Generated Specification Document

INTRODUCTION

-- Purpose and Scope --

Purpose: Provide a synthesizable USB 2.0 device (function) controller top-level that bridges a UTMI/UTMI+ PHY to a WISHBONE system bus and a shared external SSRAM payload buffer, handling packet/protocol processing, endpoint control, DMA handshakes, and interrupt/event reporting.

Scope: The usbf_top integrates the complete device-side datapath and control including: UTMI interface adaptation (receive/transmit data strobes, line-state handling, suspend/resume, reset detection, HS/FS speed negotiation); packet decode/encode with CRC5/CRC16; protocol engine for token/data/handshake sequencing; endpoint register file and match logic for up to 16 endpoints (EP0–EP15, EP1–EP15 optionally enabled via macros); ping■pong buffer management (buf0/buf1) and DATA PID/buffer select tracking; per-endpoint DMA request/acknowledge generation and a memory arbiter sharing single-ported SSRAM between PHY-side DMA and WISHBONE accesses; WISHBONE-mapped control/status and memory windows with acknowledge generation; consolidated interrupt outputs (inta/intb) and error/status sources (PID check, CRC, NSE, timeouts); frame/microframe timing capture and vendor UTMI sideband control/status (VControl/VStatus). Address widths and buffer memory size are parameterized via macros.

Out of scope: USB host functionality, the physical transceiver (PHY), high-level USB class/descriptor firmware, and application-level data management. The core assumes system firmware configures endpoint CSRs and buffers, services DMA/interrupts, and provides external SSRAM and a UTMI■compliant PHY.

-- USB Compliance and Supported Modes --

USB 2.0 device core with an 8■bit UTMI+ Level 1 PHY interface. Supports Full■Speed (12 Mbps) and High■Speed (480 Mbps) operation; Low■Speed device mode is not supported. High■Speed entry via standard USB 2.0 chirp K/J negotiation (falls back to Full■Speed if HS negotiation fails). UTMI control signals XcvSelect, TermSel, SuspendM, and OpMode are driven per mode and power state; FS termination is enabled/disabled as required and transceiver select toggles HS/FS. Implements full protocol decode and integrity checks: PID complement checking, CRC5 (token) and CRC16 (data). Recognizes OUT/IN/SOF/SETUP and HS■only PING; SPLIT/ERR/PRE and other host/hub■specific PIDs are ignored for endpoint matching. Handshake generation includes ACK/NAK/STALL in FS and adds NYET in HS when no buffers are available. Data PID sequencing (DATA0/1/2/MDATA) is enforced by endpoint policy. SOF handling is implemented; in HS, microframes are tracked (mframe count and timing). Bus reset detection and recovery are implemented (SE0 timing) and speed re■negotiation occurs after reset. Suspend/resume complies with USB 2.0 timings: enters suspend after >3 ms idle; supports remote wake via resume_req input (resume K signaling duration ≥1 ms, with appropriate settle delays); SuspendM/OpMode are managed for low■power. Attach/detach and VBUS presence are monitored; attach is debounced (~100 ms) and status is exposed. Supported transfer types: Control (EP0), Bulk, and Isochronous; interrupt type is not explicitly implemented as a distinct mode. Host mode and test modes are not implemented.

-- Feature Summary --

- USB 2.0 device core with UTMI interface: supports Full-Speed and High-Speed, automatic FS/HS speed negotiation (chirp), reset detection, suspend/resume; exposes mode_hs, usb_reset, usb_suspend, usb_attached.
- UTMI PHY glue: drives XcvSelect, TermSel, SuspendM, OpMode; handles resume signaling (drive K), line-state timing, and wake/suspend timing per USB spec.
- Packet engine: PID decode/generation, token/data handshakes (ACK/NACK/STALL/NYET), PING support, data PID sequencing, CRC5/CRC16 check and CRC16 generation, sequence and timeout error handling.
- Endpoint subsystem: up to 16 endpoints (EP0–EP15), compile-time selectable; each EP has CSR, two buffer descriptors (buf0/buf1) with address/size, auto PID toggle, buffer selection, DMA enable, direction/type (CTRL/ISO/BULK/INT), max packet size.
- Double-buffered data path with automatic descriptor updates, buffer rollover, out-to-small detection, and zero-length packet support.
- Integrated DMA engine (IDMA): streams between UTMI byte data and 32-bit memory, tracks sizes, assembles words, supports IN/OUT, exposes idma_done and byte-count used.
- Shared SSRAM buffer memory: 32-bit wide, parameterizable size via SSRAM_HADR; arbiter mediates between protocol DMA and WISHBONE host; dual-clock safe.
- WISHBONE slave interface: 32-bit data, address decoding splits register file vs buffer memory (via address MSB); supports read/write cycles with proper ack and clock crossing.
- Register file (RF): function address, per-EP control/status, buffer descriptors, maskable interrupt control, vendor UTMI register passthrough (write strobe + status readback).
- Interrupts: two aggregated outputs (inta_o/intb_o) combining per-EP events (buffer done, CRC16, timeouts, PID/sequence) and system events (attach/detach, suspend enter/exit, USB reset, RX errors); maskable sources.
- Frame/timekeeping: captures SOF frame number, microframe count, and inter-SOF timing into frm_nat for diagnostics.
- External DMA handshake per endpoint: dma_req_o[15:0]/dma_ack_i[15:0] for system-level DMA coordination.
- Clean clock-domain crossings between system clock and PHY clock throughout control/data paths.
- Top-level I/O: UTMI 8-bit data and control pins, WISHBONE bus, interrupt lines, suspend output, resume request input, SSRAM memory port, and UTMI vendor control/status.

-- Endpoint Support and Buffering --

- Supported endpoints: EP0 is always present. EP1, EP2, EP3 are enabled by build-time defines (USBF_HAVE_EP1..3). EP4–EP15 are instantiated as dummies (no function) unless corresponding defines are enabled. A power-up message confirms the contiguous set of enabled endpoints.
- Endpoint configuration (per endpoint CSR):
- Direction: csr[27:26] = 00 CTRL, 01 IN, 10 OUT.
- Transfer type: csr[25:24] = 01 Isochronous, 10 Bulk (others reserved).
- State: csr[23:22] = 01 Disabled, 10 Stalled.
- DMA enable: csr[15] (ignored for control endpoints).
- Max payload size: csr[10:0] (11-bit, bytes).
- Additional fields: csr[17] large packet allowed, csr[16] small packet allowed, csr[12:11] transaction framing. Data PID and buffer select are tracked in uc_dpd (csr[29:28]) and uc_bsel (csr[31:30]).
- Double buffering model (per endpoint):
- Two 32-bit buffer descriptors: buf0 and buf1. Reset value 0xFFFF_FFFF indicates unallocated.
- Descriptor fields: [31] NA flag (1 = not available); [30:17] size (14 bits, bytes); [SSRAM_HADR+2:4] word-aligned base address; [3:0] low address nibble. An address of all 1s also implies NA.

- Active buffer selection: if DMA enabled, hardware uses buf0 (uc_bsel not used for fetching). For control endpoints, selection follows token direction (IN uses buf1, OUT uses buf0). Otherwise, hardware prefers the non-NA buffer; uc_bsel toggles the user-visible next buffer when a buffer completes.
- Buffer lifecycle:
- IN: data is read from the active buffer until empty (new_size == 0) then buffer_done asserted.
- OUT: writes proceed until remaining size < max payload (buffer_full) then buffer_done asserted.
- On UPDATE phase, if DMA is enabled and buffer_done is reached, buf0_rl is pulsed to restore buf0 to its original descriptor (ring behavior). Otherwise, the active bufX is updated (bufX_set) with idin carrying the new size and next start address.
- uc_bsel_set and uc_dpd_set update csr[31:30] and csr[29:28] with the next buffer select and DATA PID sequencing.
- Out-of-size conditions: to_small (packet < max and small not allowed) or to_large (packet > max and large not allowed) cause NAK on OUT. In HS OUT with DMA, a short packet (sizu_c != max) sets out_to_small, updates the descriptor with the actual size, and raises an interrupt bit.
- DMA request/ack per endpoint:
- Each non-control endpoint exposes a dma_req bit (dma_req_o[15:0]), acknowledged via dma_ack_i.
- OUT endpoints assert DMA request while dma_out_cnt != 0; IN endpoints assert while dma_in_cnt < programmed buffer length.
- Flow control flags exported to the protocol layer per matched endpoint:
- dma_in_buf_sz1: at least one max packet of IN data is ready.
- dma_out_buf_avail: at least one max packet space is available for OUT.
- Handshake is edge/level conditioned with hold logic to avoid premature drop until ack.
- Protocol interaction and backpressure:
- If endpoint is stalled, device answers STALL; if both buffers are NA, or DMA cannot start (no_buf0_dma), NAK is returned (NYET in HS when appropriate). PING in HS gets ACK without data.
- Zero-length IN packets are generated when max_pl_sz == 0.
- Memory and addressing:
- External shared buffer memory is 4 KiB (SSRAM_HADR = 9 ⇒ 2^(SSRAM_HADR+1) words × 4 bytes). Descriptors hold byte addresses; the IDMA uses word addresses [SSRAM_HADR+2:2] and arbitrates with the WISHBONE via usbf_mem_arb.
- Endpoint register access:
- For each endpoint, address window exposes: CSR (offset 0), INT status (offset 1), BUF0 (offset 2), BUF1 (offset 3). Writes to BUFx set/refresh descriptors; RL (buf0_rl) re-loads BUF0 from its original value.
- Control endpoint (EP0):
- Always present. Shares the same CSR/BUF interfaces, but CSR.DMA is effectively ignored for control transfers; buffer selection follows token direction rather than uc_bsel.
- Interrupts and status:
- Per-endpoint interrupt causes include: buffer0/1 done, UPID change, CRC16 error, timeout (ACK or DATA), sequence error, and OUT short packet (out_to_small). Function-level interrupts report attach/detach, suspend/resume, reset, PID/CRC errors.

-- Clocking and Reset Overview --

- Clock domains
- phy_clk_pad_i (PHY domain): Drives the entire USB datapath and protocol: UTMI interface and line-state FSM (usbf_utmi_if/usbf_utmi_ls), Packet Layer (usbf_pl) including PD/PA/PE/IDMA, the shared SRAM arbiter (usbf_mem_arb), and the "clk" side of the register file/endpoints (usbf_rf/usbf_ep_rf).
- clk_i (System/WISHBONE domain): Drives the WISHBONE interface (usbf_wb), the "wclk" side of the register file/endpoints (bus-visible registers, interrupt aggregation), DMA request aggregation, and

select top-level status synchronizers.

- External memory clocking
- The shared SRAM interface (sram_adr_o/sram_data_o/sram_we_o/sram_re_o) is generated in the PHY clock domain (phy_clk_pad_i). The external SSRAM should be timed to the PHY clock domain.

- Reset scheme
- rst_i is an active-low reset distributed to all sub-blocks (forwarded to the PHY as phy_rst_pad_o). In the delivered RTL, resets are synchronous to the local clock (if(!rst) in posedge clocked always blocks). If USBF_ASYNC_RESET is defined at compile time, key always blocks switch to asynchronous, active-low reset (negedge rst in sensitivity lists).
- UTMI line-state FSM (usbf_utmi_ls) treats usb_vbus_pad_i as an additional POR source: when asserted it drives the FSM to POR.

- Clock-domain crossings (CDC)
- WISHBONE to PHY (usbf_wb): wb_stb_i & wb_cyc_i are sampled into the PHY domain (wb_req_s1). Address/data/we are used while the request is asserted; wb_ack_o is re-timed back to clk_i using a small pulse-stretch/edge-detect pipeline (wb_ack_s1/1a/2) to produce a one-cycle pulse on clk_i.
- System memory access (SRAM): usbf_wb presents memory (ma_*) control signals already in the PHY domain; usbf_mem_arb arbitrates between PHY DMA (mreq) and WISHBONE-initiated memory cycles (wreq) all on phy_clk, then drives the SSRAM. The WISHBONE-side acknowledge for memory cycles (ma_ack) is generated from PHY-domain handshake.
- Register file and endpoint state (usbf_rf/usbf_ep_rf): Bus accesses occur on wclk (clk_i); endpoint state, counters, and DMA pacing largely occur on clk (phy_clk). Level-type interrupt sources from endpoints are sampled and ORed in the wclk domain for inta/intb.
- DMA request/ack (per-endpoint in usbf_ep_rf): dma_req is asserted in the wclk domain. The external dma_ack (top-level input) is synchronized into the PHY domain (r4/r5) and a write-clear handshake is synchronized back into wclk, creating a robust 2-domain handshake.
- UTMI suspend/resume handoff: suspend_clr from the PHY domain is sampled in clk_i (suspend_clr_wr) to clear a stretched resume_req_r generated on clk_i from resume_req_i.

- Backpressure/feedback
- tx_valid_out into the packet engine is driven from the UTMI TxValid_pad_o (PHY domain) to gate protocol-engine timers (e.g., ACK timeout logic) accurately to what the PHY transmits.

- Integration guidance
- Prefer running clk_i and phy_clk_pad_i synchronous or frequency-related to minimize CDC complexity. If run asynchronously, observe that: (1) WISHBONE address/data/we must remain stable for the duration of a request until wb_ack_o; (2) all critical CDC paths include internal handshakes/synchronizers, but integrators should still include CDC constraints in timing sign-off.
- Ensure external SSRAM timing matches the PHY clock, and that rst_i is held low long enough for both clock domains to observe a clean reset.
- If an asynchronous reset strategy is required, define USBF_ASYNC_RESET so all guarded blocks use async active-low reset.

-- External Interfaces Overview --

- Clock and reset
- clk_i: System/Wishbone clock domain.
- phy_clk_pad_i: UTMI/PHY clock domain.
- rst_i: Synchronous active-high reset for the core; also forwarded to the PHY as phy_rst_pad_o.

- Wishbone slave interface (32■bit)

- wb_addr_i[`USBF_UFC_HADR:0], wb_data_i[31:0], wb_data_o[31:0], wb_we_i, wb_stb_i, wb_cyc_i, wb_ack_o.
- Address bit wb_addr_i[12] selects space: 0 = Register File (RF), 1 = Buffer Memory window (SSRAM).
- Single-cycle classic handshake; data width 32 bits; internal clock domain crossing handled.

- External SSRAM buffer memory interface
- sram_adr_o[SSRAM_HADR:0]: word address to external memory.
- sram_data_o[31:0]: write data; sram_data_i[31:0]: read data.
- sram_we_o: write strobe; sram_re_o: read enable (held asserted; memory must tolerate RE high).
- Single physical port arbitrated between USB DMA (PHY side) and Wishbone accesses; DMA has priority.
- Memory size = (2^(SSRAM_HADR+1)) × 4 bytes (word addressed).

- UTMI+ Low-Pin Interface (to USB PHY)
- TX: DataOut_pad_o[7:0], TxValid_pad_o, TxReady_pad_i.
- RX: DataIn_pad_i[7:0], RxValid_pad_i, RxActive_pad_i, RxError_pad_i.
- Line control/status: LineState_pad_i[1:0] (input), XcvSelect_pad_o, TermSel_pad_o, SuspendM_pad_o, OpMode_pad_o[1:0].
- Power/status: usb_vbus_pad_i (input), usb_attached/usb_suspend handled internally and exposed via susp_o.
- Supports FS/HS negotiation, suspend/resume signaling (drive-K handled internally).

- Interrupts
- inta_o, intb_o: Aggregated interrupt lines combining endpoint and core/RF events (masking/aggregation managed internally).

- Endpoint DMA handshake (per endpoint)
- dma_req_o[15:0]: Core DMA request lines (one per endpoint 0..15; only enabled endpoints will assert).
- dma_ack_i[15:0]: External DMA engine acks; used to pace endpoint DMA transfers.

- Suspend/Resume control
- susp_o: Reflects current USB suspend state.
- resume_req_i: System request to initiate resume signaling; internally latched until the PHY clears suspend.

- Vendor/PHY sideband control and status
- VStatus_pad_i[7:0]: Vendor status inputs sampled into the register file space.
- VControl_pad_o[3:0], VControl_Load_pad_o: Vendor control output bus with a load strobe when software writes the control register.

