed).

Data movement and FIFOs
- TX FIFO (sm_TxFifo): bus-side enqueues block data at address window base 0x20; SPI-side drains during writes.
- Force-empty via write to 0x24 bit0.
- RX FIFO (sm_RxFifo): SPI-side writes received bytes; bus-side pops reads from base 0x10.
- Count available at 0x12/0x13; force-empty via write to 0x14 bit0.
- Wishbone BI imposes a 1-cycle wait-state for reads at base addresses 0x10 and 0x20; software must hold the read strobe an extra cycle.
- RX FIFO pop side-effect: reading 0x10 asserts fifoREn (rising-edge pulse); software should only read when count > 0 to avoid empty pops.

Timeouts and counters (approximate thresholds)
- Read start token wait: ~0x048.
- Write data-response wait: ~0x0F00.
- Busy release polling: ~0x0B6; delCnt1/delCnt2 pace polling loops.
- Data loop counter: 9-bit, wraps after 512 bytes.

Errors and status
- readError codes: 01 command failed; 10 no start-data token before timeout.
- writeError codes: 01 command failed; 10 invalid/missing data-response; 11 busy release timeout.
- Errors aggregated at ctrlStsRegBI 0x05; transaction status at 0x04.

Limits and assumptions
- Single-block transfers only.

- Addressing is transmitted verbatim; host must ensure correct byte vs block addressing per card type (init sequence does not include CMD8/ACMD41).
- CRC fixed to 0xFF for commands and data; assumes CRC disabled after card initialization.
- CDC notes: dual-clock FIFOs use simple binary pointer synchronization; force-empty uses toggle-based CDC pulses.

Host software usage
- Write sequence:
- Fill TX FIFO with 512 bytes at window 0x20.
- Program blockAddr at 0x07–0x0A.
- Set type 11 at 0x02; start at 0x03.
- Poll 0x04 until ready; read errors at 0x05.
- Read sequence:
- Program blockAddr at 0x07–0x0A.
- Set type 10 at 0x02; start at 0x03.
- Poll 0x04 until ready.
- Drain 512 bytes from RX FIFO base 0x10; monitor count at 0x12/0x13; read errors at 0x05.

Throughput guidance
- For writes, prepare the full 512 bytes in TX FIFO before starting.
- For reads, drain RX FIFO promptly to avoid overflow at higher SPI clock rates.

-- Command Sender Engine --

Command Sender Engine (sendCmd)

Role
- Serializes SD/SPI command frames onto the SPI byte path and polls for the single-byte R1 response.
- Operates entirely in the spiSysClk domain and uses the existing byte-level SPI engines: spiTxRxData (staging) and readWriteSPIWireData (wire shifter).

Producers and arbitration
- Two independent producers: initSD and readWriteSDBlock.
- Request OR-combine: sendCmdReq = sendCmdReq1 | sendCmdReq2; readiness: sendCmdRdy=1 when idle.
- Command fields (cmdByte, dataByte1..4, checkSumByte) are bitwise ORs of two source buses. Exactly one source must assert its request and drive fields at a time and must hold them stable from before request until sendCmdRdy returns high.

Handshake
- Idle when sendCmdRdy=1. A request is accepted when sendCmdReq is seen while idle; sendCmdRdy is deasserted until the command completes (response obtained or timeout).

Frame emission
- Byte write interface: txDataWen pulses one cycle per byte when txDataFull==0; each write sets txDataFull until consumed by the shifter. Backpressure: if txDataFull==1, writes stall.
- Exact transmit sequence:
- 0xFF preamble
- cmdByte
- dataByte1, dataByte2, dataByte3, dataByte4 (32-bit argument MSB-first)
- checkSumByte (CRC; e.g., 0x95 for CMD0; 0xFF commonly used otherwise)
- After last byte is staged, wait for txDataEmpty==1 to ensure the frame has fully exited before polling.

Response polling
- Poll loop provides SPI clocks by transmitting 0xFF bytes repeatedly.
- Per poll:
- Transmit 0xFF (subject to txDataFull backpressure)
- Pulse rxDataRdyClr to clear previous byte-ready indication
- Wait for rxDataRdy==1, then capture rxDataIn into respByte
- If respByte[7]==0, response is ready; complete and present final R1 in respByte
- Else (busy bit=1), increment retry counter and repeat

Timeouts and completion
- Busy-retry limit: 512 polls. On reaching this limit without respByte[7]==0, assert respTout=1 and complete to idle; respByte holds the last observed value.
- No watchdog for missing rxDataRdy while waiting; engine can wait indefinitely if the wire path never presents a byte. Higher-level modules should bound overall wait times if needed.
- On successful response (MSB=0) or on timeout, sendCmdRdy returns high.

Chip-select and clocking
- Does not drive spiCS_n; callers must assert CS low before request and keep it low across the entire frame and polling sequence, then deassert after sendCmdRdy returns high.
- SPI clock rate (clkDelay) is owned by the wire-level shifter. Callers (e.g., initSD) may slow the clock during card initialization.

External contracts and interpretation
- Upstream modules are responsible for R1 interpretation:
- initSD: expects R1=0x01 for CMD0 and R1=0x00 for CMD1; uses respTout to flag timeout
- readWriteSDBlock: expects R1=0x00 for CMD17/CMD24; treats nonzero R1 or respTout as error
- respByte is the final R1 when completion is successful; MSB is treated as busy during polling only.

Dependencies on byte/wire engines
- spiTxRxData: txDataWen/txDataFull staging for TX; rxDataRdy and rxDataRdyClr handshake for RX byte availability.
- readWriteSPIWireData: executes the actual shift, drives txDataEmpty, and sets rxDataRdy.

Design notes and limitations
- No internal latching of command fields; improper multi-source driving will corrupt transmitted bytes via OR-combine.
- POLL_TX schedules 0xFF writes assuming txDataFull gating will eventually allow acceptance.
- All logic is synchronous to spiSysClk; no CDC is required within this engine.

Integration checklist
- Ensure only one producer drives sendCmdReq and the cmd/arg/CRC buses at a time; hold stable until sendCmdRdy=1.
- Assert CS low before sendCmdReq and keep it low until sendCmdRdy returns high.
- Provide the correct CRC byte for the command (0x95 for CMD0; 0xFF commonly used when CRC is disabled).
- Configure SPI clock rate appropriately (slow for init, higher for normal ops).
- Do not allow other TX writers to spiTxRxData while sendCmd is active.
- Handle respTout and nonzero R1 in caller error paths.

-- Byte Staging Register --

Byte Staging Register (spiTxRxData)
- Purpose: Central 8-bit, single-entry TX/RX staging element in the spiSysClk domain between multiple SPI byte producers/consumers and the SPI wire shifter (readWriteSPIWireData). Not a FIFO.

TX path
- Inputs: Up to four sources tx1..tx4 each provide txXDataIn[7:0] with a one-cycle txXDataWEn pulse.
- Arbitration: Fixed priority, tx1 highest → tx4 lowest. If multiple WEn in the same cycle, the highest priority wins.
- Latching/flag: On a winning WEn, txDataOut <= txXDataIn and txDataFull <= 1 (holds until cleared or overwritten).
- Handshake with wire engine: readWriteSPIWireData monitors txDataFull; when it accepts a byte it pulses txDataFullClr (1 cycle) to clear the flag and begin shifting.
- Precedence: If txXDataWEn and txDataFullClr are asserted in the same cycle, clear dominates the flag (txDataFull forced to 0). Data may still update; system logic should avoid this overlap.
- Overwrite: Writes are accepted even while txDataFull == 1; producers must only assert txXDataWEn when txDataFull == 0 unless willing to overwrite the staged byte.
- Streaming: Supports back-to-back bytes when the shifter clears txDataFull at each byte boundary and producers present the next byte promptly.

RX path
- Inputs: rxDataIn[7:0] with rxDataRdySet pulse from the wire engine after each shifted byte.
- Latching/flag: On rxDataRdySet, rxDataOut <= rxDataIn and rxDataRdy <= 1 (sticky until cleared).
- Clears: Any of rx1DataRdyClr..rx4DataRdyClr pulses will clear rxDataRdy.
- Precedence: If rxDataRdySet and any clear assert in the same cycle, set wins (rxDataRdy remains 1 and rxDataOut updates).
- Consumer guidance: Coordinate clears; clear rxDataRdy after consuming rxDataOut to avoid ambiguity and enable subsequent bytes.

Reset
- Synchronous, active-high: txDataOut=0x00, txDataFull=0, rxDataOut=0x00, rxDataRdy=0.

Interaction with readWriteSPIWireData
- TX: Engine samples txDataOut when txDataFull=1 and pulses txDataFullClr when starting to shift that byte.
- RX: After 8 SCK bits, engine drives rxDataIn and pulses rxDataRdySet; staging register latches and asserts rxDataRdy until cleared.

Typical integration (spiMaster)
- Likely TX priority mapping: tx1=sendCmd, tx2=readWriteSDBlock, tx3=initSD, tx4=spiCtrl direct-access. RX clearers: sendCmd, readWriteSDBlock, spiCtrl; initSD typically leaves clear deasserted.
- Bus-visible direct access: ctrlStsRegBI exposes the most recent RX byte at address 0x06 for software reads in direct-access mode.

Constraints and usage
- Observe txDataFull before asserting txXDataWEn to avoid overwriting.
- Only one producer should write per cycle; fixed priority can starve lower-priority sources under contention.
- Do not hold RX clear high continuously or ready indications may be masked.
- Entire module operates in spiSysClk; ctrlStsRegBI bridges to the bus domain for software access to the RX byte.

-- SPI Wire-Level Shifter --

Wire-level SPI byte shifter that generates SCK/MOSI and samples MISO to transfer 8-bit words MSB-first.
- SPI timing: CPOL=0 (SCK idles low). MISO is sampled once per bit while SCK is low just before the rising edge; bit counter advances on the rising edge (mode-0-like). Fixed mode; no CPOL/CPHA configurability.
- Programmable speed: clkDelay[7:0] defines the SCK half-period. Effective fSCK ~ fclk / (2*(clkDelay+1)).
- Interfaces (spiSysClk domain):
- Upstream TX: txDataIn[7:0], txDataFull (byte available), txDataFullClr (1-cycle accept pulse), txDataEmpty (engine idle indicator).
- Downstream RX: rxDataOut[7:0], rxDataRdySet (1-cycle pulse at byte completion).
- SPI pins: spiDataOut (MOSI), spiDataIn (MISO), spiClkOut (SCK).
- Control: clk, rst, clkDelay[7:0].
- Operation:
- Idle: SCK=0, txDataEmpty=1; waits for txDataFull.
- Start: on txDataFull=1, latch txDataIn, pulse txDataFullClr, clear counters, deassert txDataEmpty.
- Bit cycle (8 bits): with SCK low, drive MOSI with current MSB and sample MISO; after clkDelay, raise SCK and increment bit count; after next half-period, lower SCK and continue.
- Byte complete: after 8 bits, present rxDataOut and pulse rxDataRdySet. If txDataFull is already 1, accept next byte immediately and continue with no inter-byte gap; else assert txDataEmpty and stop toggling SCK.
- Reset (synchronous): clears counters and shift registers; SCK=0, MOSI=0, rxDataRdySet=0; returns to Idle (txDataEmpty=1).
- Throughput: one byte per 2 x (clkDelay-controlled) half-cycles x 8 bits; supports continuous back-to-back streaming when txDataFull is maintained at byte boundaries.
- Integration: upstream source/sink is spiTxRxData (arbitrates multiple producers). Chip-select is managed outside this shifter. Continuous clock-only periods are achieved by sending 0xFF bytes from upstream. clkDelay may be forced slow during SD init; otherwise set from control register and synchronized into spiSysClk.
- Limitations: single SPI mode (CPOL=0), no native CS handling, clock toggles only while transferring bytes. All logic synchronous to spiSysClk.