- Functional address (device address) and endpoints
- Device address managed in RF and used by protocol layer; endpoints 0..15 supported at the interface level; actual enabled set is compile-time configurable (USBF_HAVE_EPn).

- Clock domains and CDC
- Two asynchronous domains: Wishbone (clk_i) and UTMI/PHY (phy_clk_pad_i). All crossings (registers, DMA, memory arbitration) are handled internally; no external CDC constraints are required beyond providing stable clocks.

-- Deliverables and Supported Configurations --

- Deliverables:
- Synthesizable Verilog RTL for the complete USB Function core, including: usbf_top, usbf_utmi_if, usbf_utmi_ls, usbf_pl, usbf_pd, usbf_pa, usbf_pe, usbf_idma, usbf_mem_arb, usbf_wb, usbf_rf, usbf_ep_rf, usbf_ep_rf_dummy, usbf_crc16, usbf_crc5.
- Top-level (usbf_top) with UTMI+ PHY pad interface, WISHBONE slave interface, external synchronous SRAM interface, interrupt lines (inta_o/intb_o), DMA request/ack vectors, suspend/resume support, and UTMI vendor control/status pass-through.
- Parameterization hooks: SSRAM_HADR (default from macro; here 9 -> 10-bit word address, 4 KB buffer), WISHBONE address high bit macro USBF_UFC_HADR (here 12), and address decode macros USBF_RF_SEL/USBF_MEM_SEL (registers vs. buffer memory window on wb_addr_i[12]).
- Compile-time feature macros: USBF_HAVE_EP1..USBF_HAVE_EP15 (endpoint enables), USBF_DEBUG (diagnostics), USBF_TEST_IMPL (test hooks), optional USBF_ASYNC_RESET (switch reset style to async in supported flops).
- Built-in simulation diagnostics ($display) for endpoint configuration summary and debug traces in key state machines.

- Supported Configurations:
- Endpoints: Up to 16 endpoints (EP0..EP15). Each endpoint implements double buffering (buf0/buf1), per-endpoint CSR, DMA request line, and interrupt sources; EP0 always present. Current build enables EP1..EP3 macros (EP0–EP3 active); others instantiate dummies by default, but can be enabled via USBF_HAVE_EP.
- Transfer types and roles: Endpoint mode bits support Control (EP0), IN, OUT; transfer type bits include Isochronous and Bulk (csr[25:24]). DATA PID sequencing (DATA0/1/2/MDATA) and token responses (ACK/NACK/STALL/NYET) handled in hardware.
- Speeds and PHY: UTMI+ interface supporting Full-Speed and High-Speed operation with automatic reset, suspend, resume, and HS/FS speed negotiation. UTMI signals: DataIn/Out, TxValid/Ready, RxValid/Active/Error, XcvSelect, TermSel, SuspendM, OpMode, LineState, usb_vbus. Vendor pass-through: VControl_pad_o, VControl_Load_pad_o, VStatus_pad_i.
- System bus: WISHBONE slave, 32-bit data, address width A12:0 per macro (USBF_UFC_HADR=12). Register file vs. buffer memory selected by address bit (USBF_RF_SEL/USBF_MEM_SEL on wb_addr_i[12]).
- Buffer memory: External synchronous SRAM port (32-bit data). Address width is SSRAM_HADR+1; default SSRAM_HADR=9 -> 4 KB total buffer space. Built-in arbiter (usbf_mem_arb) manages PHY-side DMA and host WISHBONE accesses across clock domains.
- DMA: Per-endpoint DMA request vector (dma_req_o[15:0]) and corresponding ack inputs, with internal IDMA engine for endpoint data moves, word packing/unpacking, and zero-length packet support.
- Clocks and resets: Two clock domains (phy_clk_pad_i for link/PHY; clk_i for WISHBONE/control). Synchronous resets by default; optional asynchronous reset on selected flops when USBF_ASYNC_RESET is defined.
- Interrupts: Two aggregated interrupt outputs (inta_o/intb_o) with maskable sources including endpoint events (buffer done, UPID, CRC16, timeout, sequence error, out-to-small) and root function events (attach/detach, suspend/resume transitions, USB reset, RX errors, CRC5/PID errors).
- Protocol compliance features: Full PID decode/generation, CRC5/CRC16 check and generation, FS/HS timing parameters (ACK/data timeouts), NYET handling in HS OUT flow, and control endpoint handling integrated with address match (function address).
- Power management: Suspend output (susp_o), external resume request input (resume_req_i) with UTMI resume signaling, and automatic SuspendM/OpMode management.

-- Assumptions and Limitations --

- Protocol/PHY assumptions
- UTMI 8■bit parallel interface is assumed (no ULPI). Line-state, XcvSelect, TermSel, OpMode follow UTMI conventions.
- HS/FS operation is implemented; Low-Speed is not explicitly supported. Speed negotiation logic targets HS/FS only.
- Timing constants (reset/suspend/resume, handshake timeouts) are hard-coded via macros and implicitly assume a specific UTMI clock (typically 60 MHz). Using a different phy_clk requires retuning these constants.
- Resume (K) signaling is generated internally; remote-wakeup trigger from the register file (rf_resume_req) is exposed but not wired into the UTMI block at top level (resume is driven from resume_req_i via software).

- WISHBONE interface limitations
- 32■bit data bus, no byte select signals implemented, and no burst support. Unaligned/byte writes are not supported.
- Simple request/acknowledge protocol with single-cycle pulse acks; no STALL/ERR responses. Masters must tolerate multi-cycle response latency and single-cycle ack pulses.
- Address decoding uses compile-time macros; the top-level split between register file and memory depends on a single address bit (wb_addr_i[12]). This must match system integration.

- Memory/IDMA/arbiter assumptions
- External buffer memory is a dedicated 32■bit SSRAM-like interface; sram_re is held asserted continuously. Designs that require qualified read■enable may need adaptation.
- The arbiter grants either the PHY DMA or WISHBONE, with mack combinatorially tied to mreq (assumes zero■wait read visibility). Real memories with latency need additional handshake logic.
- Addressing is word-based; IDMA assembles/disassembles 32■bit words from 8■bit UTMI stream in little■endian byte order.
- Buffer descriptors use 14■bit size and an address field that includes lower nibble for flags (bsel/dpid). Software must program valid, aligned addresses and sizes. No hardware bounds checking beyond simple wrap logic.
- When DMA and the computed next write address equals end of buffer window, address wraps to 0 (global wrap), not to a per■buffer base; improper programming can corrupt memory.

- Endpoint/register file constraints
- Up to 16 endpoints (0–15) are implemented; EP0 is always present. Additional endpoints are enabled at compile time via USBF_HAVE_EPn macros and must be defined contiguously from EP1 upward (the code warns if gaps exist).
- Two buffers per endpoint (BUF0/BUF1). Software must manage buffer allocation, sizes, and status fields; hardware does not protect against overlapping buffers.
- DMA request/ack for each endpoint crosses clock domains using simple handshakes; integrators must verify CDC robustness in their environment.
- Interrupts (inta/intb) are level/latched on the WISHBONE clock domain and cleared by specific register reads; software must service them to avoid overrun or lost events.

- Packet engine/protocol handling
- PID sequencing for data packets supports DATA0/1/2/MDATA per transfer type. Some choices assume HS/FS rules (e.g., NYET use only in HS).
- Control transfers are minimally handled in hardware (token/data/handshake). Full USB chapter 9 handling (descriptors, requests) is not provided; software must implement protocol policy via register programming.
- Timeouts and error handling (CRC5/CRC16, sequence errors, NAK/STALL/NYET) follow fixed state machines; behavior outside USB spec corner cases is not parameterized.

- Clocks/resets and CDC
- Two primary clock domains: phy_clk (UTMI) and wclk (WISHBONE). Multiple signals cross between them; most crossings are registered, but not all are double■synchronized. System integrators should review CDC paths for their timing/clocking.
- Resets can be synchronous or (optionally) asynchronous via USBF_ASYNC_RESET. Without ASYNC reset, clocks must be running when reset deasserts.

- Build/configuration caveats
- Macros for address widths and decodes (USBF_UFC_HADR, USBF_SSRAM_HADR, USBF_RF_SEL, USBF_MEM_SEL) appear multiple times; ensure final values are consistent in your build to avoid mismatched decoding or memory sizing.
- Simulation■only code is present (initial $display, debug messages under USBF_DEBUG/USBF_TEST_IMPL). Synthesis tools may warn; gate out debug for production.

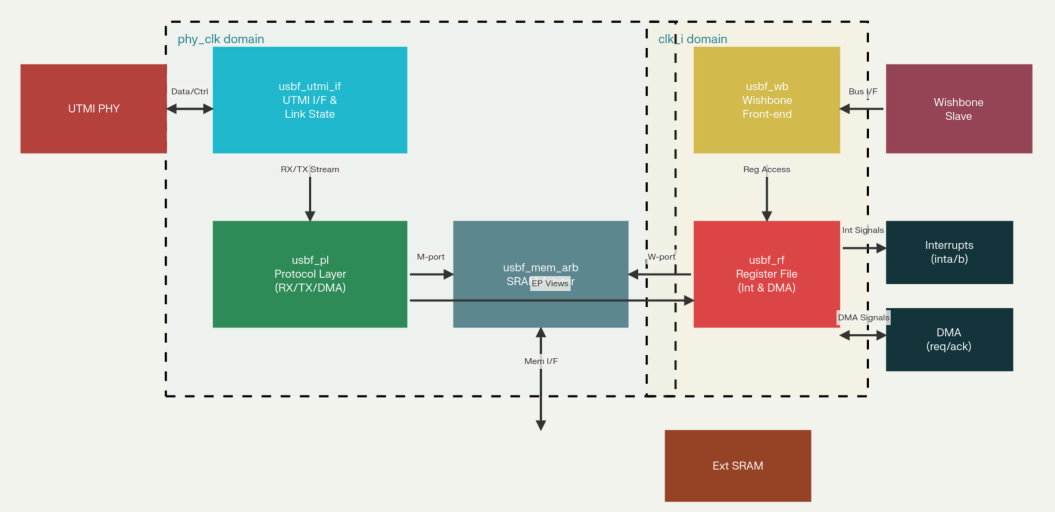
- Integration/functional limits
- No power management beyond suspend/resume signaling; no PHY power control hooks beyond UTMI signals.
- No error correction/ECC on buffer memory; data integrity relies on USB CRCs only.
- No built■in descriptor ROM or firmware; the core requires software to configure endpoints, buffers, and handle control requests.
- Vendor■specific UTMI register interface is pass■through only; correctness depends on the external PHY's vendor extension behavior.

-- Acronyms and Notation --

- USBF: USB Function core (this IP).
- UTMI: USB 2.0 Transceiver Macrocell Interface (physical layer interface signals: XcvSelect, TermSel, SuspendM, OpMode, LineState, DataIn/Out, TxValid/Ready, RxValid/Active/Error).
- WB / WISHBONE: System bus interface used for register and buffer memory access (wb_* signals).
- EP: USB endpoint (EP0–EP15). Presence controlled by macros USBF_HAVE_EPn.
- CSR: Control/Status Register; per-endpoint configuration/status word (csr bus in endpoint register file).
- DMA: Direct Memory Access; in-core mover between UTMI side and shared SRAM (dma_req/dma_ack, idma module).
- IDMA: Internal DMA engine (module usbf_idma) handling memory bursts for IN/OUT payloads.
- SSRAM: Synchronous SRAM used for endpoint buffers; address width parameter SSRAM_HADR.
- RF: Register File (module usbf_rf), includes endpoint CSRs, buffers descriptors, interrupts, and vendor/PHY control.
- PL: Protocol Layer (module usbf_pl): integrates packet decode/assemble, protocol engine, and IDMA.
- PD: Packet Decoder (module usbf_pd): decodes incoming UTMI stream, PIDs, tokens, data, CRCs.
- PA: Packet Assembler (module usbf_pa): forms outgoing packets (handshake/data) and CRC16.
- PE: Protocol Engine (module usbf_pe): endpoint state machine handling tokens, data, handshakes, buffer updates, and timeouts.
- CRC5/CRC16: USB token/data CRC calculators (modules usbf_crc5/usbf_crc16). Bit-reversal conventions applied at interfaces per USB spec.
- UTMI-LS: UTMI Link-State and speed-negotiation controller (module usbf_utmi_ls); manages FS/HS modes, reset, suspend/resume, K/J/SE0 detection.
- PIDs: USB packet identifiers. Token: OUT, IN, SOF, SETUP, PING. Data: DATA0, DATA1, DATA2, MDATA. Handshake: ACK, NACK, STALL, NYET. Special: PRE, ERR, SPLIT. Macros USBF_T_PID_* define 4-bit types; transmitted with inverted upper nibble.
- HS/FS: High-Speed / Full-Speed modes (mode_hs flag selects behavior and timing constants).

- SOF: Start Of Frame token; frame number captured in frm_nat.
- frm_nat: Frame status word combining micro-frame count, last frame number, and SOF timing counters.
- fa: USB Function Address.
- nse_err: Non-sequential error (protocol/stream sequencing).
- pid_cs_err: PID checksum error (upper/lower nibble mismatch).
- crc5_err / crc16_err: Token/data CRC errors.
- Interrupts: inta/intb aggregate endpoint and RF events; per-EP masks and status in RF.
- Buffer descriptors: buf0, buf1 are double-buffer descriptors per EP.
- Fields: [31]=NA (not allocated), [30:17]=buffer size (bytes), [SSRAM_HADR+2:0]=byte address, [3:0]={data_pid_state,next_bsel or low addr nibble} per context.
- uc_bsel: software/engine buffer select state (round-robin when not DMA).
- uc_dpd: data PID state tracking.
- buf0_set/buf1_set/buf0_rl: update or release buffer descriptors; out_to_small indicates short OUT packet in DMA mode.
- DMA sizing signals: dma_in_buf_sz1 (IN path has at least one max packet buffered), dma_out_buf_avail (OUT path has space for next max packet).
- Memory arbitration: usbf_mem_arb arbitrates between packet layer (m*) and WB access (w*); sram_* are external SRAM ports.
- Wishbone address selection macros: USBF_RF_SEL selects RF space; USBF_MEM_SEL selects buffer memory space based on a high address bit. USBF_UFC_HADR defines WB address MSB index.
- Handshake/flow signals conventions:
- *_req/*_ack: request/acknowledge handshake (e.g., dma_req/dma_ack, mreq/mack).
- *_we/*_re: write-enable/read-enable.
- *_din/*_dout: data into/out of a module or memory port.
- *_adr: address bus; [SSRAM_HADR+2:0] are byte addresses; word address is [SSRAM_HADR:0].
- rd_next: packet assembler requests next byte.
- Clock/reset domains:
- phy_clk: UTMI/PHY clock domain.
- clk_i (wclk/wb_clk): system/Wishbone clock domain.
- rst: active-high reset; default synchronous. If USBF_ASYNC_RESET is defined, selected always blocks use asynchronous reset.
- Naming suffixes (register/combination hints):
- _r: registered (flop) version of a signal; _r1/_s1/_s2 staged pipeline taps.
- _d: combinational next-state/value.
- _set/_clr: set/clear strobes; _hold: retention qualifier.
- _sel: select line; _ena/_en: enable; _na: not available/allocated.
- Timing macros (relative to PHY clock):
- USBF_HMS_DEL, USBF_RX_ACK_TO_VAL_{FS,HS}, USBF_TX_DATA_TO_VAL_{FS,HS} define protocol timeouts.
- USBF_T1_* and USBF_T2_* constants implement USB suspend/resume/reset timing per spec (e.g., 2.5us, 3.0ms, 100us, 1.0ms).
- Line-state notation (UTMI):
- J/K/SE0/SE1: logical bus states from LineState[1:0]; ls_j, ls_k, ls_se0 are used internally; idle is J (FS) or SE0 (HS).
- Address/bit indexing: [MSB:LSB] Verilog slice notation is used consistently; numeric literals use 'h hex and 'd decimal; underscores separate thousands in hex fields for readability.
- Vendor I/F: VControl/VStatus are vendor-specific UTMI sideband controls exposed in RF; VControl_Load pulses a write.

ARCHITECTURE



-- Top-Level Block Overview --

usbf_top is a USB 2.0 device (function) core integrating a UTMI+ PHY interface, a packet/protocol/data-path, a shared external SRAM buffer subsystem, and a Wishbone slave control interface. It targets full-/high-speed operation and supports up to 16 endpoints (EP0 always present; others compile-time selectable). Major blocks and roles: (1) UTMI interface (usbf_utmi_if) bridges the UTMI+ PHY to the core, handling TX/RX strobes and 8-bit data, and embeds a line-state controller (usbf_utmi_ls) for attach/reset, suspend/resume, and HS/FS speed negotiation, driving XcvSelect/TermSel/OpMode and producing mode_hs, usb_reset, usb_suspend, usb_attached. (2) Protocol Layer (usbf_pl) runs on the PHY clock and composes the data path from: packet decode (usbf_pd), packet assemble with CRC16 (usbf_pa), protocol engine (usbf_pe) implementing token/data/handshake handling, endpoint buffer selection, PID sequencing, error and timeout handling, and internal DMA (usbf_idma) that moves payloads between the UTMI stream and external SRAM using 32-bit accesses. It also computes frame/sof timing (frm_nat) and exposes endpoint selection and status updates (idin, buf0/1 control strobes, interrupt cause flags). (3) External SRAM arbiter (usbf_mem_arb) arbitrates the single 32-bit synchronous SRAM port between the protocol-layer DMA (PHY clock domain) and Wishbone accesses (system clock domain), presenting sram_adr_o/sram_data_o/sram_we_o/sram_re_o to the memory. (4) Register File and Endpoint Control (usbf_rf) provides the main CSR block (device/line status, function address), per-endpoint CSRs and buffer descriptors (buf0/buf1), consolidated interrupt generation (inta_o/intb_o) from endpoint and core sources, and per-endpoint external DMA handshake (dma_req_o/dma_ack_i). It also exposes a vendor UTMI sideband interface (VControl/VStatus). (5) Wishbone bridge (usbf_wb) is a Wishbone B3-compatible slave that decodes address space into register-file vs buffer memory regions, muxes read data, generates acks, and issues memory or RF transactions; its requests arbitrate through usbf_mem_arb to the external SRAM. Top-level interfaces: - System/Wishbone: clk_i, rst_i, wb_* for control/config and buffer memory access. - UTMI+ PHY: DataOut/TxValid/TxReady, DataIn/RxValid/RxActive/RxError, XcvSelect, TermSel, SuspendM, OpMode, LineState, usb_vbus, with resume_req_i input and susp_o output (mirrors usb_suspend). - External SRAM: 32-bit data, address width parameterized by SSRAM_HADR. - Interrupts/DMA: inta_o, intb_o; 16-bit dma_req_o/dma_ack_i (per endpoint). Clocks/resets and CDC: Two main domains—phy_clk_pad_i (UTMI/packet path and

DMA) and clk_i (Wishbone/control). Reset rst_i is distributed to both; cross-domain handshakes are localized in usbf_utmi_if, usbf_wb/usbf_rf, and usbf_mem_arb. Data/control flow summary: UTMI RX/TX streams terminate in usbf_pl; protocol engine selects endpoint and buffer (buf0/buf1) per endpoint CSR, coordinates DMA bursts via usbf_idma into external SRAM through usbf_mem_arb. Endpoint and core status/control live in usbf_rf, which also aggregates interrupts and issues external DMA requests. Software configures the core and endpoints via Wishbone (register space) and can read/write packet buffers via the memory space.

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

Overview
- The core implements a UTMI-based USB Full-/High-Speed function with a split, pipelined RX/TX data path. The physical interface (UTMI) is terminated in the PHY clock domain; packet decode/encode, protocol control, and DMA operate in the PHY domain; endpoint registers and the Wishbone interface operate in the system (WB) clock domain. A memory arbiter shares a single-port SSRAM between protocol/DMA and the WB host.

UTMI Interface and Line-State Control
- usbf_utmi_if bridges UTMI signals to internal byte-streams: RxValid/Active/Error + DataIn become rx_valid/active/err + rx_data; tx_data/tx_valid/tx_valid_last and tx_first are translated to DataOut/TxValid with TxReady backpressure.
- usbf_utmi_ls manages HS/FS mode selection, reset, suspend/resume, and UTMI control pins (XcvSelect, TermSel, SuspendM, OpMode). It can drive K-chirps (drive_k) and stretches TxValid for resume/keep-alive signaling.

Receive (RX) Data Path
1) Packet Decode (usbf_pd)
- Captures PID, validates PID checksum, parses tokens (address/endp), checks CRC5, extracts frame number (SOF), and streams payload bytes.
- Outputs: token_fadr/endp, token_valid, crc5_err, rx_data_st (byte), rx_data_valid, rx_data_done, crc16_err, seq_err.
2) Protocol Match and Control (usbf_pl + usbf_pe)
- Address match (fsel) compares token_fadr to the programmed function address; endpoint match comes from usbf_rf based on token_endp.
- usbf_pe detects transaction type (IN/OUT/SETUP/CTRL), evaluates endpoint CSR (direction/type/stall/enable, DMA enable, max packet), and buffer availability. It raises send_token (ACK/NAK/STALL/NYET) and controls DMA for data reception (rx_dma_en).
- Buffer bookkeeping: selects buf0/buf1, computes new address/size, generates idin (descriptor/status writeback), and asserts buf0_set/buf1_set/uc_bsel_set/uc_dpd_set or buf0_rl to release/rotate buffers. Error/flow control: to_small/to_large, out_to_small (HS), seq/CRC timeouts.
3) DMA Write (usbf_idma)
- On rx_dma_en, assembles incoming bytes into 32-bit words and issues SSRAM writes via mreq/mwe/madr/mdout. Tracks received byte count (sizu_c) and end-of-packet (idma_done). Handles partial-word finalization.
4) Memory Arbitration (usbf_mem_arb)
- Grants the PHY-side DMA access to SSRAM, arbitrating with WB-side accesses. Read data returns to DMA; writes go straight to memory.

Transmit (TX) Data Path
1) Protocol Decision (usbf_pe)
- For IN or control-IN, checks buffer availability and max packet. Asserts tx_dma_en and optionally send_zero_length. Selects data PID sequencing per transfer type, speed, and CSR.
2) DMA Read (usbf_idma)

- On tx_dma_en, prefetches 32-bit words from SSRAM (pipelined MEM_RD states), presents byte stream tx_data_st, and honors rd_next backpressure. Asserts send_data while bytes remain or for ZLP as needed.
3) Packet Assemble (usbf_pa)
- Emits data or immediate handshake tokens. Chooses token/data PIDs (token_pid_sel/data_pid_sel), computes/streams CRC16 (usbf_crc16), controls tx_first/tx_valid/tx_valid_last, and generates rd_next to pull data from DMA.
4) UTMI Egress (usbf_utmi_if)
- Drives DataOut/TxValid toward the PHY, merges resume drive_k when required, and observes TxReady for pacing.

Endpoint/Register File and Interrupts (usbf_rf)
- Hosts per-EP CSR and buffer descriptors (buf0/1: address, size, valid). Provides match, csr, buf0/1 to the protocol engine, aggregates per-EP DMA request lines, and generates interrupts (A/B) for buffer events, PID/CRC/timeout conditions, attach/reset/suspend/resume, and vendor/line-state updates.

Clock/Domain Boundaries
- PHY clock domain: UTMI I/F, packet decode/encode (pd/pa), protocol engine (pe), DMA (idma), memory arbiter PHY-port, and parts of rf.
- WB clock domain: Wishbone slave (usbf_wb), memory arbiter WB-port, rf register access/interrupt presentation. Handshakes and simple synchronizers are used for cross-domain req/ack and status.

-- Control Path and Register File --

Overview
- The control path is split between a WISHBONE bus front-end (usbf_wb), a register file and endpoint control block (usbf_rf and usbf_ep_rf[0..15]), and the UTMI link-state controller (usbf_utmi_ls via usbf_utmi_if). The register file exposes all software-visible control/status, interrupt, endpoint configuration, buffer descriptors, and vendor UTMI hooks.

WISHBONE access and address decoding
- WISHBONE interface: usbf_wb arbitrates between register-file (RF) and buffer memory windows. It asserts rf_we/rf_re for RF accesses and ma_* for memory.
- Selection: RF is selected when wb_addr_i[12] == 0 (USBF_RF_SEL); memory when wb_addr_i[12] == 1 (USBF_MEM_SEL). Address bus width is A12:0.
- Data mux: wb_data_o returns rf_din when RF selected, otherwise ma_din.
- Acknowledge: wb_ack_o is a single-cycle pulse per access. RF read data is valid in the same WISHBONE cycle that wb_ack_o asserts.

Register file top map (usbf_rf)
- RF address presented as adr[6:0] = wb_addr_i[8:2] (word-aligned). Within an endpoint, adr[1:0] selects the sub-register.
- Top-level words (adr[6:2] = 0x00..0x03): multiplexed via adr[2:0]
- 0x00 (Main status CSR, RO; write used for resume request):
- [4:3] LineState (UTMI)
- [2] usb_attached
- [1] mode_hs (1=HS, 0=FS)
- [0] suspend
- Write side-effect: writing this word uses bit [5] as rf_resume_req latch (set on write, auto-clear when consumed). Other written bits are ignored.
- 0x01 (Function address, R/W): [6:0] funct_adr
- 0x02 (Interrupt masks, R/W): [24:16] intb_msk[8:0], [8:0] inta_msk[8:0]

- 0x03 (Interrupt sources, RO with read-to-clear for RF sources):
- [27:16] int_srca[15:0] = (epN_inta | epN_intb) per endpoint N
- [11:3] int_srcb[8:0] RF-level sources (see Interrupts)
- Read of this word clears int_srcb bits; int_srca is not latched and does not clear on read.
- 0x04 (Frame/time, RO): frm_nat = {mframe_count[3:0], 1'b0, frame_no[10:0], 4'h0, sof_time[11:0]}
- 0x05 (UTMI vendor, mixed RW):
- Read: [7:0] utmi_vend_stat (sampled)
- Write: [3:0] utmi_vend_ctrl; writing also sets utmi_vend_wr strobe until consumed in the PHY clock domain.
- Endpoint windows (adr[6:2] = 0x04 + N for endpoint N=0..15): sub-register select by adr[1:0]
- +0 (EP CSR, R/W): 32-bit control/status per endpoint (see Endpoint CSR)
- +1 (EP INT, R/W): read shows {IEN masks and pending status}; write updates masks only
- Read side-effect: clears local int_stat[6:0] for the selected EP
- +2 (EP BUF0, R/W): buffer descriptor 0 (see Buffers)
- +3 (EP BUF1, R/W): buffer descriptor 1 (see Buffers)

Endpoint CSR (per usbf_ep_rf, exposed in usbf_rf)
- CSR layout (MSB..LSB): {uc_bsel[1:0], uc_dpd[1:0], csr1[12:0], 1'b0, ots_stop, csr0[12:0]}
- Fields consumed by the Protocol Engine (usbf_pe):
- [31:30] uc_bsel: software-visible current buffer select; updated by HW at end of transfers
- [29:28] uc_dpd: software-visible current DATA PID state; updated by HW
- [27:26] endpoint direction: 01=IN, 10=OUT, 00=CONTROL
- [25:24] transfer type: 01=isochronous, 10=bulk, 00/others as design-specific
- [23:22] endpoint state: 00=enabled, 01=disabled, 10=stall
- [17] lrg_ok: accept larger-than-MPS receipt allowed
- [16] sml_ok: accept smaller-than-MPS receipt allowed
- [15] dma_en: enable automatic DMA for non-control EPs
- [12:11] tr_fr: HS transaction scheduling (used internally in PID selection)
- [10:0] max_pl_sz: max packet size
- Write behavior: writing +0 uses din[27:15] -> csr1, din[13] -> ots_stop, din[12:0] -> csr0.

Endpoint interrupts (per usbf_ep_rf)
- Read format (+1): {2'h0, iena[5:0], 2'h0, ienb[5:0], 9'h0, int_stat[6:0]}
- Write format (+1): IENB <= din[21:16], IENA <= din[29:24]
- Pending bits int_stat[6:0] (cleared by reading +1):
- [6] OUT too small (DMA OUT short)
- [5] Sequence error (PID/sequence)
- [4] BUF1 update
- [3] BUF0 update
- [2] Unexpected PID
- [1] CRC16 error
- [0] Timeout
- Per-EP interrupt outputs inta/intb are generated by (int_stat AND corresponding IEN mask). usbf_rf ORs all endpoint inta/intb across EP0..EP15, then ORs with RF-level sources to form top-level inta_o/intb_o.

RF-level interrupts (usbf_rf)
- Sources int_srcb[8:0] (latched, auto-clear on read of top 0x03):
- [8] USB reset detected
- [7] RX error
- [6] Detach
- [5] Attach

- [4] Suspend exit
- [3] Suspend entry
- [2] No selected endpoint match (nse_err)
- [1] PID check-sum error
- [0] Token CRC5 error
- Masks: inta_msk[8:0], intb_msk[8:0] at top 0x02 control routing into inta_o/intb_o.

Buffers (per endpoint)
- Two 32-bit buffer descriptors: BUF0 (+2) and BUF1 (+3).
- Format (function of SSRAM_HADR; common fields):
- [31] NA: 1 means buffer not allocated/unavailable
- [30:17] Size in bytes for this buffer
- [SSRAM_HADR+2:0] SRAM start address (word-aligned; low bits may be overloaded on HW update)
- [3:0] Low nibble used by HW to carry either {next_dpid[1:0], next_uc_bsel[1:0]} when scheduling, or address low bits when HW advances the pointer (HW writes). Software writes should initialize address and size; HW will update size, address, and uc state as transfers complete.
- HW side-effects:
- After successful transfers, HW either rotates uc_bsel/uc_dpd (no DMA) or advances address/size (DMA), asserting buf0_set/buf1_set to commit updates.
- When a full buffer is consumed with DMA enabled, HW can release BUF0 (buf0_rl) to original value buf0_orig.

DMA handshake (exposed via usbf_rf)
- dma_req[15:0]: one request line per endpoint, asserted when dma_en=1 and HW determines data is ready (IN) or space is available (OUT). Cleared on dma_ack[n] unless hold conditions apply (internal counters prevent underrun/overflow).
- dma_in_buf_sz1/dma_out_buf_avail: per-EP indicators exported (via match selection) to the protocol engine for flow control decisions.

Protocol/PHY control hooks
- rf_resume_req (set via writing top 0x00 bit[5]) requests resume signaling; it is generated in clk_i domain and synchronized into the PHY domain (exposed to top-level; system integration determines how it is applied).
- UTMI vendor hook at top 0x05: utmi_vend_ctrl[3:0] and utmi_vend_wr strobe (W), utmi_vend_stat[7:0] (R).

Access side-effects summary
- Reading EP INT (+1) clears that EP’s int_stat.
- Reading top interrupt sources 0x03 clears RF-level int_srcb latched bits.
- Writing EP CSR (+0) updates control fields; writing masks (+1) updates IEN only.
- Writing BUF0/BUF1 sets descriptors; HW may later modify these descriptors on transfer completion.
- Writing top 0x00 with bit[5]=1 sets rf_resume_req (auto-clears on consumption).

Clocking and reset
- Control/CPU interface (RF and masks) operate on wclk (wb_clk). Endpoint state machines and data-path operate on clk (phy_clk). Cross-domain handshakes (e.g., dma_req/ack, vendor strobe, resume) are synchronized internally.

-- Protocol Engine (PD/PE/PA) Architecture --

Overview

- The Protocol Engine (PE) complex comprises three tightly-coupled blocks inside usbf_pl: PD (Packet Decoder), PE (Protocol/transaction state machine), and PA (Packet Assembler). It implements USB token/data/handshake reception and transmission, endpoint policy and buffer management, PID sequencing, DMA buffer servicing, and protocol timeouts/handshakes for Control, Bulk, and Isochronous transfers in both FS and HS modes.