-- FIFO Subsystems (TX/RX) --

FIFO Subsystems (TX/RX)
- Purpose and domains: Two asynchronous, dual-clock FIFOs bridge busClk (CPU/Wishbone) and spiSysClk (SPI engine). TX FIFO buffers bytes from software (busClk) for SPI transmission (spiSysClk). RX FIFO buffers bytes received in spiSysClk for software reads in busClk.
- Bus address map and ack policy:
- RX block 0x10..0x1F: 0x10=data pop/read (1-cycle wait-state on read), 0x12=count[15:8], 0x13=count[7:0], 0x14=control (write bit0=1 flush).
- TX block 0x20..0x2F: 0x20=data write (immediate ack on write; reads get 1-cycle delayed ack and return 0x00), 0x24=control (write bit0=1 flush).
- Only base reads at 0x10 and 0x20 insert a single wait-state; all other accesses ack immediately.
- TX FIFO behavior:
- Bus side (sm_TxfifoBI, busClk): Write to 0x20 enqueues busDataIn (fifoWEn asserted while bus write is qualified). Reads return 0x00. Write to 0x24 with bit0=1 issues force-empty.
- SPI side (consumer): Pulse fifoREn in spiSysClk to dequeue when not empty. numElementsInFifo available in SPI domain for pacing; TX occupancy not exposed to software.
- RX FIFO behavior:

- SPI side (producer): Assert fifoWEn in spiSysClk to enqueue when not full.
- Bus side (sm_RxfifoBI, busClk): Read 0x10 returns current element and asserts fifoREn (pop) in same cycle; BI inserts one wait-state to align with registered RAM read. Read 0x12/0x13 returns occupancy count. Write to 0x14 with bit0=1 issues force-empty.
- FIFO core (shared by TX/RX):
- Storage: Dual-clock simple dual-port RAM with 1-cycle registered read in the read domain.
- Pointers: Binary read/write pointers (ADDR_WIDTH+1 with MSB for wrap). Opposite pointer is single-flop synchronized into each domain.
- Flags and count: fifoFull (wrClk) when lower bits match and MSBs differ; fifoEmpty (rdClk) when synchronized write pointer equals read pointer; numElementsInFifo (rdClk) = zero-extended difference of synchronized write pointer minus local read pointer.
- Enables: fifoWEn is level-sensitive (writes each wrClk while high). fifoREn is edge-sensitive; generate a 0->1 pulse per pop and deassert between pops.
- Force-empty (flush):
- Triggered by control writes (RX 0x14 bit0, TX 0x24 bit0). Implemented via a toggle-based CDC; generates one-cycle pulses in both busClk and spiSysClk. Flush clears pointers, sets empty, clears full, and zeroes count; discard any previously latched output after flush.
- Timing and usage notes:
- RX base read has 1-cycle registered RAM latency; BI inserts one wait-state—hold strobe_i across the inserted wait. TX base read is permitted but returns 0x00 with a delayed ack.
- Software: Write TX data at 0x20; no TX occupancy register on bus. Read RX data at 0x10; poll count at 0x12/0x13 and avoid reading when empty. Use flush controls (0x24, 0x14) to abort/resync.
- SPI-side logic: Consume TX when not empty by pulsing fifoREn; produce RX when not full by asserting fifoWEn; use numElementsInFifo for pacing/end conditions.
- Limitations and cautions:
- CDC robustness: Single-flop binary pointer synchronization can cause transient flag/count glitches under high asynchrony; for hardened designs, use Gray-coded pointers with two-flop synchronizers.
- RX read-pop coupling: Reading 0x10 pops the FIFO; coordinate with count/empty to prevent unintended pops.
- Device specifics: Dual-port RAM read is registered; simultaneous read/write to the same address has device-dependent behavior. Initial power-up may create a spurious flush pulse depending on device initialization.

-- Acknowledge Timing Behavior --

ack_o is generated by spiMasterWishBoneBI and is synchronous to busClk with Wishbone-like semantics. Two timing modes exist:
- Immediate acknowledge (zero wait-state): ack_o asserts combinationally with strobe_i for all writes and for all reads except the FIFO base addresses. This includes control/status (0x00 window), non-base offsets in the RX/TX FIFO windows (0x10–0x1F and 0x20–0x2F), force-empty commands (e.g., 0x14, 0x24), and unmapped addresses (which return 0x00).
- One-cycle delayed acknowledge: applies only to reads of the RX FIFO data port at 0x10 and the TX FIFO data port at 0x20. Internally, ack_o = ack_immediate & ack_delayed, where ack_immediate is the combinational qualify of strobe_i and decode, and ack_delayed is strobe_i registered by one bus clock. The bus master must hold strobe_i across the inserted wait cycle; ack_o asserts on the second cycle.
Data validity: For immediate reads, read data is valid in the same cycle ack_o asserts. For delayed reads (0x10, 0x20), the returned byte is valid when ack_o asserts on the second cycle, aligning with the FIFO's 1-cycle read latency.
Independence from flow control: Bus ack timing is not gated by FIFO full/empty or by success of pop/push operations. Writes to TX FIFO data (0x20) and force-empty commands are always acknowledged immediately; software must enforce flow control via status paths.
Separation from internal handshakes: Internal req/ready pulses (e.g., txDataFullClr, rxDataRdySet,

sendCmdRdy) are functional acknowledgments within the spiSysClk domain and do not affect ack_o timing.
Reset/CDC notes: The delayed-ack flip-flop is not explicitly reset and only influences cycles where strobe_i is asserted. CDC toggle-based force-empty commands are bus-acked immediately; flush completion should be inferred from FIFO status, not ack_o.
Master guidance: Hold strobe_i until ack_o is observed, especially for delayed reads; sample read data only when ack_o is high. Do not interpret ack_o as indicating FIFO emptiness, peripheral readiness, or transaction completion.

OPERATION

-- Transaction Types and Sequencing --

- Transaction type encoding (spiTransType)
- 00: Single SPI byte transfer (raw SPI)
- 01: SD card initialization over SPI
- 10: SD single-block read (512 bytes)
- 11: SD single-block write (512 bytes)

- Software sequencing to start a transaction (busClk domain)
- Optional: program SPI clock delay at 0x0B
- For SD block ops (10/11): write 32-bit SD block address bytes to 0x07..0x0A
- For raw SPI (00): write TX byte to 0x06
- Write transaction type to 0x02
- Start: write 0x03 bit0=1 (one-cycle start pulse into spiSysClk)
- Completion: poll 0x04 bit0 (busy) until 0; read 0x05 for error flags; for raw SPI read RX byte at 0x06

- Top-level arbitration and timing (spiSysClk domain)
- spiCtrl samples the start pulse only when idle; it latches spiTransType, asserts a one-cycle request to the selected submodule, and sets busy (0x04 bit0)
- Exactly one transaction in flight; further starts during busy are ignored (software must gate on 0x04=0)
- Chip select: spiCS_n is the AND of submodule CS signals; any active submodule can drive CS low
- SPI clocking uses programmable clkDelay (0x0B); initSD temporarily forces a slow divider during SD init and restores normal afterwards

- Type 00 (raw SPI byte):
- spiCtrl drives CS low, pulses txDataWen to load 0x06 into the byte shifter
- 8 bits are shifted MSB-first while sampling MISO; completion sets rxDataRdy
- spiCtrl pulses rxDataRdyClr, deasserts CS, clears busy; received byte is readable at 0x06

- Type 01 (SD init):
- Force slow SPI; clock ~160 bytes of 0xFF with CS high
- Assert CS low; send CMD0 (0x40, CRC 0x95) and expect R1=0x01 (idle), retry up to 0xFF
- Send CMD1 (0x41, CRC 0xFF) and expect R1=0x00 (ready), retry up to 0xFF with pacing delays
- Release CS; restore normal SPI clock; busy clears; errors reported at 0x05 (01=CMD0 fail, 10=CMD1 fail)

- Type 10 (SD single-block read):

- Assert CS low; send CMD17 (0x51 + address, CRC 0xFF); require R1=0x00 else readError=01
- Poll 0xFF until data-start token 0xFE; timeout readError=10
- Stream 512 data bytes into RX FIFO (read at 0x10 base); ignore two CRC bytes
- Deassert CS; busy clears; byte count readable via 0x12/0x13; software must drain 512 bytes

- Type 11 (SD single-block write):
- Assert CS low; send CMD24 (0x58 + address, CRC 0xFF); require R1=0x00 else writeError=01
- Send two 0xFF gap bytes, then data-start token 0xFE
- Stream 512 bytes from TX FIFO (write at 0x20 base), then two CRC bytes (0xFF, 0xFF)
- Read data-response; accept only 0x05 else writeError=10
- Busy-wait by sending 0xFF until DO!=0; timeout writeError=11; deassert CS; busy clears
- Ensure TX FIFO holds 512 bytes before start to avoid stalls; flush via 0x24 bit0 if needed

- Command framing and polling (used by initSD and block ops)
- sendCmd frames: 0xFF preamble, cmd, 4 data bytes, checksum; waits TX empty, then polls by sending 0xFF until MSB of response clears or a retry limit (512 polls) triggers timeout

- Completion and error reporting
- Transaction completes when 0x04 bit0 returns to 0
- Errors aggregated at 0x05: SDInitError, SDReadError, SDWriteError per codes above

- CS ownership and clocking
- Raw SPI: spiCtrl owns CS for the single-byte transfer
- SD flows: initSD/readWriteSDBlock own CS for their duration; sendCmd assumes caller manages CS
- clkDelay (0x0B) applies globally; initSD overrides only during init

-- Raw SPI Byte Transfer --

Raw SPI Byte Transfer performs a single MOSI/MISO byte exchange under spiMaster using the byte engine (spiTxRxData + readWriteSPIWireData), dispatched by spiCtrl when transaction type=00.

Registers (bus domain)
- 0x0B: clkDelay (SPI SCK half-period)
- 0x06: Direct-access TX/RX byte
- 0x02: spiTransType (set to 0x00 for raw byte)
- 0x03: spiTransCtrl (bit0 start pulse)
- 0x04: spiTransSts (bit0 in-progress)
- 0x05: error flags (SD-specific; not used by raw byte path)

Software sequence
1) Optional: write clkDelay at 0x0B.
2) Write TX byte to 0x06.
3) Write 0x00 to 0x02.
4) Start: write bit0=1 at 0x03.
5) Poll 0x04 bit0 until it clears.
6) Read RX byte from 0x06.