Composition and roles
- PD (usbf_pd): Decodes incoming UTMI RX stream into PIDs, token fields, and data; validates CRC5/CRC16; generates aligned data stream for memory; flags sequencing errors.
- PE (usbf_pe): Transaction controller that decides IN/OUT processing, handshakes (ACK/NACK/STALL/NYET), PID toggling, buffer selection/updates, DMA enablement, and interrupt/event reporting; enforces endpoint policies from CSR.
- PA (usbf_pa): Serializes outbound handshakes/tokens/data, inserts DATA PID per PE policy, computes/appends CRC16, issues tx_first/tx_valid/tx_valid_last.

External context and gating
- Functional address filter (fsel) is token_fadr==fa and further gated: match_o = !pid_bad & fsel & match & token_valid & !crc5_err, where match (from RF) indicates endpoint number match.
- HS/FS mode selection provided by UTMI LS block; PE behavior (e.g., NYET/PING, timeouts) depends on mode_hs.

Key interfaces
- PD outputs: pid_* flags, token_fadr[6:0], token_endp[3:0] (as ep_sel), token_valid, pid_cks_err, crc5_err, rx_data_st[7:0], rx_data_valid, rx_data_done, crc16_err, seq_err.
- PE controls to PA/IDMA: send_token, token_pid_sel[1:0], send_data, data_pid_sel[1:0], send_zero_length, rx_dma_en, tx_dma_en, abort; memory/size: adr[SSRAM_HADR+2:0], size[13:0], buf_size[13:0], sizu_c[10:0]; endpoint updates idin[31:0] and strobes buf0_set/buf1_set/uc_bsel_set/uc_dpd_set/buf0_rl; interrupt set strobes int_to_set/int_crc16_set/int_seqerr_set/int_upid_set and flow flags nse_err/out_to_small.
- PA I/O: tx_data, tx_valid/tx_valid_last, rd_next, consumes tx_data_st from IDMA.
- IDMA (inside PL) bridges PE to external SRAM via usbf_mem_arb, using adr/size, rx/tx DMA enables, rd_next backpressure, and madrs/mdin/mdout/mreq/mack/mwe.

Packet Decoder (PD) details
- FSM: IDLE → ACTIVE → TOKEN or DATA; returns to IDLE on end-of-packet or error.
- PID capture and validation: pid_cks_err if upper/lower nibble mismatch; classifies TOKEN (OUT/IN/SOF/SETUP/PING), DATA (DATA0/1/2/MDATA), HANDSHAKE (ACK/NACK/STALL/NYET) and others.
- Token parsing: token0/token1 latched; extracts function address (token_fadr), endpoint (token_endp), and CRC5; crc5_err asserted if computed (~CRC5) mismatches received.
- Data path: 3-byte shift pipeline (d0→d1→d2) produces rx_data_st with rx_data_valid; CRC16 runs over received bytes (init FFFFh) and is checked against 800Dh at data_done (crc16_err). seq_err flags protocol sequencing faults (unexpected transitions).

Protocol Engine (PE) details
- Endpoint policy fields from CSR (selected by current endpoint match):
- Direction/type: csr[27:26] (CTRL=00, IN=01, OUT=10), transfer type csr[25:24] (ISO=01, BULK=10, INT=00), halt/disable csr[23:22] (01=disabled, 10=stall), size policy csr[17] (large OK), csr[16] (small OK), DMA enable csr[15] (only for non-CTRL), max packet size csr[10:0], software PID bits uc_dpd csr[29:28], buffer select uc_bsel csr[31:30].
- Buffer model: Two buffer descriptors per EP (buf0, buf1) with fields [31]=NA, [30:17]=size, [SSRAM_HADR+2:0]=address, [3:2]=next DATA PID, [1:0]=next buffer select. PE selects active buffer

based on CTRL/in_token or uc_bsel and buffer availability; checks NA bits and space.
- FSM states: IDLE, TOKEN, IN, IN2, OUT, OUT2A, OUT2B, UPDATEW, UPDATE, UPDATE2.
- IDLE: On match_r and !ep_disabled and !pid_SOF: decide transaction:
• If ep_stall→send STALL.
• If both buffers NA or no DMA capacity or CTRL buffer missing→send NACK.
• If HS PING→send ACK.
• If IN (EP IN or CTRL+IN): enable tx_dma_en (and send_zero_length if max_pl_sz==0) → IN.
• If OUT (EP OUT or CTRL+(OUT/SETUP)): enable rx_dma_en → OUT.
- IN: Wait for IDMA completion; if ISO, update immediately; else proceed to IN2 to await ACK within rx_ack_to.
- IN2: If token_valid&&pid;_ACK→UPDATE; else on rx_ack_to→IDLE.
- OUT: Receive payload; on tx_data_to/crc16_err/abort→IDLE. On rx_data_done: if ISO, optionally set seqerr and go UPDATEW; else proceed to OUT2A.
- OUT2A/OUT2B: Final checks; if to_small/to_large→NACK. If pid_seq_err→ACK (complete the transaction). Else ACK (or NYET in HS when no_bufs), then UPDATE.
- UPDATEW→UPDATE: commit buffer pointer/size update; if buffer completed and DMA enabled, issue buf0_rl (release) else write back updated descriptor to selected buffer; UPDATE2 latches uc_bsel/uc_dpd.
- PID sequencing: Computes this_dpid (expected incoming DATA PID) and next_dpid for subsequent transactions, per transfer type, mode_hs, and uc_dpd; compares against received DATA PID (pid_seq_err) and rotates sequence for ISO/BULK per USB rules.
- Size/address arithmetic: size_next=min(buf_size,max_pl_sz); new_size=buf_size–new_sizeb; new_adr=old_adr+size increment (packet size or sizu_c). Checks buffer_done (IN: empty; OUT: full). Detects overflow (buffer_overflow when sizu_c>buf_size during OUT).
- DMA interplay: dma_en=csr[15] & !CTRL_ep; when set, uses IDMA for payload movement; uses dma_in_buf_sz1/dma_out_buf_avail to throttle IN/OUT respectively; issues abort on overflow, mid-transaction retarget, or policy violations.
- Timeouts: rx_ack_to (IN handshake wait) uses USBF_RX_ACK_TO_VAL_FS/HS; tx_data_to (OUT data availability) uses USBF_TX_DATA_TO_VAL_FS/HS.
- Handshake selection encoding for PA: token_pid_sel (ACK=0, NACK=1, STALL=2, NYET=3); data_pid_sel reflects chosen DATA PID.
- Interrupt/event strobes (to RF):
- int_to_set on timeouts (IN2 or OUT), int_crc16_set on rx_data_done&crc16;_err, int_seqerr_set on OUT iso or detected sequencing fault, int_upid_set on unexpected PID for active EP, plus buffer completion int_buf0_set/int_buf1_set; nse_err for no-EP-match; out_to_small on HS OUT short packet under DMA.
- Descriptor update bus (idin):
- idin[31:17] = {buffer_done,new_size} or {4'h0,sizu_c} on out_to_small,
- idin[SSRAM_HADR+2:4] = new_adr (or original on out_to_small),
- idin[3:0] = {next_dpid,next_bsel} or preserved low bits when writing updated buffer address.

Packet Assembler (PA) details
- FSM: IDLE, DATA, CRC1, CRC2, WAIT (zero-length path).
- Builds outbound bytes: chooses between handshake/token (send_token|send_token_r) or data path (tx_data_st); first data byte is DATA PID; appends CRC16 (computed on-the-fly, init FFFFh, output reflected/inverted per USB).
- Control: tx_first asserted on start of token or data; tx_valid_last asserted with token or the final CRC byte; rd_next strobes IDMA when next data byte is consumed; supports zero-length packets (sends only DATA PID+CRC).

Error/exception handling
- pid_bad masks non-addressable PIDs (ACK/NACK/STALL/NYET/PRE/ERR/SPLIT and PING in FS), preventing PE match_o.

- abort sources: buffer overflow, concurrent re-match mid-transaction, policy oversize (to_large) and other checks; abort returns state to IDLE and deasserts DMA.

Timing and compliance
- CRC5/CRC16 computed per USB2.0; CRC16 good value check is 0x800D at EOP.
- HS features: PING handling (ACK response), NYET on OUT when no buffers, separate timeout values, out_to_small flag for non-full HS OUT DMA bursts.

Data/memory path
- For OUT: PD→IDMA (rx_data_st/valid)→SSRAM via arbiter; PE updates/release descriptors.
- For IN: SSRAM→IDMA→PA; PE arms DATA PID, zero-length if max_pl_sz==0; waits for ACK (non-ISO).

Integration in usbf_pl
- PD, PE, PA, and IDMA are composed; frm_nat (frame and micro-frame timing) is generated for RF diagnostics but does not alter PE decisions; tx_valid_out (from UTMI IF) feeds back to PE for ACK-wait timing.

Reset and clocking
- All state machines are synchronous to phy_clk (USB clock); reset is rst. Async reset support is wrapped by `USBF_ASYNC_RESET defines in RTL.

-- Endpoint Architecture and CSR/Buffer Structure --

Endpoints 0..15 are implemented by identical endpoint register-file instances (EP0 is always present; EP1..EP15 presence is compile-time selectable). Each endpoint exposes four 32■bit registers in its window: 0x0=CSR, 0x4=INT, 0x8=BUF0, 0xC=BUF1. The SoC register file (usbf_rf) maps endpoint windows sequentially: window index 0x04+N (N=0..15) selects endpoint N; within a window, adr[1:0] selects CSR/INT/BUF0/BUF1.

Endpoint CSR (EPx.CSR, read/write except noted) is a 32■bit composite: [31:30] uc_bsel (current SW/PE buffer select, updated by hardware when uc_bsel_set), [29:28] uc_dpd (current SW/PE data PID state, updated by hardware when uc_dpd_set), [27:26] EP type/direction: 00=Control, 01=IN, 10=OUT, [25:24] transfer type: 01=Isochronous, 10=Bulk (other values reserved), [23:22] EP state: 00=enabled, 01=disabled, 10=stall, [21:18] endpoint address (matches token ep_sel), [17] large packet allowed (lrg_ok), [16] small packet allowed (sml_ok), [15] DMA enable (ignored/forced off for Control), [14] reserved (reads 0), [13] ots_stop (Out■To■Small monitor enable; when set, hardware writes a status code into csr1[8:7] on an out_to_small event), [12:11] tr_fr (HS transaction framing; determines DATA PID progression in HS), [10:0] max_pl_sz (11■bit maximum packet size). Firmware writes CSR as one word (bits [27:15],[13],[12:0]); uc_bsel/uc_dpd are updated by hardware via uc_bsel_set/uc_dpd_set using idin.

Endpoint INT (EPx.INT): read returns {2'b0, iena[5:0], 2'b0, ienb[5:0], 9'b0, int_stat[6:0]}. Write (we1) programs masks: ienb<=din[21:16], iena<=din[29:24]. Read of INT (re on adr==1) clears int_stat. Per■endpoint interrupt sources (int_stat) set when the addressed endpoint matches the current token: [6] out_to_small (OUT size != max_pl_sz with DMA), [5] int_seqerr_set (DATA PID sequencing error), [4] int_buf1_set (buffer 1 updated/completed), [3] int_buf0_set (buffer 0 updated/completed), [2] int_upid_set (unexpected PID for EP type/state), [1] int_crc16_set (CRC16 error on received data), [0] int_to_set (ACK/DATA timeout). INTA/INTB outputs per endpoint are the masked ORs of these bits with iena/ienb. Device■level interrupt aggregation and masks live in the usbf_rf top scope (separate from per■endpoint INT).

Endpoint buffers (EPx.BUF0/BUF1) are 32■bit descriptors for the shared 32■bit wide SSRAM buffer space: [31] NA (not available flag; if set, buffer is treated as unavailable), [30:17] size (14■bit buffer size in bytes), [SSRAM_HADR+2:0] buffer start address (byte address; lower two bits are byte offsets), [3:0] low nibble used by hardware either to carry address low bits on updates or to store {next_dpid[1:0], next_bsel[1:0]} when the protocol engine advances PID/buffer state. Software initializes BUF0/BUF1 (both the address and size). Hardware may update a buffer after a transfer as follows: on UPDATE, if the transfer did not consume the entire buffer and DMA is disabled, the engine writes back new_size and new_adr and rotates uc_bsel; if DMA is enabled and the buffer is fully consumed, buf0_rl triggers to reload BUF0 from its original copy (buf0_orig). For OUT endpoints with DMA, out_to_small causes a status writeback: idin[31:17]<={4'h0,sizu_c} and address low nibble preserved; for normal updates idin[31:17]<={buffer_done,new_size} and idin low nibble is either new_adr[3:0] (when updating address) or {next_dpid,next_bsel} (when only PID/buf select advances). The NA condition is detected as bit[31]==1 or address field all■ones; such buffers are skipped/NACKed as appropriate.

Endpoint match and operation: An endpoint matches an incoming token when ep_sel (from token) equals CSR[21:18]. For matched endpoints, the protocol engine (PE) drives: buf0_set/buf1_set (write idin to the selected buffer descriptor), buf0_rl (reload BUF0 from its original value), uc_bsel_set/uc_dpd_set (update CSR[31:28] from idin[1:0]/[3:2]), and event bits that raise int_stat. Data flow uses IDMA + memory arbiter into a unified SSRAM; per■endpoint DMA request (dma_req[n]) asserts when: OUT endpoint has queued data to write (dma_out_cnt!=0) or IN endpoint needs to fetch data (dma_in_cnt
Address map summary (relative to each endpoint window): +0x0: CSR, +0x4: INT, +0x8: BUF0, +0xC: BUF1. Endpoint window base indices are contiguous from EP0 at window 0x04. All registers are 32■bit word accessible over the WISHBONE RF space.

-- DMA and Memory Subsystem (SSRAM and Arbiter) --

Overview
- The USB function core uses a single 32■bit, word■addressed, single■port SSRAM as a shared buffer memory for packet payloads and software access. Access is arbitrated between the packet DMA engine (IDMA, running on phy_clk) and a Wishbone memory window (software, wb_clk).

External SSRAM interface
- Width: 32■bit data (sram_din/sram_dout)
- Address: [SSRAM_HADR:0] word address (low 2 address bits are byte selects handled internally by the DMA)
- Controls: sram_we (write enable), sram_re (always 1), sram_adr (word address)
- Parameterization: SSRAM_HADR = `USBF_SSRAM_HADR` (default 9), yielding 2^(SSRAM_HADR+1) words = (1 << (SSRAM_HADR+1)) * 4 bytes (default 4 KiB)

Arbiter (usbf_mem_arb)
- Ports (IDMA side, phy_clk domain): madr[m], mdout<=sram_din, mdin=>sram_dout, mwe, mreq, mack
- Ports (Wishbone side, wb_clk domain via usbf_wb): wadr[w], wdout<=sram_din, wdin=>sram_dout, wwe, wreq, wack
- Selection: IDMA has priority. Wishbone selected when (wreq | wack) & !mreq; otherwise IDMA selected.
- Handshakes: mack = mreq (immediate acknowledge to IDMA). Wishbone ack wack is issued (on phy_clk) when wreq is seen and no concurrent mreq; wack is held off while mreq is high.
- Address/data mux: sram_adr/sram_dout steer from the selected master; sram_din is broadcast to both masters (mdout, wdout).

- Write enable: sram_we = (wreq & wwe) when Wishbone selected; else sram_we = (mwe & mack) when IDMA selected. Read enable is tied high.
- Clocks: Arbiter internal flops (e.g., wack_r) run on phy_clk. Wishbone requests (wreq/wwe/wadr/wdin) originate on wb_clk; IDMA requests (mreq/…) on phy_clk. The arbiter provides a simple level■based handshake across domains; system integration should ensure wb_clk/phy_clk relationship is acceptable for this scheme.

Wishbone memory window (usbf_wb)
- Address decode: Memory window selected when `USBF_MEM_SEL` (default wb_addr_i[12] == 1). Register file when `USBF_RF_SEL`.
- Mapping: ma_adr (Wishbone address) is truncated to SSRAM word address: wadr = ma_adr[SSRAM_HADR+2:2]. No byte enables; software must perform 32■bit accesses.
- Acknowledgment: A single acknowledge is returned per request when the arbiter grants the bus (may stall while IDMA is active).