Functional behavior
- Start and type cross from busClk to spiSysClk via ctrlStsRegBI, producing a one-cycle spiTransCtrl pulse.
- spiCtrl (type=00) sets spiTransSts=1, drives spiCS_n low, and pulses txDataWen to stage the TX byte into spiTxRxData.

- spiTxRxData asserts txDataFull; readWriteSPIWireData immediately clears it (txDataFullClr) and begins shifting.
- Wire shift: SCK idles low; half-period from clkDelay. MOSI updates during SCK low; MISO is sampled once per bit during the low phase; bit counter advances on SCK rising edges. After 8 bits, rxDataRdySet pulses with the received byte.
- RX capture: spiTxRxData latches rxDataIn, asserts rxDataRdy; spiCtrl pulses rxDataRdyClr, deasserts spiCS_n high, clears spiTransSts, returns to idle. ctrlStsRegBI mirrors the RX byte to 0x06 for bus reads.

Chip select policy
- Top-level spiCS_n = spiCS_nFromInitSD & spiCS_nFromRWSDBlock & spiCS_nFromSpiCtrl. In raw byte mode, only spiCtrl drives CS; CS is low for the single-byte duration and returns high on completion.

Throughput and boundaries
- Exactly one byte per start pulse; CS toggles high between bytes. For multi-byte frames with continuous CS low, use higher-level flows (e.g., sendCmd or SD block).

Timing
- Programmable SCK half-period via 0x0B; SCK idle low. Data phasing as described above; verify target device compatibility.

Status and errors
- 0x04 bit0 indicates in-progress; set on start, clear on completion. No raw-byte error flags; 0x05 pertains to SD submodules only.

Reset
- Bus or external reset synchronizes into spiSysClk via ctrlStsRegBI, returning spiCtrl to idle, CS high, clearing byte-engine flags and transaction status.

Constraints
- Single-byte staging (no FIFO). Do not rewrite 0x06 after start; relies on txDataFull/txDataFullClr. TX byte must be written before start; RX is mirrored back for read. Control/status accesses in this window are immediate-ack.

Modules involved
- ctrlStsRegBI (registers + CDC), spiCtrl (dispatcher + CS), spiTxRxData (byte staging), readWriteSPIWireData (wire shifter + SCK gen), spiMasterWishBoneBI (decode + acks).

Handshakes
- txDataWen → txDataFull → txDataFullClr; rxDataRdySet → rxDataRdy → rxDataRdyClr.

-- SD Initialization Sequence --

SD Initialization Sequence (legacy CMD0/CMD1)

- Purpose: Place SD card into SPI mode and exit idle using CMD0 then CMD1.

- Software trigger and interface
- Program transaction type = 0x01 at 0x02, then write 0x03[0]=1 to start.
- spiCtrl latches request, sets spiTransSts=1 (0x04 nonzero), pulses SDInitReq.
- Software polls 0x04 until 0 (done), then reads 0x05 for initError: 0x00=OK, 0x01=CMD0 failed,

0x02=CMD1 failed.
- SPI clock delay register at 0x0B is used outside init; init overrides it.

- SPI clock and chip-select behavior
- During init, spiClkDelayOut is forced to 0x3b (slow). After completion, restored to spiClkDelayIn (from 0x0B).
- spiCS_n is the logical AND of submodule CS requests; initSD drives CS low only during each command; CS idles high otherwise (including pre-clocking).

- Pre-clocking (with CS high)
- Transmit 0xA0 bytes of 0xFF (160 bytes = 1280 SCK edges) via TX path.
- Respect txDataFull/txDataEmpty; wait for TX drain (txDataEmpty=1) before proceeding.

- Command transport (sendCmd submodule)
- Frame: one 0xFF preamble, then cmdByte, 4 argument bytes (MSB first), and CRC byte.
- Poll for R1 by sending 0xFF until a non-busy response (R1[7]=0) is received or retry count reaches 512 (respTout).
- CRCs: 0x95 for CMD0; 0xFF for CMD1 (CRC off assumed after init).
- sendCmd manages RX-ready clearing; initSD does not assert rxDataRdyClr.

- CMD0 loop (enter SPI mode; expect R1=0x01)
- Assert CS low; issue CMD0: cmd=0x40, arg=0x00000000, CRC=0x95 via sendCmd.
- Wait for sendCmdRdy; if respByte != 0x01, retry (deassert CS between attempts) until loopCnt reaches 0xFF.
- On success, deassert CS and continue. On failure, set initError=0x01 and exit.

- CMD1 loop (leave idle; expect R1=0x00)
- Assert CS low; issue CMD1: cmd=0x41, arg=0x00000000, CRC=0xFF via sendCmd.
- Insert inter-attempt spacing: nested delays with delCnt2 up to 0xFF for each increment of delCnt1 up to 0x177.
- Wait for sendCmdRdy; if respByte != 0x00, retry (deassert CS between attempts) until loopCnt reaches 0xFF.
- On success, deassert CS and proceed. On failure, set initError=0x02 and exit.

- Completion
- Restore spiClkDelayOut to software-programmed value (0x0B).
- Raise SDInitRdy=1; spiCtrl observes ready and clears spiTransSts (0x04 -> 0).

- Counters and timeouts
- Pre-clocking count: 0xA0 bytes.
- CMD0/CMD1 retry loopCnt: up to 0xFF attempts each.
- CMD1 spacing delays: delCnt1 up to 0x177; delCnt2 up to 0xFF.
- sendCmd busy-poll retry: 512 polls (respTout) when R1[7] remains 1; if rxDataRdy never asserts, sendCmd can wait indefinitely (respTout covers only repeated busy responses).

- Integration requirements
- Only one source may drive sendCmdReq/cmd/arg/CRC at a time; initSD must have exclusive access during init to avoid bus OR conflicts.

- Limitations and compatibility
- This legacy flow omits CMD8 and ACMD41 (CMD55+ACMD41). Many SD v2.0/SDHC cards require those and may not initialize with CMD0/CMD1 alone.
- On success (initError=0), subsequent block operations (e.g., CMD17/CMD24) proceed using the

restored SPI clock.

-- SD Block Read Operation --

Implements a single-block SD card read in SPI mode (CMD17), transferring 512 bytes into a bus-visible RX FIFO with chip-select control, SPI clocking, command framing, and status/error reporting. Prerequisites: initialize the card via initSD (legacy CMD0/CMD1); software must match address format to card type (SDSC byte address vs SDHC block address). Registers and addresses: program SPI half-period at 0x0B (clkDelay); write 32-bit address to 0x07..0x0A (little-endian by register address); select transaction type at 0x02 with type=0b10 for block read; start at 0x03 bit0=1; poll 0x04 until 0 (complete); read data from RX FIFO at 0x10 (one-cycle wait-state per read); FIFO count at 0x12/0x13; force-empty via 0x14 bit0=1; errors at 0x05. Operation flow: spiCtrl samples start/type when idle, asserts readWriteSDBlockReq with selector 2'b10; readWriteSDBlock holds chip select low for the full operation; top-level spiCS_n is the AND of CS requests from initSD, readWriteSDBlock, and spiCtrl. Command phase: readWriteSDBlock drives sendCmd with CMD17 (0x51), the 32-bit address, CRC=0xFF, and preamble 0xFF; sendCmd transmits cmd and polls for R1 by clocking 0xFF until respByte[7]==0 or timeout (~512 polls). If R1!=0 or the response times out, readError=01 and the operation aborts. Token wait: after an ACKed command, readWriteSDBlock clocks 0xFF and samples MISO until start token 0xFE is observed; if not seen within timeout (~0x048 polls), readError=10 and abort. Data transfer: upon 0xFE, 512 payload bytes are clocked (send 0xFF per byte), captured on rxDataRdy pulses, and written to sm_RxFifo via rxFifoWen; a 9-bit loop counter ensures exactly 512 bytes; two trailing CRC bytes are clocked and ignored; CS deasserts once TX drains and readWriteSDBlockRdy returns high; spiCtrl clears busy (0x04). SPI timing/datapath: mode 0 (CPOL=0, CPHA=0); SCK idles low, data changes during SCK low, bit count on rising edge; clkDelay at 0x0B sets SCK half-period (initSD may override during init, block read uses programmed value); readWriteSPIWireData shifts MSB-first, samples MISO once per bit during SCK low, asserts rxDataRdySet every 8 bits; spiTxRxData arbitrates TX sources and latches RX bytes; continuous streaming supported when txDataFull is maintained. RX FIFO/bus: reading 0x10 both returns current output and pops one byte; the bus front-end inserts a single wait-state—keep strobe asserted; once primed, back-to-back reads drain one byte per bus cycle; use 0x12/0x13 to avoid underflow; force-empty at 0x14 generates synchronized one-shot flush pulses in both clock domains. Status/errors: 0x04 indicates in-progress (set on start, cleared on completion); 0x05 readError codes—00: no error, 01: CMD17 failed or timed out, 10: start token timeout; latched per operation, cleared by new start or reset. CDC/timing caveats: RX/TX FIFOs use simple dual-clock CDC with single-flop pointer sync (binary counters); adequate for moderate skew, avoid extreme clock ratios. Software-visible sequence: (1) initialize card; (2) set clkDelay at 0x0B; (3) write address at 0x07..0x0A; (4) set type=0b10 at 0x02; (5) write start at 0x03 bit0=1; (6) poll 0x04 to 0; (7) read 512 bytes from 0x10 (optionally pace via 0x12/0x13 and flush via 0x14); (8) check 0x05 for errors.

-- SD Block Write Operation --

Purpose: Execute one 512-byte SD block write in SPI mode, end-to-end from software to wire, with clear status/error reporting.

Preconditions: Card initialized in SPI mode; software uses correct address mode (byte vs block); CRC disabled; TX application FIFO must contain ≥512 bytes before start; verify spiTransType mapping in RTL (readWriteSDBlock expects Req[1:0]=01 for write, 10 for read; other texts differ).

Software (busClk domain via ctrlStsRegBI/spiMasterWishBoneBI):
- Optional flush: write 0x24 bit0=1 to force-empty TX FIFO (sm_TxFifo) coherently across domains.
- Enqueue data: write exactly 512 bytes to TX FIFO at window base 0x20; writes ack immediately.
- Program address: write 32-bit SD address across 0x07..0x0A (little-endian by address).
- Optional timing: set SPI clock half-period at 0x0B.

- Transaction type: program 0x02 to select SD block write (confirm RTL encoding).
- Start: write 0x03 bit0=1.
- Monitor: poll 0x04 (transaction status) until it returns 0; read 0x05 for error flags (00=success; 01=CMD24 fail; 10=bad/no data-response; 11=busy timeout).

On-wire (spiSysClk domain via readWriteSDBlock/sendCmd/byte engine):
- Chip-select: readWriteSDBlock holds spiCS_n low for entire transaction; top-level CS is AND of submodule CS_n requests.
- Command phase: send preamble 0xFF, CMD24 (0x58), 4-byte address, checksum (CRC=0xFF); poll for R1 response and expect 0x00; timeout/nonzero sets writeError=01 and abort.
- Data phase: gap with two 0xFF; send start data token 0xFE; stream 512 payload bytes by draining sm_TxFifo (assert txFifoRen, write to byte engine when txDataFull=0); send two 0xFF CRC bytes (ignored).
- Response/busy: read data-response token; accept only if low 5 bits == 0x05; else/timeout set writeError=10; then keep sending 0xFF and sample MISO until nonzero (card not busy); on timeout set writeError=11.
- Exit: after TX path drains (txDataEmpty=1), deassert CS, assert readWriteSDBlockRdy=1; spiCtrl returns to idle and clears transaction status.

Clocks/CDC/Timing:
- SPI timing: CPOL=0; MOSI stable while SCK low; MISO sampled once per bit during low phase; SCK half-period programmable at 0x0B.
- CDC: sm_TxFifo bridges busClk→spiSysClk; force-empty pulse ensures coherent flush; Wishbone BI adds one-cycle wait-state only for reads at FIFO windows (0x10/0x20); writes to 0x20 ack immediately.

Errors (0x05 writeError): 01=CMD24 timeout/R1!=0; 10=no/invalid data-response token; 11=busy-release timeout; 00=success.

Notes:
- Ensure FIFO does not underflow during payload or stream may stall and trigger timeouts.
- sendCmd response timeout uses a local counter (~512 polls); busy-release timeout uses a shorter counter; exact values are RTL constants.
- Status at 0x04 is set on start and auto-clears when spiCtrl idles after readWriteSDBlockRdy.
- Only one submodule may drive CS low; arbitration is spiCS_n = spiCS_nFromInitSD & spiCS_nFromRWSDBlock & spiCS_nFromSpiCtrl.

-- Command Framing and Response Polling --

sendCmd is a micro-FSM that frames SD SPI-mode commands and polls for the R1 response. A transaction starts when sendCmdReq is asserted while sendCmdRdy=1; the caller must hold chip select low for the entire sequence. Framing emits exactly seven bytes to the TX path, each only when txDataFull=0: 0xFF preamble, cmdByte, dataByte1, dataByte2, dataByte3, dataByte4, and checkSumByte (CRC). The byte wires (cmdByte, dataByte1..4, checkSumByte) are the per-clock OR of two producers; the system must ensure only one producer drives them and that they remain stable from before sendCmdReq through the framing phase. After the last byte is written, sendCmd waits for txDataEmpty=1 to ensure the shifter has drained before polling. Polling cycles are: write 0xFF to TX to generate SCKs; pulse rxDataRdyClr; wait for rxDataRdy==1; capture rxDataIn to respByte; evaluate respByte[7] (MSB): 1 indicates card busy (continue polling), 0 indicates a valid non-busy R1 (complete). A 10-bit counter limits busy retries; when it reaches 512 evaluations, the FSM completes with respTout=1. There is no timeout for missing rxDataRdy; the FSM can wait indefinitely in the RX-wait state. On completion, sendCmdRdy returns high; respByte contains the last observed response; respTout flags a busy-retry timeout. Integration and constraints: sendCmd does not control CS or add gap bytes; callers (initSD, readWriteSDBlock) must manage spiCS_n and any extra 0xFF clocks outside sendCmd. CRC/checkSumByte is supplied by the caller (e.g., 0x95 for CMD0, 0xFF for typical SPI-mode commands after init). TX/RX coordination relies on the byte engine: txDataEmpty denotes idle, rxDataRdy is set per received byte by the wire shifter; sendCmd clears rxDataRdy prior to each poll to avoid stale data.

-- FIFO Operation and Backpressure --

Two dual-clock FIFOs provide flow control between busClk and spiSysClk: sm_TxFifo (bus producer → SPI consumer) and sm_RxFifo (SPI producer → bus consumer). FIFO write enable is level-sensitive (push one entry per wrClk while fifoWEn=1; assert only when fifoFull=0). FIFO read enable is rising-edge-sensitive (pop one entry per low→high fifoREn; assert only when fifoEmpty=0). Read port has one rdClk latency; sm_RxFifo's bus pop aligns via the BI and a fixed one-cycle read wait-state at 0x10 and 0x20 (strobe must be held; ack on second cycle).

Backpressure and handshakes: SPI wire path is a single-byte staging register, not a FIFO. Present a TX byte only when txDataFull=0; the shifter asserts txDataFullClr when it consumes the byte. txDataEmpty indicates the shifter is idle; continuous streaming requires keeping the staging full at each byte boundary. RX wire-level uses rxDataRdy/rxDataRdyClr at byte boundaries; capture only when rxDataRdy=1.

TX chain: software writes 0x20 (addr 0 in sm_TxfifoBI) when sm_TxFifo not full; the SD write controller pops when sm_TxFifo not empty and only forwards to the wire when txDataFull=0. RX chain: the SD read controller writes into sm_RxFifo when not full and rxDataRdy=1; software reads 0x10 to return current output and pop for the next element, pacing via count at 0x12/0x13 and/or fifoEmpty.

Force-empty (flush): 0x24 bit[0] for TX, 0x14 bit[0] for RX. A toggle-based CDC generates one-cycle pulses in both busClk and spiSysClk, coherently clearing storage, pointers, flags, and count; space successive commands to avoid coalescing, and re-prime streams after flush.

Occupancy and visibility: sm_RxFifo exposes count at 0x12 (high) and 0x13 (low); sm_TxFifo count is not memory-mapped (flags/counters are available at top level). Pointer synchronization uses single-flop CDC and binary counters; full/empty and count are eventually consistent across domains. For higher robustness at extreme skew, consider Gray-coded pointers with two-stage synchronizers in future revisions.

-- SPI Clock Programming and Rate Changes --

Defines how the SPI master clock (SCK) rate is programmed, generated, and temporarily overridden during SD initialization.

- Programming interface:
- An 8-bit half-period parameter (clkDelay) is exposed at ctrlStsRegBI address 0x0B (read/write).
- On reset, clkDelay defaults to 0x00 in the bus domain.
- Writes are CDC-synchronized into the spiSysClk domain and take effect immediately in the clock generator.

- Clock generation:
- The wire-level shifter (readWriteSPIWireData) drives SCK with CPOL=0.
- Low and high phases are each held for clkDelay-controlled durations using an equality-based counter in the spiSysClk domain (implementation-specific off-by-one may apply).
- Practical guidance: program clkDelay >= 1 to ensure a valid, nonzero half-period. Effective f_SCK is derived from f_spiSysClk and clkDelay; use empirical calibration if exact frequency is required.

- Rate change behavior:
- Changes to 0x0B affect the next half-period immediately; avoid updating while a transfer is active to prevent mid-bit distortion. Update only when the SPI path is idle or between bytes (e.g., when transmit FIFO/engine is empty).

- SD initialization override:
- While SD initialization is active, hardware forces a slow SCK by overriding the effective delay to 0x3B.
- When initialization completes, the effective delay automatically reverts to the base value programmed at 0x0B; software intervention is not required.
- The 0x0B register always reflects the base value; there is no bus-visible register for the current effective delay during the override.

- System-level considerations:
- Higher-level FSM timeouts are count/poll based; changing clkDelay alters wall-clock durations but not internal count limits.
- All non-init transactions (raw SPI, sendCmd, single-block read/write) use the currently programmed base delay and do not override the rate.

-- Error Handling and Status Reporting --

Scope: Defines how the design reports status and errors to software, what conditions generate them, how they clear, and available recovery mechanisms.

Software-visible registers
- 0x04 Transaction Status (busy): Mirrors spiCtrl.spiTransSts. Asserts when a transaction is launched (via control at 0x03) and auto-clears when the SPI side returns to idle. Busy remains asserted if upstream flows stall.
- 0x05 Error Flags: {reserved[7:3], SDWriteError[2], SDReadError[1], SDInitError[0]}. Each flag is a level reflection of upstream 2-bit error codes; any non-zero code drives its flag to 1. There is no write-to-clear; flags clear when the originating submodule reports success (code returns to 00) or on global reset.
- 0x06 Direct-access byte: Read returns most recent RX byte; write enqueues a TX byte for raw SPI. No separate error flag for direct-access.
- FIFO-related: RX FIFO base read at 0x10 (1-cycle wait-state), count at 0x12/0x13, flush at 0x14 bit[0]=1. TX FIFO base write at 0x20, flush at 0x24 bit[0]=1. No standard TX occupancy/full register in the bus map.

Error sources and mapping
- SD initialization (initSD): initError[1:0] encodes failures (01=CMD0 failed, 10=CMD1 failed). ctrlStsRegBI maps any non-zero initError to SDInitError=1 (0x05[0]). Clears on a successful init or reset.
- SD block read/write (readWriteSDBlock):
- readError[1:0] (01=command failed; 10=start token 0xFE not seen) → SDReadError=1 (0x05[1]).
- writeError[1:0] (01=command failed; 10=no/invalid data-response; 11=busy hold too long) → SDWriteError=1 (0x05[2]).
- Both clear on next successful operation or reset.
- Command sender (sendCmd): Sets respTout when busy-poll retries exceed 512; provides respByte. These are consumed by initSD/readWriteSDBlock to form their error codes and are not directly software-visible. There is no timeout for "no RX data ever arrives" in WAIT_RX; upstream FSMs must ensure RX servicing, otherwise transactions can stall and busy may remain set.

FIFO status philosophy
- RX FIFO: No explicit overflow/underflow error flags. Software should read 0x12/0x13 to gate reads; reading when empty does not pop and may return stale data. A 1-cycle wait-state is inserted on reads at 0x10.
- TX FIFO: No overflow error flag and no standard bus-readable occupancy/full status. Software must pace writes to avoid overruns, using higher-level flow-control if available.

Bus interface behavior
- Read acks are immediate except for FIFO base addresses (0x10, 0x20), where a single wait-state is inserted. This latency is normal and not an error.
- Unmapped reads return 0x00 without an error indication.

Recovery mechanisms
- Global reset (0x01 bit[0]=1): Issues a synchronized reset across busClk and spiSysClk. Clears busy, deasserts CS, resets internal error/status in SPI domain to defaults (error codes → 0). SD address/config registers retain prior values unless software rewrites.
- FIFO flush: RX at 0x14 bit[0]=1, TX at 0x24 bit[0]=1. Generates synchronized one-shot pulses in both clock domains to force-empty contents and status, providing non-destructive recovery without full reset.

Operational expectations and limitations
- Error flags at 0x05 are live mirrors of submodule state; no write-to-clear.
- No FIFO overflow/underflow detection is reported; software must gate operations and use flush/reset for recovery.
- Potential synchronizer artifact: a spurious flush pulse at power-up is possible on the SPI side if the flush synchronizer is not explicitly reset.
- Chip-select arbitration is deterministic (logical AND of submodule CS_n). Simultaneous assertions are by design and are not treated as errors.

-- Reset and Force-Empty Behavior --

Reset sources and distribution:
- Sources: external rstFromWire and software reset via ctrlStsRegBI write at 0x01 bit0=1.
- Shaping: ctrlStsRegBI stretches reset in busClk with a 6-bit shift register (rstSyncToBusClkOut), then double-synchronizes it into spiSysClk (rstSyncToSpiClkOut) for safe SPI-domain reset.