DMA Engine (usbf_idma)
- Purpose: Moves packet payloads between UTMI data streams and SSRAM buffers under control of the protocol engine.
- Addressing: IDMA uses byte address adr[SSRAM_HADR+2:0] internally; the arbiter sees word address madr = adr[SSRAM_HADR+2:2]. Little■endian packing/unpacking across the 32■bit word lanes (byte lane selected by adr[1:0]).
- RX path (OUT/SETUP): Assembles 32■bit words from incoming bytes; issues SSRAM writes when a word is complete or at packet end (partial word). Uses mreq/mwe handshakes. Optionally wraps to buffer start when dma_en=1 and the end of the buffer (adr + buf_size) is reached.
- TX path (IN): Prefetches 32■bit words from SSRAM (double■word ping■pong into rd_buf0/rd_buf1) and streams bytes to the packet assembler, asserting rd_next to advance.
- Sizes and flow control: Tracks remaining byte count (sizd_c) and user■visible size (sizu_c). Generates idma_done when the programmed size is exhausted (TX) or end of RX packet is reached.

Concurrency and policy
- Priority: IDMA always preempts Wishbone. While IDMA mreq is asserted, Wishbone memory acks are deferred.
- Latency: IDMA sees mack immediately (mack=mreq), modeling a single■cycle SSRAM access. Wishbone typically receives a one■phy_clk■cycle ack when IDMA is idle; otherwise it waits.

Notes and limitations
- No byte write enables on SSRAM; all software writes are full 32■bit words.
- Read path is always enabled; address selection determines which location is seen.
- The arbiter implements a minimal cross■clock handshake. Deployments with unrelated wb_clk/phy_clk should review CDC robustness or align clocks accordingly.
- Buffer wrap (ring) behavior is enabled only when dma_en=1 for the endpoint.

-- Wishbone Slave Interface Architecture --

The core exposes a 32■bit, classic WISHBONE slave interface with a single address region split into Register File (RF) and Buffer Memory (SSRAM) spaces.

- Bus signals: wb_clk, rst, wb_addr_i[USBF_UFC_HADR:0], wb_data_i[31:0], wb_we_i, wb_stb_i, wb_cyc_i, wb_data_o[31:0], wb_ack_o. No sel_i/cti_i/bte_i/stall_o/err_o/rty_o are used or generated.
- Data width and alignment: 32■bit words; lower address bits [1:0] are ignored internally (word■aligned accesses only). Byte enables are not supported (full■word accesses).
- Address width: parameter USBF_UFC_HADR (final define in this file is 12), so A12:0 are

implemented.
- Region selection (A12):
- A12 = 0 (USBF_RF_SEL): Register File space (function + endpoint registers).
- A12 = 1 (USBF_MEM_SEL): Packet Buffer Memory (SSRAM) space.
- RF address decoding: usbf_rf sees adr[6:0] = wb_addr_i[8:2] (word addressing). RF reads assert rf_re; writes assert rf_we and pass wb_data_i on rf_dout.
- Buffer memory decoding: usbf_mem_arb uses word address wadr = wb_addr_i[SSRAM_HADR+2:2]. SSRAM_HADR (final define is 9) yields 4 KiB total buffer memory ((1<<(SSRAM_HADR+1)) * 4 = 4096 bytes).
- Read data return: wb_data_o multiplexes ma_din (memory) or rf_din (registers) based on the address MSB (A12). Masters should sample wb_data_o when wb_ack_o is asserted.
- Transaction model: classic single transfer. A request is recognized when wb_stb_i & wb_cyc_i is high. The slave returns a single■cycle wb_ack_o pulse when the operation completes. No pipelining or bursts.
- Acknowledge generation and CDC: The WISHBONE front■end (usbf_wb) runs an internal state machine in the PHY clock domain (phy_clk) and synchronizes a single■cycle acknowledge back to the WISHBONE clock domain (wb_clk) using a two■stage pulse synchronizer. RF accesses acknowledge after a short fixed latency; memory accesses acknowledge when the memory arbiter grants the WISHBONE port.
- Memory arbiter behavior: usbf_mem_arb arbitrates between the packet layer DMA port (M) and the WISHBONE port (W). The WISHBONE port is served when no DMA request is active; otherwise, WISHBONE is back■pressured until DMA deasserts its request. WISHBONE memory acks (ma_ack) are produced from the arbiter once the W port is selected.
- Side effects: RF reads may clear internal sticky status bits as defined in usbf_rf; writes update control, endpoint CSRs, and buffer descriptors. Memory operations only access the shared packet buffers.
- Reset: Active high rst (optionally asynchronous if USBF_ASYNC_RESET is defined) resets the WISHBONE interface state machine to IDLE; wb_ack_o is deasserted.
- Timing expectations: RF space typically acknowledges in a few wb_clk cycles (subject to CDC), memory space acknowledges variably depending on DMA activity and PHY clock arbitration.
- Unsupported features: partial■word writes, burst/CTI/BTE cycles, error/retry responses, and stall insertion are not implemented.

-- UTMI Interface and Link State Machine --

The UTMI Interface and Link State Machine provide the on-PHY-clock bridge to a UTMI-compliant USB PHY and implement FS/HS attach, reset, suspend/resume, and high■speed negotiation. They comprise two blocks: usbf_utmi_if (electrical/handshake bridge) and usbf_utmi_ls (link state machine).

usbf_utmi_if (UTMI bridge on phy_clk):
- RX path: registers RxValid, RxActive, RxError and DataIn to rx_valid, rx_active, rx_err, rx_data.
- TX path: drives DataOut with tx_data when TxReady or tx_first; otherwise, if the link state machine requests a K drive (drive_k=1), outputs 0x00. TxValid is asserted when any of: tx_valid, tx_valid_last, or drive_k are true, and it remains asserted until TxReady or the K drive completes (holding rule: TxValid <= tx_valid | tx_valid_last | drive_k | (TxValid & !(TxReady | drive_k_r))). tx_ready mirrors TxReady. tx_first primes the first data byte regardless of TxReady.
- Low-level UTMI controls (outputs to PHY): XcvSelect (1=FS, 0=HS), TermSel (FS termination on/off), SuspendM (asserted while in suspend without a pending resume request, or while LineState==K), OpMode (00=normal/bit-stuff on, 10=non-driving/bit-stuff off), plus mode_hs, usb_reset, usb_suspend, usb_attached. It accepts resume_req (device-initiated resume request) and produces suspend_clr to indicate the core has exited suspend (top-level clears a latched resume request on suspend_clr).

usbf_utmi_ls (link state machine on phy_clk):
- Interprets UTMI LineState (J/K/SE0/SE1) and classifies idle: idle = SE0 in HS, idle = J in FS.

Measures protocol timing using counters parameterized by macros: 250 ns ticks (USBF_T1_PS_250_NS), 2.5 µs, 62.5 µs, 3.0 ms, 3.125 ms, 5.0 ms, 100 µs, 0.5 ms, 1.0 ms, 1.2 ms, 100 ms.
- States and key behavior:
• POR: Initialize FS mode (XcvSelect=FS, TermSel on, OpMode=normal), clear suspend; go to ATTACH.
• ATTACH: After ~100 ms stable presence assert usb_attached and enter NORMAL.
• NORMAL: Monitor idle durations. If (FS) idle SE0 >2.5 µs (reset) enter RESET; if idle >3 ms enter SUSPEND (FS). If HS and idle >3 ms, drop to FS electricals and enter RES_SUSP.
• RES_SUSP: After >100 µs decide RESET (SE0) or SUSPEND (J).
• SUSPEND: Exit to RESET on SE0>2.5 µs. Enter RESUME on K. If idle >5 ms and a resume_req is present, initiate RESUME_REQUEST.
• RESUME_REQUEST: Device-initiated resume: assert FS term, disable bit-stuffing (non-driving), after wakeup window drive RESUME_SIG.
• RESUME_SIG: Drive K (drive_k=1) for >1.0 ms, then go to RESUME; RESUME clears suspend and, once SE0 seen, re-enable normal mode and proceed to RESUME_WAIT; after >100 µs enter NORMAL.
• RESET: Assert usb_reset, force FS electricals and non-driving mode for >1.0 ms, then SPEED_NEG.
• SPEED_NEG / SPEED_NEG_K / SPEED_NEG_J: Perform HS chirp (drive K, count alternating K/J detections). On six K/J toggles enter SPEED_NEG_HS; if SE0 long occurs, fall back to SPEED_NEG_FS.
• SPEED_NEG_HS: Select HS (XcvSelect=HS, TermSel off, bit-stuff on, mode_hs=1). Once bus returns to idle, enter NORMAL.
• SPEED_NEG_FS: Select FS (XcvSelect=FS, TermSel on, bit-stuff on, mode_hs=0), then NORMAL.
- Outputs driven per state: mode_hs (latched FS/HS), usb_reset (one-shot during RESET), usb_suspend (set in SUSPEND, cleared on resume/reset entry), usb_attached (set after ATTACH), XcvSelect/TermSel/OpMode as above, drive_k for resume and chirp signaling, and suspend_clr when leaving suspend.

Integration notes:
- The top-level latches an external resume_req and clears it upon suspend_clr. The UTMI IF/LS logic runs entirely in the PHY clock domain and provides all UTMI control pins, bus state indications, and timing-compliant resume/reset/negotiation sequences required by the USB 2.0 FS/HS protocol.

-- Interrupt Architecture --

Overview
- Two interrupt outputs: inta_o and intb_o (Wishbone clock domain). Each is a level-OR of endpoint interrupts and masked function-level (core) events.

Top-level composition
- inta_o = (OR of ep[0..15].inta) OR (|(int_srcb & inta_msk))
- intb_o = (OR of ep[0..15].intb) OR (|(int_srcb & intb_msk))
- int_srcb[8:0] are function-level latched sources. inta_msk[8:0] and intb_msk[8:0] independently route any function source to A and/or B lines.
- int_srca[15:0] is a live status bitmap: int_srca[n] = epn_inta OR epn_intb (not latched, not maskable; read-only status).

Function-level interrupt sources (int_srcb)
Bit -> Event (latched, clear on read of RF INT_SRC)
- [8] USB reset detected
- [7] UTMI RxError
- [6] Device detach (usb_attached falling edge)

- [5] Device attach (usb_attached rising edge)
- [4] Resume from suspend (suspend_end)
- [3] Enter suspend (suspend_start)
- [2] No-matching-endpoint on addressed token (NSE)
- [1] PID checksum error
- [0] Token CRC5 error

Masking/route
- Write RF INT_MASK to set inta_msk[8:0] and intb_msk[8:0]. If a bit is set in a mask, the corresponding source contributes to that line. A source may drive both lines if both masks are set.

Clearing
- Read RF INT_SRC (see Register access) to clear int_srcb bits (read-to-clear). int_srca is not cleared (it mirrors current per-endpoint line activity).

Per-endpoint interrupt architecture (ep0..ep15)
- Each endpoint implements 7 latched causes in int_stat[6:0] (set in PHY/packet clock domain, reported in Wishbone domain). They are cleared by reading the endpoint INT register (read-to-clear when re asserted and adr==1 within the endpoint block).
- Cause-to-line routing is controlled per endpoint by two 6-bit enable fields written into the endpoint INT register:
- iena[5:0] routes selected causes to the endpoint's A line (epX_inta).
- ienb[5:0] routes selected causes to the endpoint's B line (epX_intb).
- Cause IDs and mask-bit mapping (note that two causes share mask bit 3):
- int_stat[0] Timeout -> mask bit 0
- int_stat[1] CRC16 error -> mask bit 1
- int_stat[2] Unexpected PID (UPID) -> mask bit 2
- int_stat[3] Buffer0 update -> mask bit 3
- int_stat[4] Buffer1 update -> mask bit 3 (shared)
- int_stat[5] Sequence error (ISO OUT) -> mask bit 4
- int_stat[6] OUT packet smaller than max pkt (DMA) -> mask bit 5

Aggregation
- usbf_rf ORs all epX_inta to form inta_ep and all epX_intb to form intb_ep; these are then ORed with function-level masked results to produce inta_o/intb_o.

Register access (RF block, Wishbone A[12]=0 region)
- INT_MASK: sets inta_msk[8:0] and intb_msk[8:0] (write). Addressed via RF adr[2:0]==2.
- INT_SRC: read returns {3'h0, int_srcb[8:0], 4'h0, int_srca[15:0]}. Reading at RF adr==0x03 clears int_srcb (read-to-clear). Endpoints are cleared by reading each endpoint's INT register (adr[1:0]==1 within the endpoint window).
- Endpoint windows: RF adr[6:2]==0x04..0x13 select EP0..EP15; inside each, adr[1:0]==1 is the INT register (read returns {..., iena, ..., ienb, ..., int_stat}; write updates iena/ienb).

Behavior and timing
- inta_o/intb_o are level signals, synchronous to Wishbone clock (wclk). They deassert when all contributing sources are cleared or masked.
- Function-level sources are latched until RF INT_SRC is read. Endpoint causes are latched until the corresponding endpoint INT register is read.
- Software servicing flow: on inta_o/intb_o, read RF INT_SRC to handle/clear function-level events and to discover which endpoint(s) are asserting (int_srca). Then read each active endpoint's INT register to obtain/clear its cause(s).

-- Clock and Reset Domains and CDC Strategy --

Clocking overview
- Two primary, independent clock domains are used:
1) PHY/UTMI domain (phy_clk_pad_i). Hosts the UTMI interface (usbf_utmi_if/usbf_utmi_ls), the packet layer (usbf_pl), packet decoder/encoder (usbf_pd/usbf_pa), packet engine (usbf_pe), the internal DMA (usbf_idma), and the shared SRAM arbiter side for the PHY master (usbf_mem_arb.m*).
2) System/WISHBONE domain (clk_i). Hosts the WISHBONE bridge (usbf_wb), the register file and endpoint register bank front-end and interrupt aggregation (usbf_rf), UTMI vendor control/status registers, and the shared SRAM arbiter side for system accesses (usbf_mem_arb.w* via usbf_wb).

Reset scheme
- A single reset input rst_i is fanned out to both domains as rst. The PHY reset output phy_rst_pad_o is a direct copy of rst_i.
- All modules use rst synchronously to their local clock by default. If the synthesis macro USBF_ASYNC_RESET is defined, many always blocks switch to asynchronous reset (negedge rst). No additional reset synchronizers are inserted between domains; system integration should ensure reset assertion is asynchronous and de■assertion is aligned or otherwise safe for each clock domain.

CDC strategy and placement
- Data-path containment: All time■critical byte and packet processing, and the SRAM data path and arbitration, are kept in the PHY domain. System/WB sees memory through a bridge rather than a dual■clock RAM interface.
- WISHBONE-to-PHY bridge (usbf_wb):
• WB request ingress: WB strobe/cycle (wb_stb_i & wb_cyc_i) are sampled directly into the PHY clock (wb_req_s1) and used by a PHY-clock state machine that generates memory/register cycles. This is a level■sampled crossing (single stage), relying on the request being stable long enough.
• ACK egress back to WB: Acknowledge (wb_ack_d) generated in PHY is synchronized into WB clock with a two■flip■flop chain (wb_ack_s1a/wb_ack_s2), then edge■detected to form wb_ack_o. This implements a standard 2■FF synchronizer for the return pulse.
• Address/data for WB memory cycles: ma_adr is a combinational copy of wb_addr_i and is consumed on the PHY side (via usbf_mem_arb.wadr). Validity relies on the usbf_wb state machine holding the request while address/data remain stable; no explicit multi■FF synchronizer is used on address/data.
- Memory arbitration (usbf_mem_arb): Entirely in PHY clock. The system (WB) side is presented as a PHY■clocked requester (w* signals) driven by usbf_wb; hence the actual SRAM interface has no dual■clock crossing.
- UTMI control/status:
• resume_req: Synchronized into PHY clock within usbf_utmi_ls via a 2■FF synchronizer (resume_req_s1/resume_req_s). The corresponding suspend_clr generated in the PHY domain is sampled in the WB domain without an explicit 2■FF synchronizer and only used to clear resume_req.
• Vendor control writes: Written in WB domain (utmi_vend_ctrl_r / utmi_vend_wr_r) then sampled once into PHY domain (single■FF stage). Intended as slow control; pulses should be wide enough to be seen by the PHY clock.
• Vendor status reads: UTMI vendor status is captured in PHY, then re■sampled once in WB domain (single■FF stage).
- Register file and endpoint register bank (usbf_rf/usbf_ep_rf):
• The endpoint control/status registers and per■endpoint DMA/bookkeeping live in the PHY domain. Interrupt outputs inta/intb are produced in the WB clock domain inside usbf_ep_rf by reading PHY■domain status; this is implemented via direct sampling (single■FF or combinational read across domains), assuming level stability over several cycles.
• Global RF status/interrupt bits (attach, deattach, USB reset, CRC/sequence errors, suspend start/end) are latched in the WB domain by sampling PHY■domain signals using single■FF stages; these are level■type events with clear-on-read.