Top-level effects on reset:
- Chip-select defaults high (spiCS_n=1); all CS request lines inactive; request strobes low.
- ctrlStsRegBI clears transaction status and start strobes; spiClkDelay cleared; SDAddr is not reset and must be reprogrammed.

Per-module reset behavior:
- ctrlStsRegBI: clears spiTransTypeSTB, spiTransCtrlSTB, spiTransStatusSTB, spiDirectAccessTxDataSTB, and spiClkDelay; start pulses and reset are CDC-safe.
- spiCtrl (dispatcher): enters reset then Idle; spiCS_n high, request strobes low, rxDataRdyClr low, spiTransSts cleared.
- initSD: state/output clear; returns to Idle with SDInitRdy=1; spiClkDelayOut follows reprogrammed value.
- readWriteSDBlock: returns to safe defaults, then Idle with ready=1 and spiCS_n=1 before accepting new requests.
- sendCmd: synchronous reset to INIT then IDLE; txDataWen=0, rxDataRdyClr=0, respByte=0, respTout=0, timeOutCnt=0; sendCmdRdy=1 in IDLE.
- spiTxRxData: clears TX/RX bytes and flags (txDataFull=0, rxDataRdy=0).

- readWriteSPIWireData: clears counters/shift regs; SCK idles low, MOSI=0; rxDataRdySet=0; txDataEmpty=1 in Idle.
- sm_TxFifo / sm_RxFifo (wr/rd domains): synchronous resets clear write/read pointers; fifoEmpty=1, fifoFull=0, element counts=0; RAM contents unchanged but become inaccessible due to pointer reset.

Force-empty (flush) mechanism:
- TX FIFO flush command: write 0x24 bit0=1 (window 0x20, offset 0x04).
- RX FIFO flush command: write 0x14 bit0=1 (window 0x10, offset 0x04).
- Generation: BI wrappers edge-detect the bus write to produce a one-cycle pulse in busClk and, via toggle-based CDC, a matching one-cycle pulse in spiSysClk. Synchronizer flops in spiSysClk are not explicitly reset; a spurious pulse at power-up is possible.
- Effect: treated as a soft reset of FIFO pointers/flags in both domains; fifoEmpty asserted, fifoFull deasserted, element counts cleared; RAM contents not cleared.

System-level interactions and requirements:
- Reset or force-empty aborts any in-flight FIFO transactions at the next local clock edge; software/hardware must observe fifoEmpty/fifoFull and counts and re-prime streaming after a flush.
- After reset, software must reprogram SDAddr and all transaction parameters before starting operations; ctrlStsRegBI transaction status (0x04) is cleared.
- sendCmd and wire-level shifter flags are cleared, preventing spurious polling/transfers post-reset; CS remains high until a new transaction is issued.

Timing and CDC cautions:
- Force-empty events should be spaced by several spiSysClk cycles to ensure distinct toggle sampling and avoid pulse coalescing.
- sm_fifoRTL uses single-stage binary pointer synchronization (not Gray/two-flop); exercise caution for truly asynchronous or high-skew clocking.
- Read enable is edge-sensitive; write enable is level-sensitive; read-side occupancy computed in rdClk and clears on reset/flush.
- Wishbone BI has no explicit reset; FIFO read windows include a one-cycle alignment wait and are unaffected by reset/flush.

-- Back-to-Back Streaming Behavior --

Back-to-back streaming transfers bytes on MOSI/MISO without idle SCK gaps whenever the next byte is available at each 8-bit boundary. Streaming is governed by the TX staging flag (txDataFull/txDataEmpty), RX ready (rxDataRdy), FIFO status, and the programmed clock divider (clkDelay), and applies uniformly to SPI wire-level shifts, SD single-block reads/writes, command framing, and initialization clocks.

Wire-level SPI shifter (readWriteSPIWireData):
- On completion of each 8-bit transfer, if txDataFull==1, the shifter immediately reloads txDataIn, asserts txDataFullClr, pulses rxDataRdySet for the completed byte, and re-enters the low-phase state to keep clocking with no Idle gap.
- If txDataFull==0 at the boundary, streaming pauses; the shifter idles (txDataEmpty=1) until a new byte is staged.

TX staging and producer timing:
- spiTxRxData is a single-byte staging register. Producers must present the next byte in the cycle(s) after txDataFullClr to keep txDataFull asserted by the next boundary.
- txDataFullClr overrides a concurrent set; producers should avoid writing in the same cycle as clr and instead write on subsequent cycles to maintain continuous acceptance.

- Multiple TX sources are prioritized (tx1 > tx2 > tx3 > tx4). Only one source must drive during streaming to avoid contention; sendCmd sources use bitwise OR and must be mutually exclusive and stable per byte.

SD single-block streaming (readWriteSDBlock):
- Write (CMD24): After the 0xFE token, bytes are fed from sm_TxFifo into TX staging whenever txDataFull==0, producing contiguous MOSI data for 512 bytes; CRC bytes and busy-polling follow without gaps.
- Read (CMD17): 0xFF is clocked continuously on TX while 512 RX bytes are captured on each rxDataRdy; two CRC bytes are clocked contiguously after data.
- A 9-bit wrap counter (loopCnt) gates 512-byte loops without per-byte compares to sustain throughput.

Command framing and polling (sendCmd):
- Preamble, command, argument, and checksum bytes emit back-to-back, each gated by txDataFull; the frame drains until txDataEmpty.
- Response polling sends a contiguous stream of 0xFF bytes while sampling RX until the target responds.

Initialization clocks (initSD):
- Bursts 0xFF pre-clocks into sm_TxFifo until full, waits for drain (txDataEmpty), then issues CMD0/CMD1. SCK remains contiguous at a reduced clkDelay during preamble and command phases.

FIFOs and bus-side streaming:
- sm_TxFifo: Bus writes to 0x20 base ack immediately; software can stream TX bytes each bus cycle until fifoFull. The SPI side must pulse fifoREn per element to sustain drain.
- sm_RxFifo: Bus reads of 0x10 base insert a one-cycle wait; once primed, steady-state draining is ~1 element per two bus cycles. Software should plan loops around the enforced wait.
- Force-empty pulses in either domain immediately interrupt any ongoing stream and clear occupancy; space force-empty commands by several spiSysClk cycles to avoid coalescing.

Flow control and backpressure:
- Streaming pace is set by txDataFull/txDataEmpty and FIFO full/empty; the shifter only stalls at byte boundaries when no next byte is ready.
- RX throughput is governed by rxDataRdy; consumers must service and clear per-byte to avoid stalls.

Chip-select continuity:
- SD block operations hold CS asserted across the full 512-byte data stream, CRC, and busy phases.
- Single-byte and framed command transfers assert CS only for their respective windows; CS arbitration is active-low AND across submodules, so only one streaming source should assert CS at a time.

Programmable rate and CDC caveats:
- clkDelay defines the SCK half-period; initSD uses a slow divider (e.g., 0x3b), then normal flows use the programmed rate.
- sm_fifoRTL uses single-flop binary pointer synchronization; full/empty/count may lag by a few cycles under harsh async ratios. Do not rely on instantaneous status for boundary decisions; design software/logic to tolerate minor status latency.

-- Bus Read Wait-State Behavior --

Reads complete with zero wait-state everywhere except the FIFO data ports. For most addresses (0x00–0x0F control/status; 0x12/0x13 RX count and other non-base offsets in 0x10–0x1F; and unmapped), ack_o asserts in the same busClk cycle as strobe_i and data is returned immediately (unmapped reads return 0x00). Reads to the RX FIFO data port (0x10) and TX FIFO data port (0x20) insert a fixed one-cycle wait-state: ack_o is asserted on the second busClk cycle only if the master holds strobe_i across the wait. Data for these ports is valid only in the cycle ack_o is asserted; TX FIFO reads still return 0x00 by design. Streaming reads of 0x10: the first read incurs the wait, then one byte per cycle can be sustained if strobe_i remains asserted (ack each cycle after the first). Ack generation: for zero-wait regions, ack_o = strobe_i; for 0x10/0x20 reads, ack_o = (strobe_i delayed by one busClk) AND strobe_i. The wait-state does not depend on FIFO-empty; software must poll RX FIFO occupancy (0x12/0x13) before reading 0x10. The master must keep strobe_i asserted until ack_o; dropping it early aborts the access and may desynchronize FIFO side effects.

REGISTERS

-- Address Map Summary --

Address decoding (addr & 0xF0): 0x00–0x0F Control/Status, 0x10–0x1F RX FIFO, 0x20–0x2F TX FIFO; others unmapped (reads return 0x00). All registers are byte-wide. SDAddr is little-endian by address (0x07 LSB … 0x0A MSB). Wishbone ack: all accesses ack in 1 cycle except reads to 0x10 and 0x20, which ack on the next cycle.

Control/Status (0x00–0x0F):
- 0x00 RO: Module ID (0x12)
- 0x01 WO: Soft reset request (bit0=1)
- 0x02 RW: SPI transaction type [1:0]
- 0x03 WO/RB: Start strobe (write bit0=1; readback reflects strobe in same cycle)
- 0x04 RO: Transaction status (bit0=1 busy, clears on completion)
- 0x05 RO: SD error flags {writeError, readError, initError}
- 0x06 RW: Direct-access data (write TX byte, read RX byte)
- 0x07 RW: SDAddr[7:0]
- 0x08 RW: SDAddr[15:8]
- 0x09 RW: SDAddr[23:16]
- 0x0A RW: SDAddr[31:24]
- 0x0B RW: SPI clock half-period (clkDelay)

(Unlisted offsets in this block return 0x00.)

RX FIFO (0x10–0x1F):
- 0x10 RO: FIFO data (pop-on-read); read inserts 1-cycle wait-state; read only when count>0
- 0x12 RO: FIFO count [15:8]
- 0x13 RO: FIFO count [7:0]
- 0x14 WO: Force empty (bit0=1 one-shot flush)

(Unlisted offsets in this block return 0x00.)

TX FIFO (0x20–0x2F):
- 0x20 WO: FIFO data write (enqueue byte); reads return 0x00 and still insert 1-cycle wait-state
- 0x24 WO: Force empty (bit0=1 one-shot flush)

(Unlisted offsets in this block return 0x00.)

-- Control/Status Registers --

Control/Status Registers (byte-wide, 0x00..0x0F)
- Bus timing: reads/writes in this window are acknowledged immediately (no wait-states). All registers are 8-bit unless noted.
- Clock domains: busClk is the register interface clock; spiSysClk is the SPI/SD engine clock. Start/reset strobes are synchronized into spiSysClk; status/flags/data are synchronized back into busClk.
- Reset values: unless explicitly stated, software should not assume a defined reset value and must program fields before use.

Registers
- 0x00 (RO) Module ID/Version
- Fixed value 0x12.

- 0x01 (WO) Soft Reset
- Write bit[0]=1 to request a synchronized reset across busClk and spiSysClk.
- Other bits ignored. No readback (reads return 0x00).

- 0x02 (RW) SPI/SD Transaction Type
- Bits [1:0]: select operation (program before starting a transaction).
- 2’b00: single raw SPI byte
- 2’b01: SD card initialization
- 2’b10: SD single-block read
- 2’b11: SD single-block write
- Bits [7:2] read as 0.
- Sampled in spiSysClk before use.

- 0x03 (WO, edge-strobe; RO: immediate reflection only) Transaction Start
- Write bit[0]=1 to start the selected transaction; converted to a one-cycle pulse in spiSysClk.
- Read of 0x03 does not return a latched state; it only reflects the strobe level combinationally in the same bus cycle as a write, otherwise reads as 0x00. Use 0x04 to poll progress.

- 0x04 (RO) Transaction Status
- Bit[0]=1 while a transaction is in progress; clears to 0 when complete (driven by SPI/SD side).
- Bits [7:1]=0.
- Synchronized into busClk for safe polling.

- 0x05 (RO) SD Error Flags (summary)
- Bits [2:0]={bit2: SDWriteError, bit1: SDReadError, bit0: SDInitError}.
- Bits [7:3]=0.
- Read-only summary of submodule error indications; detailed error codes are internal and not exposed. Clear behavior is internal (not software-writable); flags are synchronized into busClk.

- 0x06 (RW) Direct-Access Data (for raw SPI byte transactions)
- Write: enqueue one TX byte to the SPI side (sampled in spiSysClk before use).
- Read: returns the last RX byte produced by the SPI side (synchronized into busClk).
- For transaction type 2’b00: write TX byte here, start via 0x03, poll 0x04 to 0, then read RX byte here.

- 0x07..0x0A (RW) SD Block Address (32-bit), little-endian by address
- 0x07: [7:0]; 0x08: [15:8]; 0x09: [23:16]; 0x0A: [31:24].
- Used by single-block read/write operations. Not explicitly reset; software must initialize after reset before issuing block operations.

- 0x0B (RW) SPI Clock Half-Period Delay (clkDelay)
- Controls the half-period of spiClkOut in the SPI bit-banging engine (readWriteSPIWireData).
- initSD may temporarily override/adjust clocking internally during card initialization.

- 0x0C..0x0F Reserved
- Reads return 0x00; writes are ignored.

Operational notes
- Use 0x03 only to initiate a transaction; do not rely on reads of 0x03. Poll 0x04 for busy/done.
- Program 0x02 (type), 0x07..0x0A (block address for SD block ops), and 0x0B (clock) before asserting start at 0x03.
- All CDC for start/reset (0x03/0x01), type (0x02), direct TX (0x06 write), status (0x04), direct RX (0x06 read), and errors (0x05) is handled internally.

-- Direct-Access Data Registers --

Direct-Access Data Registers provide a single-byte SPI transfer path that bypasses the TX/RX FIFOs. They reside in the control/status register block (address window 0x00..0x0F) and operate with the transaction-type, start-control, and status registers to perform one raw SPI byte.

Registers
- 0x06 Direct-Access Data: Write sets TX byte for next single-byte transfer; Read returns the most recent RX byte from the last completed single-byte transfer.
- 0x02 Transaction Type: Program 0x00 to select direct-access single-byte operation.
- 0x03 Transaction Control: Write bit0=1 to start; generates a clean one-cycle start pulse synchronized into the SPI domain.
- 0x04 Transaction Status: Read bit0=1 while busy; auto-clears to 0 at completion.
- 0x0B SPI Clock Delay: Read/write half-period of SCK for direct-access transfers.

Software sequence
1) Optional: Set SPI speed via 0x0B; reset via 0x01 bit0=1 if needed.
2) Write 0x02 = 0x00.
3) Write TX byte to 0x06.
4) Write 0x03 bit0=1 to start.
5) Poll 0x04 until bit0=0.
6) Read 0x06 for RX byte.

Behavior
- Immediate bus acks on 0x06, 0x02, 0x03, 0x04, 0x0B.
- RX byte at 0x06 persists until overwritten by a new transfer or reset; reads return 0x00 after reset until a transfer completes.
- Isolated from FIFOs: does not use TX/RX FIFO windows (0x20/0x10); use FIFOs for multi-byte streaming.
- Chip-select: spiCtrl drives CS low for the byte transfer and releases on completion.

Clock-domain crossing and timing
- 0x06 write is latched in busClk and sampled in spiSysClk before transfer; 0x06 read returns a bus-domain mirror of the SPI-side RX latch.
- Start (0x03) is stretched/synchronized into the SPI domain; busy (0x04) is latched in busClk and auto-clears when the SPI-side operation completes.
- No separate RX-ready register; use 0x04 to determine completion.

Notes
- FIFO base windows (0x10/0x20) may have a 1-cycle wait; unrelated to direct-access.
- SPI speed is programmable via 0x0B; direct-access uses the configured delay during type 0x00 operations.

-- SD Address Registers --

Holds the 32-bit SD command argument for single-block read/write operations. Located in the Control/Status register window (Wishbone 0x00..0x0F), at offsets 0x07–0x0A. Byte layout: 0x07 → SDAddr[7:0], 0x08 → SDAddr[15:8], 0x09 → SDAddr[23:16], 0x0A → SDAddr[31:24] (little-endian by address for software; SPI transmits MSB-first). Access: Read/Write with immediate acknowledge (no wait states). Reset: No defined default; contents are undefined after reset and must be programmed before use (no auto-clear). Clock domains: Written in busClk via ctrlStsRegBI; sampled in spiSysClk by the SD controllers—software must program all four bytes before asserting the start at 0x03. Consumption: readWriteSDBlock uses SDAddr[31:0] as blockAddr for CMD17 (read) and CMD24 (write), forwarding the four bytes as the command argument. Addressing mode: Value is transmitted verbatim; software must supply the correct addressing (byte address for SDSC, block index for SDHC/SDXC). Incorrect mode can cause command failure or data at unexpected locations. Typical use: Program 0x07–0x0A, set transaction type at 0x02 (10 = read, 11 = write), then assert start at 0x03; optionally read back 0x07–0x0A to verify.

-- SPI Clock Configuration Register --

Name: SPI Clock Configuration Register (CLK_DELAY); Address: 0x0B in the Control/Status block; Width/Access: 8-bit R/W; Reset: 0x00 (fastest SCK). Function: Sets the SPI SCK half-period for the wire-level shifter; each SCK half-period lasts (CLK_DELAY + 1) spiSysClk cycles; SCK frequency ≈ f_spiSysClk / (2 × (CLK_DELAY + 1)); larger values slow SCK linearly. Scope: Applies to all SPI byte transfers (single-byte engine, sendCmd, SD block read/write) except during SD initialization. SD initialization override: Hardware forces CLK_DELAY = 0x3B during init; readback of 0x0B always returns the programmed value; after init completes, operation reverts to the programmed value. Bus/CDC: Immediate acknowledge on reads/writes via spiMasterWishBoneBI; value is safely transferred into the spiSysClk domain by ctrlStsRegBI. Reset behavior: External or software reset clears the stored value to 0x00; SPI engine idles with SCK low. Dynamic update: Shifter samples this register continuously; changing it mid-transfer can alter subsequent half-cycles of the current byte; software should update only when idle (SPI engine not busy and TX path empty). Mode: SPI timing fixed to CPOL=0 (Mode 0 style); this register controls bit rate only. Examples (spiSysClk = 50 MHz): CLK_DELAY = 4 → SCK ≈ 5 MHz; CLK_DELAY = 0x3B → SCK ≈ 416.7 kHz (used during SD init).

-- RX FIFO Registers --

RX FIFO Registers (byte-wide) at address window 0x10–0x1F, selected by the bus front-end and sourced from the RX FIFO (sm_RxFifo).

Registers:
- 0x10 RX FIFO Data (RO, pop-on-read, 1-cycle wait):
- Reading returns the next FIFO byte and pops one element.
- Bus inserts a single wait-state; ack asserts on the second held cycle so returned data matches the popped element.
- Throughput: after the first read, back-to-back reads can drain at one byte per bus cycle (if not empty).
- Software must check occupancy before reading; reading while empty is undefined.
- 0x12 RX FIFO Count High (RO, immediate): returns numElementsInFifo[15:8] observed in the bus

clock domain.
- 0x13 RX FIFO Count Low (RO, immediate): returns numElementsInFifo[7:0] observed in the bus clock domain.
- Count is derived from synchronized pointers; values are eventually consistent and may briefly lag the producer domain.
- 0x14 RX FIFO Control (WO, immediate): Bit[0]=1 issues a force-empty (flush) command; other bits ignored. Generates a one-shot flush pulse in both bus and SPI clock domains; FIFO pointers/flags are synchronously cleared. No readback/status.

Bus timing/ack:
- 0x10 reads: one wait-state; ack on second cycle.
- 0x12, 0x13 reads and 0x14 write: immediate acknowledge (no wait-state).

Notes:
- All registers are 8-bit; RX data path is 8-bit.
- Force-empty uses a toggle-based CDC; a spurious pulse at power-up is theoretically possible depending on device initialization.
- The FIFO does not inherently block underflow/overflow; software must read only when count>0, and producers must honor fifoFull.
- Performance: once primed, sustained reads of 0x10 achieve one byte per bus cycle.

-- TX FIFO Registers --

TX FIFO Registers (address window 0x20..0x2F, selected when addr & 0xF0 == 0x20)
- 0x20: TX FIFO Data (write-only)
- Write: enqueue one byte into the transmit FIFO; acknowledged immediately.
- Read: returns 0x00 and incurs a one-cycle wait-state.
- 0x24: TX FIFO Control (write-only)
- Bit[0]=1: Force Empty (flush). Generates clean single-cycle flush pulses in both busClk and spiSysClk domains and immediately clears FIFO contents/status. Other bits ignored.
- Read: returns 0x00; writes acknowledged immediately.
- 0x21, 0x22, 0x23, 0x25..0x2F: reserved; reads return 0x00; writes have no defined effect.
Behavior
- TX FIFO is write-only in this window; occupancy/full/empty status is not readable.
- No overflow protection at the bus interface; software must avoid writing when the FIFO is full.
- Write-side strobe is generated combinationally on bus write to 0x20 and advances the FIFO on wrClk.
- Force-empty uses a toggle-based CDC to produce one-shot pulses in each clock domain; space successive flush commands to avoid coalescing.
- Reads are driven as 0x00 by sm_TxFifoBI; spiMasterWishBoneBI inserts the one-cycle wait-state only on reads to 0x20.

-- Read Acknowledge Timing Rules --

Read acknowledge (ack_o) is generated by spiMasterWishBoneBI. Immediate (zero-wait) ack applies to all read addresses except the FIFO base data ports: control/status block 0x00..0x0F is immediate; RX/TX FIFO window offsets other than the base (0x10+x, 0x20+x with x ≠ 0) are immediate. A single-cycle delayed ack is inserted for reads to the RX FIFO base at 0x10 and the TX FIFO base at 0x20: ack_o asserts on the second busClk cycle as ack_o = (strobe_i delayed by 1) AND (current strobe_i). The bus master must hold strobe_i, address, and select stable across this extra cycle and deassert strobe_i after ack_o to complete the cycle. Data validity for the delayed case: at 0x10 the returned byte is valid only when ack_o asserts and corresponds to the element popped by the read; at 0x20 reads return 0x00 and should not be used for data, though the same delayed ack policy is enforced. Back-to-back RX FIFO reads require deasserting strobe_i after each ack to create distinct fifoREn rising edges; each read to 0x10 incurs the same single-cycle wait, yielding a sustained rate of one byte per two busClk cycles unless the bus protocol pipelines reads correctly. To avoid stale data, software should read 0x10 only when the exposed element count at 0x12/0x13 indicates count > 0. No additional wait states are inserted beyond the single cycle at the FIFO base ports, and ack timing is independent of FIFO fullness or other runtime conditions. For all reads, sample busDataOut when ack_o asserts; immediate addresses produce ack and valid data in the same cycle, while 0x10/0x20 produce ack and valid data on the following cycle.