Assumptions and integration guidance
- The design primarily uses: (a) 2■FF synchronizers for specific control pulses (e.g., resume_req ack path), (b) request/ack level handshakes where one side holds a level until the other domain observes it, and (c) many single■stage, level■sampled crossings for slow■moving status/interrupt/control bits.
- No explicit dual■clock FIFOs are used. No timing relationship between clk_i and phy_clk_pad_i is required by construction, but because several crossings are single■stage, integrators should ensure that level■type signals remain stable for multiple cycles of the receiving clock, or strengthen CDC (add 2■FF synchronizers) if targeting strict CDC sign■off.
- If USBF_ASYNC_RESET is enabled, assertion is asynchronous per block; ensure reset release meets each domain's clocking requirements.

-- Scalability and Parameterized Resources --

The core is designed to scale primarily through two mechanisms: (1) parameterized buffer memory addressing and (2) compile-time selectable endpoints. Key aspects: 1) Buffer memory depth: Parameter SSRAM_HADR (default from `USBF_SSRAM_HADR`) sets the external SRAM address width and scales all related address fields and counters across the DMA path. - External SRAM address bus: sram_adr_o[SSRAM_HADR:0]. - Wishbone-to-SRAM addressing: wadr = ma_adr[SSRAM_HADR+2:2] (word addressing). - Internal buffer descriptors (EP Buf0/Buf1) carry address fields sized to [SSRAM_HADR+2:0]; software must align descriptor packing to the chosen SSRAM_HADR. - Memory size formula (bytes): 4 * 2^(SSRAM_HADR+1). Changing SSRAM_HADR updates usbf_pl/usbf_pe/usbf_idma/usbf_mem_arb consistently via parameter propagation from usbf_top. 2) Endpoint scalability (0..15): EP0 is always present; EP1..EP15 are compile-time selectable via `USBF_HAVE_EPn` macros. Enabled endpoints instantiate usbf_ep_rf; disabled endpoints instantiate usbf_ep_rf_dummy, preserving vector widths and tie-offs. The design enforces contiguous EP enables (1..N) with an initial-time check. DMA interface scales to 16 per-EP request/ack bits: dma_req_o[15:0] and dma_ack_i[15:0] (unused bits are tied off by dummy EPs). 3) Wishbone address scalability: The Wishbone address width is set by `USBF_UFC_HADR`. Memory vs register-file selection is controlled by macros `USBF_MEM_SEL` and `USBF_RF_SEL` (in this file, last defines select wb_addr_i[12]). Adjusting `USBF_UFC_HADR` and these select macros together moves the split point between RF and memory spaces without RTL changes. 4) Mode scalability at runtime: High-Speed vs Full-Speed behavior is handled dynamically (mode_hs) in UTMI/Link state machines and protocol engines; no compile-time sizing impact. 5) Build-time feature toggles: `USBF_ASYNC_RESET`, `USBF_DEBUG`, and `USBF_TEST_IMPL` gate reset style and diagnostics without changing functional resource sizing. 6) Integration considerations when scaling: - If you change SSRAM_HADR, ensure external SRAM depth matches, and update software to encode/decode EP buffer descriptors using the new [SSRAM_HADR+2:0] address field width. - If you enable more endpoints, ensure corresponding DMA ack bits are driven and that system software maps their RF windows. - Keep `USBF_UFC_HADR`, `USBF_RF_SEL`, and `USBF_MEM_SEL` consistent; the last macro definitions take precedence in this file. Default configuration in this source sets SSRAM_HADR=9 (≈4 KiB buffer), `USBF_UFC_HADR`=12, and enables EP1..EP3; others synthesize as dummies.

OPERATION

-- Power-Up, Reset, and Initialization --

Overview
- The core has two clock domains: system (clk_i) and PHY/UTMI (phy_clk_pad_i). A single reset input (rst_i) fans out to all blocks; phy_rst_pad_o mirrors rst_i to the PHY.
- By default, resets are synchronous to their respective clock domains. If USBF_ASYNC_RESET is defined at synthesis, resets become asynchronous.

Reset assertion (rst_i = 1)
- All FSMs return to IDLE/known states. No memory or bus transactions are issued.
- UTMI interface (usbf_utmi_if/usbf_utmi_ls):
- State machine enters POR. XcvSelect is set for Full-Speed (FS), TermSel asserted (FS termination on), mode_hs = 0, usb_attached = 0, usb_suspend = 0.
- TxValid is deasserted; no transmit activity. drive_k = 0.
- suspend_clr is inactive.
- Top-level outputs/flags:
- susp_o reflects usb_suspend (low on reset).
- resume_req_r (internal latch of resume_req_i) is cleared.
- Wishbone interface (usbf_wb):
- State = IDLE; wb_ack_o deasserted. No RF/MEM requests.
- Register file (usbf_rf):
- funct_adr = 0.
- Interrupt masks = 0; all RF interrupt source bits cleared; inta_o/intb_o low.
- Vendor control/output (utmi_vend_ctrl/utmi_vend_wr) cleared.
- rf_resume_req cleared.
- Endpoints (usbf_ep_rf, all implemented EPx):
- CSR fields reset to 0 (dir = control, type = control, DMA disabled csr[15] = 0, max packet = 0, status bits = 0). uc_bsel = 0, uc_dpd = 0.
- buf0 = buf1 = 0xFFFF_FFFF (unallocated). Endpoint-matching/interrupt status cleared. DMA request outputs = 0.
- Protocol/link (usbf_pl, usbf_pe, usbf_pd, usbf_pa, usbf_idma):
- All protocol engines IDLE. No tx/rx activity, no token generation, no DMA.
- Memory subsystem (usbf_mem_arb/usbf_idma):
- No writes; sram_we_o = 0. sram_re_o held high by design but no transactions occur without requests. All DMA address/size counters reset.

Power-up and attachment (VBUS present)
- With usb_vbus_pad_i asserted and rst_i released, the UTMI line-state controller sequences: POR -> ATTACH -> NORMAL.
- In ATTACH it waits approximately 100 ms before asserting usb_attached and entering NORMAL.
- Speed negotiation and reset handling:
- SE0 > ~1 ms causes RESET; controller performs HS chirp and then selects HS or FS, updating mode_hs, XcvSelect, and TermSel accordingly.
- Suspend is detected from extended idle; resume signaling (drive K) is generated as required. On exiting suspend, suspend_clr is pulsed.

Resume request handling
- External resume_req_i is level-latched into resume_req_r and automatically cleared on suspend_clr, ensuring a complete resume sequence.

USB bus reset indication
- Upon detecting a USB bus reset, the RF “core” interrupt source bit (int_srcb[8]) is set and remains set until the software reads the interrupt source register (read-to-clear).

Software initialization sequence (typical)
- After rst_i deassertion and usb_attached asserted (and/or after a USB reset is observed):
1) Program the Function Address (RF register index 0x1).
2) For each used endpoint, program EP CSR (type, direction, max packet), initialize buf0/buf1 descriptors (base address, size, initial DPID/BSEL), and only then set csr[15] = 1 to enable DMA (if used).
3) Configure interrupt masks; clear pending interrupt sources by reading the RF interrupt source register.
4) Ensure the external SSRAM buffer space is initialized before enabling traffic. Uninitialized (0xFFFF_FFFF) buffer descriptors are treated as not allocated and will cause NAK/NYET per protocol engine rules.

Operational guarantees after initialization
- Endpoints NAK until valid buffers are allocated (and for control EP, until appropriate stage is active).
- No outstanding Wishbone or memory transactions exist unless initiated by software or the protocol engine.

Reset requirements
- Hold rst_i asserted until both clk_i and phy_clk_pad_i are stable.
- Reasserting rst_i at any time returns the core to the safe, quiescent defaults described above in both clock domains (sync or async as built).

-- Device Attach/Detach and Speed Negotiation --

Overview
- The USB function core implements device attach/detach detection and high■speed/full■speed (HS/FS) selection/negotiation inside the UTMI line-state controller (module usbf_utmi_ls), accessed through usbf_utmi_if and exposed at the top level (usbf_top).

Key UTMI signals and controls
- Inputs: LineState[1:0] (UTMI bus), usb_vbus, phy clock, rst, resume_req.
- Derived line states: SE0=00, J=01, K=10, SE1=11. Idle is J in FS, SE0 in HS.
- Outputs to PHY: XcvSelect (FS=1, HS=0), TermSel (FS termination), OpMode (bit-stuff enable/disable), SuspendM, drive_k (used for K signaling during resume and HS chirp).
- Status out: mode_hs (1=HS, 0=FS), usb_reset, usb_suspend, usb_attached.

Device attach
- Power-on/reset (POR) entry conditions: rst deasserted or a usb_vbus event forces the state machine into POR.
- POR actions: select FS (XcvSelect<=FS), enable FS termination (TermSel<=1), clear usb_attached, clear suspend, then transition to ATTACH.
- ATTACH timing: wait ~100 ms (macro USBF_T2_C_100_MS via internal counters). After the delay, set usb_attached=1 and transition to NORMAL operation.
- System-visible attach event: usbf_rf edge-detects usb_attached and latches an “attach” interrupt source (int_srcb[5]).

Device detach
- A VBUS change that drives the controller back to POR clears usb_attached to 0.
- System-visible detach event: usbf_rf detects the falling edge of usb_attached and latches a “deattach” interrupt source (int_srcb[6]).

Reset detection and entry to negotiation

- In NORMAL (operational) state, the controller monitors line idle duration using programmable timing (USBF_T1_* macros). When FS and a reset (SE0) condition is observed for the required interval, the state machine enters RESET.
- RESET actions: assert usb_reset, force FS electricals (XcvSelect<=HS select for chirp drive plus TermSel<=1; bit-stuff off), then after ~1 ms (USBF_T2_C_1_0_MS) transition to SPEED_NEG.

High■speed negotiation (chirp) and fallback to FS
- SPEED_NEG: drive K (drive_k asserted) and initialize chirp counting; after ~1.2 ms (USBF_T2_C_1_2_MS) start sampling host chirps.
- SPEED_NEG_K / SPEED_NEG_J: alternate states that count received K/J chirps from the host; each valid opposite-state transition increments chirp_cnt.
- HS entry criterion: on reception of 6 valid K/J chirps (chirp_cnt==6), enter SPEED_NEG_HS.
- SPEED_NEG_HS actions: enable bit-stuffing (OpMode), select HS transceiver (XcvSelect<=HS), disable FS termination (TermSel<=0), set mode_hs=1; then transition to NORMAL when SE0 long is observed.
- FS fallback criterion: if a long SE0 occurs during negotiation before 6 chirps are counted, enter SPEED_NEG_FS.
- SPEED_NEG_FS actions: enable bit-stuffing, select FS transceiver (XcvSelect<=FS), enable FS termination (TermSel<=1), set mode_hs=0; then transition to NORMAL.

Suspend/Resume notes (interaction with attach/negotiation)
- The controller detects idle durations (USBF_T1_* macros) to enter SUSPEND; in suspend it may generate resume signaling on resume_req via RESUME_REQUEST/RESUME_SIG (drive_k asserted for resume K).
- On exit from suspend (RESUME), if in HS the controller restores HS settings; otherwise it restores FS.

Top-level observability and integration
- usbf_top propagates: mode_hs, usb_reset, usb_suspend, usb_attached to the protocol layer; and captures attach/detach as interrupt sources in usbf_rf (maskable via register interface).
- The protocol layer (usbf_pl) uses mode_hs and usb_reset to adapt packet timing and state; attach status can be read and its edges can generate interrupts.

Timing constants
- All timing thresholds are parameterized via macros:
- T1_* (e.g., USBF_T1_C_2_5_US, _3_0_MS, _3_125_MS, _5_MS) for idle-based decisions (reset/suspend heuristics).
- T2_* (e.g., USBF_T2_C_2_5_US, _100_US, _1_0_MS, _1_2_MS, _0_5_MS tick basis, _100_MS) for reset/negotiation windows, wakeup and attach delays.

Compliance summary
- After attach (VBUS presence), the device waits ~100 ms before declaring attached.
- On reset, the device performs HS chirp negotiation per USB 2.0: drives K, counts up to 6 K/J chirps from host; selects HS on success, otherwise falls back to FS.
- Electrical controls (XcvSelect, TermSel, OpMode) are driven appropriately for FS vs HS during attach, negotiation, and normal operation.
- Attach/detach edges are latched and exposed to system software via interrupt status bits.

-- Function Address Assignment --

Function Address Assignment defines how the USB device address (7-bit) is stored and used for token matching. The core exposes a 32-bit register in the RF (register file) main page at word offset 0x01 (internal RF address adr==7'h01). Reset value is 0x00000000. Bits [6:0] hold the function address

(0–127); bits [31:7] are reserved (read as 0, write as 0).

Write behavior: On a WISHBONE write to RF offset 0x01 with WE=1, the core latches din[6:0] into funct_adr. This write occurs on the system/bus clock (wclk). There is no deferred activation: the new address takes effect immediately in the packet processing pipeline.

Use in packet matching: The packet layer compares the received token address (token_fadr) against funct_adr (fa) to produce fsel = (token_fadr == fa). Endpoint activity only proceeds when match_o = fsel & endpoint match & token_valid & !crc5_err & not a prohibited PID. Thus, after assignment, the core responds only to tokens addressed to funct_adr.

Timing/clock domains: funct_adr is written in the WISHBONE clock domain (wclk) and consumed in the PHY/packet domain (clk/phy_clk) without an explicit synchronizer. System software must update funct_adr at a quiescent time (e.g., after completing the Status stage of SET_ADDRESS) to avoid transient mismatches during in-flight tokens.

Reset/USB reset: The register clears to 0 only on core reset (rst). It is not automatically cleared on a USB bus reset event (usb_reset). Firmware should set funct_adr to 0 upon detecting a USB reset (int source provided by rf.int_srcb[8]) to comply with the USB specification.

Software model:
- Read @ RF[0x01]: returns {25'h0, funct_adr}.
- Write @ RF[0x01]: din[6:0] becomes funct_adr; din[31:7] ignored.
- Program flow for SET_ADDRESS: handle the control transfer; after the Status stage completes, write the assigned address to RF[0x01].

-- Token Detection and Endpoint Matching --

Overview: Incoming UTMI bytes are decoded into USB PIDs and token fields, validated (PID complement and CRC5), then qualified against the device Function Address and the configured Endpoint numbers. Only qualified tokens are allowed to drive the protocol engine; others are ignored and/or flagged as errors.

1) PID detection and validation (usbf_pd)
- Latching: On rx_active & rx_valid, the first byte (PID) is latched into pid.
- PID integrity: pid_cks_err asserts if pid[3:0] != ~pid[7:4] (exposed at top as pid_cs_err).
- PID classification: Individual one-hot flags for OUT/IN/SOF/SETUP, DATA0/1/2/MDATA, ACK/NACK/STALL/NYET, PRE/ERR/SPLIT/PING.

2) Token field extraction (usbf_pd)
- Sequencing: Token bytes token0 then token1 are captured in TOKEN state.
- token_valid: Asserts after the second token byte or when an ACK is seen (got_pid_ack). Used to time address/endpoint checks.
- Field mapping: token_fadr[6:0] = token0[6:0]; token_endp[3:0] = {token1[2:0], token0[7]}; frame_no = {token1[2:0], token0} for SOF.
- CRC5 check: Computed over {token_fadr, token_endp}, inverted, then compared with token1[7:3]. crc5_err asserts on mismatch.

3) Token filtering before match (usbf_pl)
- pid_bad: Filters out handshake PIDs (ACK/NACK/STALL/NYET), PRE/ERR/SPLIT always, and PING when not in HS mode.
- Only non-handshake, valid tokens proceed: gating conditions require !pid_bad & token_valid & !crc5_err.

4) Function Address match (usbf_pl + usbf_rf)
- fa (funct_adr) is a writeable 7-bit register in usbf_rf (WB space adr==7'h1) holding the device address.
- fsel = (token_fadr == fa). Only tokens addressed to this function are processed.

5) Endpoint match (usbf_rf + usbf_ep_rf)
- ep_sel (from token_endp) is presented to all 16 endpoint register files.
- Each endpoint compares ep_sel to its configured endpoint number csr[21:18]. When equal, that endpoint's ep_match asserts.
- rf.match is the OR of all epX_match, registered (match_r1). Disabled endpoints (dummy instances) never match.
- The endpoint's CSR/buffer context (csr, buf0, buf1, and DMA hints) corresponding to the matching ep are multiplexed to the protocol engine.

6) Final match qualification into Protocol Engine (usbf_pl)
- match_o = !pid_bad & fsel & match & token_valid & !crc5_err.
- match_o is fed to usbf_pe as match. If a directed token (OUT/IN/SETUP) fails endpoint match, usbf_pe sets nse_err (No Such Endpoint): nse_err <= token_valid & (pid_OUT | pid_IN | pid_SETUP) & !match.

7) Timing and domains
- Detection (usbf_pd), filtering, and match qualification (usbf_pl) run on the UTMI clock (phy_clk_pad_i).
- Endpoint comparison logic inside usbf_rf/usbf_ep_rf also clocks on phy_clk for match paths; WB programming uses wclk.

Result: A token is considered valid-and-directed when PID is consistent, CRC5 passes, it is addressed to this function (fa), and its endpoint number matches a configured endpoint. Only then does match_o assert, enabling the protocol engine for the transaction; errors (pid_cs_err, crc5_err, nse_err) annotate invalid/misdirected tokens.

-- IN Transaction Flow --

IN Transaction Flow (device-to-host data stage)

Entry conditions
- Trigger: Receipt of a valid IN token for the addressed endpoint (match asserted) and not pid_SOF.
- Endpoint gating in IDLE state (usbf_pe):
- If endpoint stalled (csr[23:22] == 2'b10) -> send STALL token and return to IDLE.
- If no buffer available or DMA cannot serve (e.g., both buffers NA, or DMA in-buffer insufficient) -> send NAK token and return to IDLE.
- Otherwise, if endpoint direction/config allows IN (IN_ep or CTRL_ep with IN phase), proceed with an IN transfer.

Initiation (state IN)
- tx_dma_en is asserted to start fetching payload from buffer memory via IDMA (usbf_idma).
- If MaxPacketSize (csr[10:0]) == 0, send_zero_length is asserted (zero-length DATA with only CRC).
- The DATA PID to use (DATA0/1/2/MDATA) is selected by data_pid_sel, computed from endpoint type (iso/bulk/interrupt/control), HS/FS mode, transfer framing, and the current/next data toggle (uc_dpd); allow_pid reflects received DATAx for validation.
- Starts ACK wait timer: rx_ack_to_clr_d deasserted to enable rx_ack_to counting.
- Transition: When IDMA signals idma_done:
- Isochronous (csr[25:24] == 2'b01): skip ACK wait -> UPDATE.

- Others (bulk/int/control): go to IN2 to await host ACK.

Data transmission path
- usbf_idma reads from buffer at adr/size determined by usbf_pe, honoring buf_size and DMA enable.
- usbf_pa emits: DATA PID -> payload bytes -> CRC16. It handles zero-length case by emitting only CRC after PID.
- send_data drives pa; rd_next handshakes the byte stream; tx_valid/tx_first/tx_valid_last are generated to UTMI MAC.

ACK handling (state IN2, non-iso endpoints)
- Waits for token_valid && pid_ACK from host.
- rx_ack_to timeout uses mode-dependent constants (HS: USBF_RX_ACK_TO_VAL_HS, FS: USBF_RX_ACK_TO_VAL_FS). If timeout (rx_ack_to asserted) occurs before ACK, aborts to IDLE (no buffer pointer/toggle update).
- On ACK reception, proceeds to UPDATE.

Buffer/accounting update (state UPDATE then UPDATE2)
- int_set_en asserted to enable endpoint interrupt cause generation.
- If buffer_done && dma_en:
- buf0_rl_d asserted to release the consumed buffer (toggle buffer ownership in ring mode).
- Else:
- buf_set_d asserted to write back updated buffer address/size and next data toggle/buffer select into idin; buf0_set or buf1_set strobes choose which buffer gets updated.
- idin contents:
- [31:17] = {buffer_done, new_size} (or {0, sizu_c} on certain conditions like out_to_small, not used for IN data writes).
- [SSRAM_HADR+2:4] = new_adr (next buffer address in words).
- [3:0] = {next_dpid, next_bsel} (encodes next DATA PID and buffer select).
- UPDATE2:
- uc_bsel_set and uc_dpd_set asserted to commit next buffer select and data PID toggle into csr.
- Returns to IDLE.

Error/abort behavior
- If a new token match occurs while busy (match & state != IDLE) or other error conditions, abort is asserted, forcing return to IDLE.
- For IN path specifically:
- Isochronous: no ACK wait; always updates buffer/toggle in UPDATE.
- Bulk/Interrupt/Control: missing ACK (timeout) ends transfer without UPDATE (no toggle/buffer update), leaving state for reattempt on a subsequent IN token.

Key interactions and constraints
- Buffer selection: buf_sel = 0 when DMA enabled; otherwise based on uc_bsel and buffer availability. For IN+DMA, buf0 is preferred unless endpoint configuration dictates otherwise via uc_bsel.
- Size selection: size_next = min(buf_size, MaxPacketSize). Zero-length handling if MaxPacketSize==0.
- Handshakes prior to IN data stage (in IDLE): STALL or NAK are sent via send_token/token_pid_sel when endpoint is stalled or lacks resources.
- Timing: ACK wait cleared on tx_valid activity and restarted per transaction; counts in PHY clock domain.

Signals of interest
- Entry/decision: match, IN_ep/CTRL_ep, ep_stall, no_buf0_dma, buf0_na/buf1_na, mode_hs.

- Start: tx_dma_en, send_zero_length, data_pid_sel.
- IDMA path: adr, size, dma_en, idma_done, rd_next, tx_data_st, send_data.
- ACK wait: rx_ack_to_clr_d, rx_ack_to, token_valid, pid_ACK.
- Update: int_set_en, buf0_rl, buf0_set, buf1_set, idin, uc_bsel_set, uc_dpd_set.
- Interrupts: int_buf0_set/int_buf1_set asserted in UPDATE when buffer_done and buffer allocated.

Outcome
- On successful IN transaction (data sent and ACKed, or any iso IN): data toggle and buffer bookkeeping are updated, and buffer may be released or advanced; endpoint interrupt sources are generated accordingly. On NAK/STALL gating or ACK timeout, no state updates are committed.

-- OUT and SETUP Transaction Flow --

Scope: OUT and SETUP packet handling inside the USB function core protocol/link layer (modules usbf_pl/usbf_pe/usbf_idma/usbf_pd). This describes how OUT and SETUP tokens are accepted, data are moved to memory, handshakes are generated, and endpoint/buffer state is updated.

Token gating and endpoint match:
- A token is considered for processing only if: token_valid=1 with no CRC5 error, PID is not a non-data/bad PID (not ACK/NACK/STALL/NYET/PRE/ERR/SPLIT; PING is separate), device address matches (fsel), and the endpoint number matches a configured EP (match from register file). If a valid OUT/IN/SETUP token does not match, nse_err is raised.

Entry to OUT/SETUP flow:
- Control endpoint (CTRL_ep): SETUP and OUT tokens go through the OUT data path; control endpoints never use DMA (dma_en=0 for CTRL).
- Non-control OUT endpoint: OUT token goes through the OUT data path. When a matching, enabled endpoint is selected and not stalled, the protocol engine asserts rx_dma_en and transitions to OUT state.

Data reception and integrity:
- usbf_pd streams received bytes and computes CRC16. It asserts rx_data_valid and eventually rx_data_done when the payload ends. usbf_idma writes payload into the buffer/memory designated by the current buffer descriptor, accumulating the received byte count sizu_c.
- If a payload CRC16 error occurs (crc16_err=1) or a data timeout occurs while expecting data (tx_data_to=1) or an abort condition is detected, the transfer is dropped and the state machine returns to IDLE without sending a handshake (ISO and non-ISO alike), per USB rules for CRC errors.

Handshake policy after data (non-isochronous):
- After rx_data_done, the engine evaluates conditions in OUT2A/OUT2B to decide the handshake:
- Size policy checks: max packet size is csr[10:0]. If sml_ok=0 and received size < max, or lrg_ok=0 and received size > max, a NAK is returned and the transfer is discarded (to_small/to_large).
- PID sequence error (unexpected DATA PID relative to toggle state): a sequence error sets int_seqerr_set; for non-ISO endpoints the core sends ACK and returns to IDLE (ISO discussed below).
- High-speed flow control: if in HS mode and no_bufs=1 (no buffer available for next transaction), return NYET; otherwise return ACK.
- PING is handled separately (not part of OUT and SETUP data phase), but note HS PING gets ACK/NACK based on buffer readiness; OUT handshake is generated only after data as described above.

Isochronous OUT:
- For isochronous endpoints (txfr_iso=1), after rx_data_done there is no handshake. If a PID sequence

error is detected, an interrupt (int_seqerr_set) is generated, but no token is sent. The flow goes to an update stage to commit descriptors.

Buffer selection and DMA/data movement:
- Buffer descriptors buf0/buf1 carry address, size, and meta (including DP/BS). For DMA-enabled non-control endpoints, buf_sel is forced to buf0 during the transaction; for non-DMA/control, buf_sel follows endpoint/control rules.
- usbf_idma writes received bytes into memory at adr, increments MADR on each accepted word, and tracks sizu_c (received bytes). It can also transmit (IN), but for OUT/SETUP only the receive path is used.

Update phase and descriptor/DP/BS maintenance:
- After completing the data and handshake decision, the engine enters UPDATE/UPDATE2:
- If buffer_done=1 and DMA is enabled: assert buf0_rl to release/rotate the buffer (ring advance).
- Otherwise: compute new_size and new_adr based on sizu_c (or max packet size for certain DMA cases) and program idin with updated fields. Assert buf0_set or buf1_set (depending on which buffer was used) to write the descriptor with the new size and address tail. Short OUT in DMA mode sets out_to_small and writes the original size/location.
- Update and publish next toggle and buffer select (uc_dpd_set and uc_bsel_set), and raise the appropriate buffer interrupt (int_buf0_set/int_buf1_set) when buffer is consumed.

Flow-control prechecks and stalls:
- If the endpoint is stalled, the core responds with STALL (handshake) and returns to IDLE.
- If both buffers are not available, or (in HS) DMA indicates insufficient space for at least one max packet (no_buf0_dma/dma_out_buf_avail=0), the core flow-controls the host. For OUT this results in returning NAK/NYET as described above after data; for HS PING the immediate ACK/NAK is returned.

Abort conditions:
- The active transaction is aborted if: buffer_overflow is detected, a matching token arrives while the endpoint state machine is not in IDLE (new transaction preempts), or a to_large condition is detected mid-flow. Abort drops the packet and returns to IDLE without handshake.

SETUP specifics (control):
- SETUP tokens are treated as OUT to the control endpoint (CTRL_ep). The state machine routes them into the OUT path (rx_dma_en asserted). Data PID handling for control is forced so that SETUP stage uses DATA0 and internal next/this PID tracking updates accordingly. Control endpoints do not use DMA; descriptor and DP/BS updates are applied via buf0_set/buf1_set and uc_*_set in UPDATE/UPDATE2.

Interrupts and status:
- The engine asserts per-endpoint status/interrupt bits: int_seqerr_set on PID sequence error, int_to_set on timeouts (IN/OUT), int_crc16_set on CRC error (at data_done), and buffer interrupts on buffer commit. out_to_small records short OUTs in DMA mode. nse_err is asserted when an OUT/IN/SETUP token is received for a non-matching endpoint or address.

-- Ping-Pong Buffer Management and Data PID Handling --

Overview
- Each endpoint has two 32-bit buffer descriptors (B0/B1) and endpoint control/state in CSR. This block manages ping■pong (double) buffering and Data PID sequencing for IN/OUT/CTRL transfers in both FS and HS modes.

Buffer Descriptor Format and Status

- bufX[31] = NA (Not Available) flag. Additionally, a descriptor is considered not allocated if its base address field is all ones.
- bufX[SSRAM_HADR+2:0] = Base address (word+byte address; low nibble holds byte address).
- bufX[30:17] = Remaining size (bytes) of this buffer region.
- uc_bsel (CSR[31:30]) and uc_dpd (CSR[29:28]) are maintained in the endpoint CSR, not inside the buffer descriptor. They are updated from the IDIN bus when uc_bsel_set/uc_dpd_set strobe.

Active-Buffer Selection (buf_sel)
- DMA enabled non-control endpoints (CSR.DMA_EN=1 and DIR!=CTRL): Always use B0.
- Control endpoints (DIR=CTRL): Use B0 for OUT/SETUP stages, B1 for IN stages (buf_sel = in_token).
- All other cases (non-DMA, non-CTRL): Choose B1 when (uc_bsel[0] == 1) or B0 is NA, and B1 is available; otherwise choose B0. This realizes ping■pong by toggling uc_bsel when a buffer completes.

Transfer Accounting and Completion
- MaxPacket = CSR[10:0]. For each transaction the engine computes:
- size_next = min(buf_size, MaxPacket)
- new_size = buf_size - bytes_transferred
- new_adr = adr + bytes_transferred
- Bytes transferred:
- IN: size_next
- OUT (DMA path): MaxPacket
- OUT (non-DMA path): sizu_c (measured received bytes)
- Buffer completion (buffer_done) criteria:
- IN: new_size == 0 (buffer empty)
- OUT: new_size < MaxPacket (buffer considered full enough for a packet)

Descriptor Write■Back and Ping■Pong Update
- On UPDATE state:
- If DMA enabled and buffer_done: assert buf0_rl to reload B0 from its original descriptor (buf0_orig). This creates a cyclic, constant descriptor for DMA ring operation.
- Else: assert bufX_set (X = selected buffer) to write back descriptor with updated address and size.
- On UPDATE2 state: assert uc_bsel_set and uc_dpd_set to commit next ping■pong pointer and next Data PID to CSR.
- Ping■pong pointer update (next_bsel):
- DMA endpoints: forced to 0 (no ping■pong).
- Non-DMA: uc_bsel increments when buffer_done; otherwise it holds.

IDIN Bus Composition (write■back vector)
- IDIN[31:17] <= {buffer_done, new_size} by default. For OUT with DMA when the packet is smaller than MaxPacket (out_to_small), it carries {4'h0, sizu_c} instead.
- IDIN[SSRAM_HADR+2:4] <= new_adr[MSBs] (or original adr when out_to_small).
- IDIN[3:0] <=
- During bufX_set: new_adr[3:0] (address low bits), writing the descriptor.
- During uc_bsel_set/uc_dpd_set: {next_dpid[1:0], next_bsel[1:0]}, updating CSR.

Flow Control, NAK/NYET Policy, and Interrupt Tags
- If endpoint is STALLed: return STALL.
- If both buffers are NA, or DMA resources are not ready (IN and !dma_in_buf_sz1, or OUT and !dma_out_buf_avail), or CTRL-phase buffer is missing: return NACK from IDLE.
- For HS OUT completion (OUT2B):
- If no buffers available: return NYET; else ACK.

- If RX Data PID sequence error: ACK but raise int_seqerr_set.
- int_buf0_set/int_buf1_set pulse when a buffer transitions through completion (buffer_done) and is eligible; these are qualified by allocation status to avoid reporting on unallocated buffers.
- out_to_small pulses for DMA OUT when received size != MaxPacket; buf1 is updated with size and an interrupt can be generated.