-- Reset Effects on Registers --

Reset is sourced from external rstFromWire and software reset (ctrlStsRegBI 0x01[0]); it is stretched/deasserted in busClk and double-synchronized into spiSysClk, producing rstSyncToBusClkOut and rstSyncToSpiClkOut. Effects per register set:
- Bus-visible ctrlStsRegBI (busClk domain): ID (0x00) retains 0x12; spiTransTypeSTB, spiTransCtrlSTB, spiTransStatusSTB clear to 0; spiDirectAccessTxDataSTB clears to 0x00; spiClkDelay clears to 0; SDAddr bytes (0x07–0x0A) are not reset (undefined until written); direct-access RX readback returns 0 immediately after reset (mirrors SPI RX register). Transaction status pipeline and CDC flops are cleared; start/reset strobes re-arm via their shift/sync chains.
- Wishbone front-end (spiMasterWishBoneBI): ack_immediate/ack_delayed have no explicit reset; unmapped reads return 0x00.
- SPI/SD control layer (spiSysClk domain): spiCtrl registers reset to entry/Idle with spiCS_n high; all request/clear pulses low; spiTransSts=0. initSD clears outputs/counters; enters Idle with SDInitRdy=1; rxDataRdyClr stays 0; spiClkDelayOut follows spiClkDelayIn when idle. readWriteSDBlock returns to Idle defaults on reset (ready=1, spiCS_n=1, counters/flags deasserted; error codes reinitialized at start of new op). sendCmd synchronous reset: state=INIT; txDataWen=0; txDataOut=0x00; rxDataRdyClr=0; respByte=0x00; respTout=0; sendCmdRdy=0 then transitions to IDLE with sendCmdRdy=1; timeOutCnt=0.
- Byte engine and SPI wire-level: spiTxRxData clears txDataOut=0x00, txDataFull=0, rxDataOut=0x00, rxDataRdy=0. readWriteSPIWireData clears shift regs/counters/flags; SCK/MOSI=0; FSM=Idle; txDataEmpty=1.
- FIFOs (sm_TxFifo, sm_RxFifo using sm_fifoRTL): On reset/force-empty, wrClk domain: fifoFull=0, bufferInIndex=0; rdClk domain: fifoEmpty=1, bufferOutIndex=0, fifoREnDelayed=0; numElements=0. Bus-side BI edge-detect/toggle clears in busClk; spi-side force-empty synchronizer flops have no explicit reset (possible benign initial pulse). Dual-port RAM (sm_dpMem_dc) is not reset/initialized; contents and registered read data are undefined until written.
- Top-level CS combine: each producer drives CS high on reset; combined spiCS_n defaults high.
Cautions: Initialize SDAddr after any reset; do not read FIFOs when fifoEmpty=1 and re-prime with fresh writes; acknowledge logic depends on strobe_i, not reset; observe that some ready flags assert only after one cycle post-reset (e.g., sendCmdRdy).

CORE CONFIGURATION

-- Programmable Parameters --

- All registers are software-programmable via the busClk-domain Wishbone-like interface.
- 0x01 Soft reset (WO, bit[0]): write 1 to request a synchronized reset across bus and SPI domains (stretch-deasserted).
- 0x02 SPI transaction type (RW, [1:0]): 00=single-byte SPI, 01=SD init, 10=SD single-block read,

11=SD single-block write.
- 0x03 SPI transaction start (WO, bit[0]): write 1 to emit a one-cycle start pulse; program type and any parameters first.
- 0x06 SPI direct-access data (RW, 8-bit): write to set TX byte for single-byte transaction; read returns the last RX byte from the most recent single-byte transaction.
- 0x07..0x0A SD block address (RW, 32-bit, little-endian by address): 0x07=[7:0], 0x08=[15:8], 0x09=[23:16], 0x0A=[31:24]; reinitialize after any reset before SD block ops.
- 0x0B SPI SCK half-period (RW, 8-bit): number of spiSysClk cycles per half-period (larger = slower). Note: during SD init, hardware overrides to 0x3B; normal operation uses the programmed value.
- RX FIFO window (0x10..0x1F):
- 0x10 Pop-and-read next RX byte (RO; read pops one element; 1-cycle latency with a bus wait-state).
- 0x12 RX FIFO count high (RO), 0x13 RX FIFO count low (RO).
- 0x14 RX FIFO force-empty (WO, bit[0]): write 1 to flush; generates one-shot flush pulses in both clock domains (space successive writes by a few spiSysClk cycles).
- TX FIFO window (0x20..0x2F):
- 0x20 Write TX byte (WO; enqueues immediately).
- 0x24 TX FIFO force-empty (WO, bit[0]): write 1 to flush; one-shot pulses in both domains.
- Address decoding: control/status at 0x00.., RX FIFO at 0x10.., TX FIFO at 0x20..; unmapped addresses read as 0x00.
- Programming sequence: set 0x02 (type), optionally 0x06 (TX byte), and 0x07..0x0A (SD block address) before asserting 0x03 (start).

-- FIFO Depth and CDC Characteristics --

Depth and pointers
- Each dual-clock FIFO (sm_TxFifo: busClk->spiSysClk, sm_RxFifo: spiSysClk->busClk) has a depth of 2^ADDR_WIDTH entries.
- Read/write pointers are ADDR_WIDTH+1 bits (extra MSB for wrap detection).
- Occupancy is computed as the unsigned difference between synchronized pointers and exposed as a 16-bit numElementsInFifo (zero-extended). Exact ADDR_WIDTH is not specified; select a value that meets worst-case buffering (e.g., 512-byte SD block transfers).

CDC scheme (flags, pointers)
- Binary pointers are synchronized across domains with a single flip-flop.
- Empty detection: pointer equality; full detection: lower bits equal with MSB difference.
- Due to single-flop binary-pointer synchronization, flags and occupancy can glitch or be stale by a few cycles under asynchronous clock phase relationships; software/consumers should gate pops/pushes with not-empty/not-full respectively.

Force-empty (flush) crossing
- A bus-driven toggle generates a force-empty event; a 3-flop synchronizer in spiSysClk and a corresponding synchronizer in busClk produce single-cycle pulses in each domain to clear FIFO pointers/flags (soft reset of contents).
- SPI-side synchronizer chain lacks explicit reset; a benign spurious pulse at power-up is possible depending on device initialization.
- Space successive force-empty events by multiple spiSysClk cycles to avoid coalescing or missed pulses.

Implementation notes / recommendations
- FIFO RAM is dual-port, dual-clock (sm_dpMem_dc) wrapped by sm_fifoRTL.
- For production-grade CDC, prefer Gray-coded pointers and two-stage synchronizers for cross-domain pointer/flag transfer.
- Account for potential latency in flag/occupancy visibility when designing flow control; verify chosen

ADDR_WIDTH against sustained throughput requirements.

-- Chip-Select Arbitration Policy --

spiCS_n is active-low and arbitrated by a wired-AND of three registered sources: spiCS_n = spiCS_nFromInitSD & spiCS_nFromRWSDBlock & spiCS_nFromSpiCtrl. Defaults: each source drives 1 in reset/idle; any source driving 0 asserts chip-select, and the aggregate returns high only when all sources return 1. Timing/clocking: all sources are synchronous to spiSysClk; the top-level AND is combinational over registered outputs for deterministic, glitch-free behavior.

Arbitration: no explicit priority—low wins. The system controller sequences requests to enforce mutual exclusion; if overlap occurs, CS remains low until all requesters release.

Per-operation behavior:
- Raw SPI byte (type 00, spiCtrl): Assert low at transfer start (txDataWen), hold through the byte, release on rxDataRdy after pulsing rxDataRdyClr.
- SD initialization (type 01, initSD): Keep CS high during 0xFF preamble clocks; assert low only while issuing CMD0/CMD1 via sendCmd; release immediately when the command engine reports ready.
- SD single-block read/write (types 10/11, readWriteSDBlock): Keep CS low for the full transaction—command, token wait, 512-byte stream, CRC, and busy polling; release at completion (after TX drain for read; after busy release for write).

Ownership: sendCmd and wire-level engines do not drive CS; they operate under their caller's CS policy. Software has no direct CS control; CS is managed by FSMs. Reset: all contributors return to idle with CS high.

-- Timeouts and Counters --

- Clocking/units baseline
- Most counters run in spiSysClk cycles; some are poll/attempt counts or byte counts. Actual wall time depends on spiSysClk and the programmed SPI bit rate.
- SPI SCK pacing: clkDelay (8-bit) sets the SCK half-period in readWriteSPIWireData; clkDelayCnt counts to clkDelay per half-phase. During initSD, clkDelay is overridden to 0x3B.

- SPI wire engine (readWriteSPIWireData)
- clkDelayCnt: cycles each SCK low/high half-phase up to clkDelay; no timeout.
- bitCnt: counts transmitted bits; asserts rxDataRdy after 8 bits. Supports continuous back-to-back bytes if data is available; no timeout.

- ctrlStsRegBI pulse/stretching
- Reset-stretch: 6-bit shift register keeps reset asserted ~6 busClk cycles; then double-synchronized into spiSysClk.
- Start-stretch: 6-bit level stretch in busClk; 3-flop synchronizer and edge detect generate a single-cycle start pulse in spiSysClk.
- No explicit timeouts; fixed-cycle pulse widths and CDC latencies only.

- Wishbone wait-state (spiMasterWishBoneBI)
- Deterministic 1-cycle read wait-state on FIFO base addresses (0x10, 0x20). No timeout logic.

- Direct byte staging (spiTxRxData)
- Single-byte TX/RX flags/arbitration; no counters or timeouts.

- SD initialization (initSD)

- Pre-clock: loopCnt sends 160 dummy 0xFF bytes (CS high) to generate ~1280 SCK cycles.
- CMD0 retry: loopCnt increments per attempt; stops when R1==0x01 or loopCnt==0xFF. Timeout/fail sets initError=2'b01.
- CMD1 retry: loopCnt up to 0xFF; inter-attempt pacing via delCnt2 (0..0xFF) nested under delCnt1 (0..0x177). Stops when R1==0x00 or loopCnt==0xFF. Timeout/fail sets initError=2'b10.
- No absolute timebase; limits are attempt counts plus deliberate pacing delays.

- SD command sender (sendCmd)
- timeOutCnt (10-bit) during busy polling: increments while respByte[7]==1; if it reaches 10'h200 (512 polls), sets respTout=1 and aborts.
- No timeout for "no response" (rxDataRdy never asserts): FSM can remain in WAIT_RX indefinitely; system-level watchdog may be required.

- SD single-block read/write (readWriteSDBlock)
- Data length: loopCnt is 9-bit and wraps after 512 bytes; transfer ends on wrap (implicit 512-byte count).
- Read (CMD17) token wait: timeOutCnt with pacing via delCnt1/delCnt2 (each up to 0xFF). If start token 0xFE not seen before timeOutCnt≈0x048, sets readError=2'b10.
- Write (CMD24):
- Data-response wait: timeout threshold ≈0xF00; if expected token (0x05) not received, sets writeError=2'b10.
- Busy-release wait: timeout threshold ≈0x0B6; if card remains busy (MISO==0), sets writeError=2'b11.
- Command phase: if response times out or R1!=0, sets readError/writeError=2'b01.
- Pacing counters delCnt1/delCnt2 spread poll intervals; real time depends on spiSysClk and SCK rate.

- Asynchronous FIFOs (sm_fifoRTL via sm_TxFifo/sm_RxFifo)
- Pointers: bufferInIndex and bufferOutIndex are ADDR_WIDTH+1-bit binary counters; MSB used for wrap/full/empty logic.
- Occupancy: numElementsInFifo is a 16-bit zero-extended difference computed in read domain; reflects writes after CDC latency.
- Read edge detect: fifoREnDelayed produces a one-pop per 0->1 edge; holding fifoREn high does not auto-repeat.
- No timeout logic; proper not-full/not-empty handshakes required to avoid overflow/underflow.

- FIFO force-empty timing (sm_TxfifoBI/sm_RxfifoBI)
- Bus-side edge creates a one-cycle forceEmptySyncToBusClk pulse; a 3-flop CDC into spiSysClk generates a single spiSysClk-cycle forceEmpty pulse via XOR of the last two stages.
- Space successive force-empty events by several spiSysClk cycles to avoid coalescing; no timeout logic.

- Transaction controller (spiCtrl)
- No internal counters/timeouts; progress governed by submodule handshakes (rxDataRdy, SDInitRdy, readWriteSDBlockRdy).

- Implementation notes
- Timeout thresholds in readWriteSDBlock (~0xF00, ~0x0B6, ~0x048) are design-specific constants; verify in RTL if precise timings matter.
- Real-time behavior depends on spiSysClk and clkDelay; SCK frequency ≈ spiSysClk / (2*(clkDelay+1)) subject to off-by-one details.

-- Error Code Definitions --

Error codes are reported via ctrlStsRegBI at read address 0x05. The 8-bit value is a live mirror (not sticky) of upstream error outputs sampled each busClk.

Bit mapping
- [7:6] = 0 (reserved)
- [5:4] = SDWriteError (CMD24)
- [3:2] = SDReadError (CMD17)
- [1:0] = SDInitError (initSD)

Coding (2-bit per field; values not listed are reserved/undefined)
- 00: No error (idle or last operation succeeded)

SDInitError (initSD sequence)
- 00: No error
- 01: CMD0 failed (did not reach R1 = 0x01 before retry limit)
- 10: CMD1 failed (did not reach R1 = 0x00 before retry limit)
- 11: Reserved/undefined

SDReadError (single-block read, CMD17)
- 00: No error
- 01: Command failed (response timeout via sendCmd or R1 != 0)
- 10: Start-data token 0xFE not seen before timeout
- 11: Reserved/undefined

SDWriteError (single-block write, CMD24)
- 00: No error
- 01: Command failed (response timeout via sendCmd or R1 != 0)
- 10: No or invalid data-response token after block data
- 11: Card remained busy too long after accept

Behavior and clearing
- Each field reflects the last completed operation for its function.
- The corresponding field is cleared to 00 at the start of a new transaction and on global or soft reset.
- There is no explicit bus-side clear; 0x05 continuously mirrors upstream error signals.

Scope and domain
- Only SD init/read/write operations produce error codes; raw single-byte SPI transfers (type 00) do not affect 0x05.
- Errors originate in the spiSysClk domain and are sampled into busClk by ctrlStsRegBI.
- Internal response timeouts (sendCmd.respTout) are mapped to the "command failed" code (01) for the relevant operation.

Reset
- Writing 0x01 with bit0 = 1 (soft reset) resets upstream FSMs, clearing all error fields to 00.

-- Integration and Performance Notes --

Integration summary: spiMaster combines a Wishbone-like bus front-end (spiMasterWishBoneBI), control/status register block with CDC (ctrlStsRegBI), SD controller FSMs (initSD, readWriteSDBlock), a shared command sender (sendCmd), a single-byte TX/RX staging register (spiTxRxData), a programmable SPI byte engine (readWriteSPIWireData), and dual-clock FIFOs (sm_TxFifo, sm_RxFifo) to bridge busClk and spiSysClk.

Clocks, resets, CDC: Two domains (busClk for software, spiSysClk for SPI/SD). Reset and start are generated in busClk, stretched, and synchronized into spiSysClk; status/RX byte are synchronized back to busClk. Force-empty events use toggle-based CDC with one-shot pulses in both domains; space successive force-empty commands by several spiSysClk cycles. Synchronizer flops for force-empty are not reset; a benign spurious pulse at power-up is possible.

Bus interface and address map: 16-byte windows decoded by spiMasterWishBoneBI: 0x00.. control/status (ctrlStsRegBI), 0x10.. RX FIFO, 0x20.. TX FIFO. Acknowledge policy is zero-wait-state for most accesses; reads to the 0x10 and 0x20 base addresses insert exactly one wait-state. Bus masters must hold strobe across this wait. RX FIFO base reads pop one byte with a 1-cycle latency handled inside the BI; TX FIFO writes are immediate.

SPI protocol and chip-select: SPI engine generates SCK with programmable half-period (clkDelay), CPOL=0, MSB-first. MISO is sampled in the low phase; bit count advances on rising edges. initSD forces a slower SPI clock during card initialization, restoring clkDelay afterward. spiCS_n is the AND of CS requests from initSD, readWriteSDBlock, and spiCtrl; any active submodule can hold CS low—avoid overlapping activities.

Shared command path: sendCmd services both initSD and readWriteSDBlock; request lines are ORed and command bytes are bitwise OR-combined each cycle. System integration must ensure only one source drives commands at a time and holds fields stable for the entire transaction. Busy polling counts busy responses only; no internal timeout for missing responses (WAIT_RX can hang)—consider a system-level watchdog.

Data path and FIFOs: Software enqueues TX bytes at 0x20; SPI side dequeues. RX bytes are enqueued by SPI and popped by software reads at 0x10; counts exposed at 0x12/0x13. sm_fifoRTL uses single-flop binary pointer CDC; at high skew/asynchronous ratios there is risk of metastability/flag glitches—use Gray-coded pointers with multi-stage synchronizers for production/high-speed. Read enable is edge-sensitive on the read side; writes are level-sensitive on the write side. Force-empty controls: TX at 0x24[0]=1, RX at 0x14[0]=1.

Staging register (spiTxRxData): Provides single-byte TX/RX for the wire engine and raw SPI. Priority rules: TX clear overrides set in the same cycle; RX set overrides clear. Do not write a new TX byte while txDataFull is asserted unless overwriting is acceptable; coordinate rxDataRdyClr among consumers.

Control layer and software use: spiCtrl dispatches based on type (00=raw SPI, 01=SD init, 10/11=SD block read/write), issues one-cycle request pulses, and exposes busy at 0x04. Error flags at 0x05 report init/read/write failures. Typical sequence: program spiTransType (0x02) and clkDelay (0x0B), set SD address (0x07..0x0A) for block ops, then write start at 0x03; for raw SPI use 0x06 for TX/RX and poll busy until clear.

Performance notes: Per-byte wire time ~16*(clkDelay+1) spiSysClk cycles (8 bits, two half-cycles/bit), excluding controller overhead. readWriteSPIWireData supports back-to-back streaming with no idle cycles when txDataFull is presented at byte boundaries. SD block transfers stream 512 bytes while CS remains low; throughput is bounded by SPI bit rate and card response/token waits.

Cautions and limitations: Avoid concurrent requests; start new transactions only when spiTransSts indicates idle and, for SD flows, sendCmd is ready. Honor FIFO flow control (don't write on full; don't read on empty). Verify target device SPI mode compatibility (Mode 0) and adjust clkDelay for signal integrity. SD flows implement legacy CMD0/CMD1 only; CMD8/ACMD41 (SD v2.0/SDHC) are not supported. CRC handling is simplified (fixed 0xFF, read CRC ignored); ensure card CRC is disabled. Use force-empty for abort/recovery and allow settling time between commands.

-- Protocol and Compliance Notes --

- SPI mode and timing: Master, MSB-first, CPOL=0; MISO is sampled just before SCK rising edge (Mode 0 intent—verify target tolerance). SCK period programmable via clkDelay; initSD limits SCK to ≤400 kHz, then reverts to user rate. CS is active-low and is the AND of submodule requests; held low for full transactions and high when idle. Byte engine supports continuous back-to-back bytes; single-byte op holds CS for one byte only.
- SD SPI protocol coverage: Initialization uses legacy CMD0 (CRC 0x95) followed by CMD1 loops; no CMD8/ACMD41 and no SDHC/SDXC negotiation/detection. Only single-block operations supported (CMD17 read, CMD24 write). Block length assumed 512 bytes; CMD16 not issued. CRC disabled except for CMD0; other command CRC fields fixed to 0xFF; data CRC not checked and set to 0xFF on writes. Responses/tokens: R1 accepted when MSB=0; read expects 0xFE start token; write expects data-response 0x05 and busy polling until release. Address is transmitted as provided (no byte/block address translation).
- Timeouts and pacing: Waits are poll-count based and scale with spiSysClk and SCK; some paths can stall indefinitely if no response. CS must remain asserted across SD command, response, and data phases.
- Bus interface compliance: Simplified Wishbone-like interface with fixed address windows (0x00 control/status, 0x10 RX FIFO, 0x20 TX FIFO). ack_o is immediate except one wait-state inserted on reads of 0x10 and 0x20. No ERR/RTY/STALL, fixed timing, combinational selects/reads.
- Clock-domain crossing and FIFOs: Dual-clock FIFOs bridge domains and provide element count and force-empty. Pointer synchronization uses single-flop binary crossing (not Gray-coded/two-flop), which can cause flag/count glitches under extreme clock ratios/skew; use within characterized conditions or upgrade CDC. Read enable is edge-sensitive; write enable is level-sensitive; external logic must honor not-empty/not-full. Force-empty uses a toggle synchronizer; spi-side flops are unreset and may emit a spurious pulse at power-up.
- Command engine integration: Two producers are ORed; command bytes are the bitwise OR of sources per cycle. System must ensure single-source ownership and hold all command fields stable until sendCmdRdy.
- Configurability and interoperability notes: Only SPI Mode 0 supported (no CPOL/CPHA configurability). Continuous streaming without CS gaps suits SD SPI; engine does not enforce CS toggling or inter-byte delays required by some SPI slaves. Software must set clkDelay to meet ≤400 kHz during init and provide correct address form (byte vs block) per card type/init path. CMD59 (CRC enable/disable) unsupported.