DMA Aids (computed in endpoint RF)
- dma_in_buf_sz1: IN has at least one MaxPacket worth of data ready in B0 (and MaxPacket != 0).
- dma_out_buf_avail: OUT has at least one MaxPacket worth of free space remaining in current OUT buffer.

Data PID Generation and Checking
- The engine keeps a 2-bit current Data PID state (uc_dpd) per endpoint in CSR and computes:
- this_dpid: expected PID for the current transfer (drives data_pid_sel for transmit; used to check received PIDs).
- next_dpid: PID for the next transaction, written back to uc_dpd during UPDATE2.
- Selection logic depends on:
- Endpoint direction (CSR[27:26]), type/transfer attributes (CSR[25:24], CSR[12:11]), high-speed vs full-speed (mode_hs), control-stage (in_token, out_token, setup_token), and observed HS multi■data patterns (MDATA/DATA2 rules per USB2 bulk HS).
- pid_seq_err is asserted when a received DATA PID doesn't match this_dpid. For iso OUT it raises int_seqerr_set; for bulk OUT the state machine ACKs and completes while flagging the error.

Special Cases
- Control endpoints: data PID handling resets appropriately around SETUP/IN/OUT phases; buffer selection is fixed per phase (B0 for OUT/SETUP, B1 for IN).
- Zero-length IN: If MaxPacket == 0, send_zero_length is asserted to transmit ZLP.

Summary of Handshake Outputs
- token_pid_sel: ACK/NACK/STALL/NYET chosen per endpoint state, buffer availability, error conditions, and HS vs FS rules.
- data_pid_sel: DATA0/DATA1/DATA2/MDATA chosen by this_dpid for transmit.

Key Signals
- buf0_set, buf1_set: Write back updated buffer descriptor.
- buf0_rl: Reload B0 from its original CPU■programmed descriptor (DMA ring).
- uc_bsel_set, uc_dpd_set: Update ping■pong pointer and Data PID state in CSR.
- int_buf0_set, int_buf1_set, int_seqerr_set, out_to_small: Interrupt sources tied to buffer completion, PID sequence error, and short OUT packet in DMA mode.

-- DMA Read/Write Operation and Buffer Updates --

Scope
- Describes how payload data is moved between the UTMI interface and the shared SSRAM (internal DMA), and how endpoint buffer descriptors are updated afterward. Includes flow-control conditions that gate DMA and the exact buffer-maintenance actions.

Terminology and roles
- RX DMA (OUT/SETUP): host sends data to the device. Data is written from UTMI to SSRAM.
- TX DMA (IN): device sends data to the host. Data is read from SSRAM to UTMI.
- Packet Engine (PE, usbf_pe): decides when to perform RX/TX DMA, computes new buffer pointers/sizes, and issues buffer update commands.

- Internal DMA (IDMA, usbf_idma): performs aligned memory reads/writes on the shared SSRAM using mreq/mack/mwe/madr/mdout/mdin.
- Register File (RF, usbf_ep_rf): holds endpoint CSR and buffer descriptors (buf0, buf1, original buf0). Applies PE updates and generates external dma_req (system-level DMA) and availability flags (dma_in_buf_sz1, dma_out_buf_avail).
- Memory arbiter (usbf_mem_arb): arbitrates SSRAM between IDMA (m* port) and Wishbone (w* port).

When DMA is used
- DMA enable is CSR[15] and only active for non-control endpoints (csr[27:26] != 2'b00). For DMA-enabled endpoints the PE always operates on buf0 (buf_sel = 0).
- Before servicing tokens the PE checks flow-control: for IN, dma_in_buf_sz1 must be 1 (>= one max packet available); for OUT, dma_out_buf_avail must be 1 (>= one max packet space available). If not, PE NACKs the token.

IDMA operation (aligned SSRAM access)
- Address inputs: base adr[SSRAM_HADR+2:0] and transfer size size[13:0] come from the PE. buf_size[13:0] provides the configured buffer span.
- Addressing is word-based on the SSRAM (madr = adr_cw; 32-bit), with internal byte-steering using adr_cb[1:0].
- Wrap control (DMA only): last_buf_adr = adr + buf_size. When an increment would reach last_buf_adr, IDMA wraps adr_cw to 0 (start of the configured region) for ring-buffer semantics.

RX DMA (OUT/SETUP)
- State sequence: WAIT_MRD -> MEM_WR -> MEM_WR1 -> MEM_WR2.
- WAIT_MRD: pre-read the current word (mreq with dtmp_sel=1) to merge a possibly unaligned first write.
- MEM_WR: assemble 32-bit words from rx_data_st by byte position (adr_cb[1:0]) and write (mwe=1). word_done when the 4th byte arrives or at end-of-packet (wr_last flush).
- MEM_WR1/2: finalize partial last word and complete the final write.
- sizu_c counts received bytes (increments on rx_data_valid). idma_done asserts when rx_data_done or after the last flush.

TX DMA (IN)
- State sequence: MEM_RD1 -> MEM_RD2 -> MEM_RD3 (loops).
- MEM_RD1: issue first read (mreq) and mark rd_first.
- MEM_RD2/3: pipeline alternating word reads; fill rd_buf0/rd_buf1 on mack, and feed bytes to the transmitter via tx_data_st according to adrb_next[2:0].
- sizd_c is loaded from size and decremented on rd_first/mack and each rd_next until zero; idma_done asserts when sizd_c reaches zero (or on abort).
- send_data is asserted from the first fetched word until the last byte; send_zero_length is honored to emit a zero-length data phase when requested.

Arbitration to SSRAM (usbf_mem_arb)
- Grants to Wishbone when (wreq | wack) & !mreq, otherwise to IDMA. sram_adr/sram_dout/sram_we mux between w* and m* sides. sram_re is always 1. mack mirrors mreq (single-cycle ready by construction in this model).

Buffer accounting in PE (post-transfer)
- Operates on descriptor fields: bufX_adr = bufX[SSRAM_HADR+2:0], bufX_size = bufX[30:17], uc_bsel = csr[31:30], uc_dpd = csr[29:28], max_pl_sz = csr[10:0].
- Size to subtract (new_sizeb):
- OUT & DMA: use max_pl_sz (fixed-slot semantics).

- IN: use size_next = min(buf_size, max_pl_sz).
- OUT & non-DMA: use actual received sizu_c.
- New metrics: new_size = buf_size - new_sizeb; new_adr = adr + new_sizeb (both saturate to configured widths). Buffer completion test:
- IN: buffer_done when new_size == 0.
- OUT: buffer_done when new_size < max_pl_sz (not enough space for another full packet).

Handling short OUT packets under DMA
- If OUT & DMA and received sizu_c != max_pl_sz, out_to_small is set. In this case the PE writes back:
- idin[31:17] = {4'h0, sizu_c} (actual short size),
- idin[SSRAM_HADR+2:4] = original buf0_adr (do not advance),
- idin[3:0] unchanged for address; UC fields handled in UPDATE2.
- RF captures this in buf1 (always @(buf1_set || out_to_small) buf1 <= idin) to preserve the short-packet metadata.

Buffer update commands from PE
- UPDATE state after successful data phase:
- If buffer_done && DMA: assert buf0_rl (release). RF restores buf0 <= buf0_orig (the original descriptor staged by software), effectively closing the slot and re-arming the ring.
- Else: assert bufX_set to write idin into the active descriptor (buf0 for DMA; buf0/1 per selection for non-DMA). idin packs: {new_size or short-size, new_adr[SSRAM_HADR+2:4], low[3:0] = new_adr[3:0] when buf_set_d else {next_dpid, next_bsel}}.
- UPDATE2 state: assert uc_bsel_set and uc_dpd_set to write back updated UC buffer select and next data PID sequencing bits.

Flow-control signals into PE and NAK/NYET behavior
- no_buf0_dma is true for DMA endpoints when:
- IN: !dma_in_buf_sz1 (fewer than one max packet available to send),
- OUT: !dma_out_buf_avail (fewer than one max packet worth of free space).
- In PE.IDLE, if ep is stalled: send STALL. If any of: both buffers NA, no_buf0_dma, or control-direction buffer NA: send NACK. HS PING is ACKed. Otherwise PE proceeds with RX/TX DMA and subsequent updates.
- For HS OUT with no buffers, PE may send NYET instead of ACK per mode_hs and no_bufs.

Endpoint RF side accounting and external DMA handshake
- RF keeps counters per endpoint:
- OUT: dma_out_cnt decrements on dma_ack_i (external DMA serviced) and increments by max_pl_sz on buf0_set/rl. dma_out_left = buf0_orig.size - dma_out_cnt; dma_out_buf_avail = (dma_out_left >= max_pl_sz).
- IN: dma_in_cnt increments on dma_ack_i and decrements by max_pl_sz on buf0_set/rl. dma_in_buf_sz1 = (dma_in_cnt >= max_pl_sz).
- RF raises dma_req when either IN needs data or OUT has data to drain, subject to hold conditions. External system asserts dma_ack to accept or provide data.

Completion signaling
- idma_done informs the PE that the data mover finished for the current token (either RX end-of-packet processing or TX size exhausted). The PE then performs UPDATE/UPDATE2 to finalize buffer metadata and UC fields.

Error/exception interactions (brief)
- Abort of IDMA/PE paths occurs on protocol/CRC/timeouts or size violations. In these cases PE returns to IDLE without applying normal ACK/NYET or without advancing descriptors (except where

out_to_small dictates preserving a short-packet record).

-- Handshake Generation and Timeouts (ACK/NACK/STALL/NYET) --

- Scope: This block specifies when the device returns STALL, NACK, ACK, or NYET handshakes, and how the IN-ACK and OUT-data timeouts are handled in the packet engine (usbf_pe) and packet arbiter (usbf_pa).

- Handshake emission mechanism:
- Handshakes are emitted by asserting send_token together with token_pid_sel. The usbf_pa module encodes token_pid_sel into the proper PID bytes (including complement) and transmits exactly one handshake packet (state TOKEN is a 1-cycle staging state).
- Mapping: 0=ACK, 1=NACK, 2=STALL, 3=NYET.

- When a matching token is received (match_r) and the endpoint is enabled (ep_disabled=0) and the token is not SOF:
- STALL: If the endpoint is stalled (csr[23:22]==2'b10), immediately send STALL and finish (IDLE→TOKEN→IDLE).
- NACK (resource unavailable at token time): If buffers are not available or DMA cannot proceed, immediately send NACK and finish. Conditions:
- (buf0_na && buf1_na) OR no_buf0_dma OR (CTRL endpoint and required direction buffer not present: CTRL+IN needs buf1, CTRL+OUT needs buf0).
- ACK (PING response in HS): If a HS PING token is received (pid_PING && mode_hs), immediately send ACK.
- IN data phase start: If IN endpoint (or CTRL IN) is selected and max_pl_sz==0, assert send_zero_length; enable TX-DMA and enter IN state; otherwise TX-DMA without handshake here (host later ACKs).
- OUT data phase start: If OUT endpoint (or CTRL OUT/SETUP) is selected, enable RX-DMA and enter OUT state.

- OUT transaction handshakes (non-isochronous):
- After successful data reception (rx_data_done), evaluate in OUT2B:
- NACK on size policy violations: if to_small (shorter than allowed when sml_ok=0) or to_large (larger than allowed when lrg_ok=0), return NACK.
- ACK on success when buffers are available.
- NYET in HS only when no further buffers are available (mode_hs && no_bufs). This indicates the function accepted this OUT but cannot accept another in the same microframe.
- Sequence error (PID toggle mismatch): int_seqerr_set is raised; device still returns ACK (accepts the packet but notes the sequence error).
- Isochronous OUT (txfr_iso): No handshake is sent (USB spec-compliant behavior); state proceeds to update bookkeeping.
- Abort paths (any time during OUT/OUT2A/OUT2B): If abort, CRC16 error, or TX data timeout triggers, return to IDLE with no handshake.

- IN transaction handshakes:
- The device does not send a handshake after IN data packets (the host does). The core waits for a host ACK in state IN2. On ACK receipt (token_valid && pid_ACK) it proceeds to UPDATE; on timeout it abandons and returns to IDLE.
- Isochronous IN: No ACK wait; transitions directly from IN to UPDATE.

- Control endpoints:
- Immediate NACK if the required buffer for the control direction is not present: CTRL+IN needs buf1; CTRL+OUT/SETUP needs buf0.

- NYET generation (HS only):
- Sent after a valid OUT when mode_hs==1 and no_bufs==1, where no_bufs reflects buffer not-available conditions across the currently selected and alternate buffers (including DMA availability signals dma_out_buf_avail/dma_in_buf_sz1 and max-payload sizing checks).
- In FS, the same condition results in ACK (NYET is HS-only per spec).

- Timeouts:
- IN ACK wait timeout (rx_ack_to):
- Applies to non-iso IN transfers while waiting for the host ACK in state IN/IN2.
- Counter runs when the core is in IN/IN2; reset on tx_valid activity and outside IN/IN2.
- Thresholds: USBF_RX_ACK_TO_VAL_FS=36 cycles (FS), USBF_RX_ACK_TO_VAL_HS=22 cycles (HS), measured on the core/PHY clock domain.
- On timeout (no ACK in time): transition to IDLE (no endpoint update is performed).
- OUT data timeout (tx_data_to):
- Counter clears whenever rx_active is asserted; otherwise it increments.
- Thresholds: USBF_TX_DATA_TO_VAL_FS=36 cycles (FS), USBF_TX_DATA_TO_VAL_HS=22 cycles (HS).
- In OUT state, if tx_data_to asserts (bus inactive too long during expected reception), transition to IDLE (no handshake is sent).

- Additional notes and guards:
- Handshakes are never generated for SOF.
- If the endpoint is disabled (csr[23:22]==2'b01), no handshakes/data operations are initiated.
- Abort conditions (buffer overflow, overlapping match while active, size-too-large at start) force exit to IDLE; these suppress handshakes for the current transaction.
- usbf_pa ensures proper single-PID emission with last-cycle extension; CRC for data packets is handled in the transmitter but does not affect the handshake PID generation beyond the state machine decisions above.

-- Error Handling (PID, CRC5/CRC16, NSE, Sequence Errors) --

Scope: This core detects, reports, and reacts to protocol errors on the UTMI/USB packet stream. Error conditions are surfaced both as device-level interrupt sources (rf.int_srcb) and, when applicable, as endpoint-level interrupt bits (ep.int_stat) that are individually masked and cleared by software.

Sources and detection:
- PID checksum (pid_cs_err): In usbf_pd, a PID is flagged bad when pid[3:0] != ~pid[7:4]. Exposed to the RF block as pid_cs_err and latched into rf.int_srcb[1]. The core does not gate token processing solely on this flag; it is for software observability.
- CRC5 on tokens (crc5_err): usbf_pd recomputes CRC5 over {faddr,endp} and compares with the token's CRC (active low). crc5_err asserts at token_valid and mismatch. Device-level: rf.int_srcb[0]. Functional effect: usbf_pl suppresses endpoint match on CRC5 error (match_o requires !crc5_err), so the protocol engine treats such tokens as non-matches (see NSE below). No ACK/handshake is sent in response.
- CRC16 on data (crc16_err): usbf_pd accumulates CRC16 over payload (init 0xffff) and flags error at end of DATA (expected remainder 0x800d). Endpoint-level: usbf_pe sets int_crc16_set on rx_data_done & crc16_err for the matched endpoint (ep.int_stat[1]). Functional effect: in OUT state, crc16_err causes immediate abort to IDLE (no ACK sent), allowing host retry. IN path (device transmit) has no receive-CRC16 check by the device.
- Non-match/NSE (nse_err): In usbf_pe, nse_err is asserted when token_valid & (pid_OUT | pid_IN | pid_SETUP) & !match. Because match_o = !pid_bad & fsel & match & token_valid & !crc5_err, both

true address/endpoint mismatches and CRC5-failed tokens appear as "no-match" to the PE and raise nse_err. Device-level: rf.int_srcb[2]. No handshake is returned.
- Data PID sequence error (int_seqerr_set): In usbf_pe, pid_seq_err asserts if the received DATA PID does not match the expected sequence (this_dpid vs pid_DATA0/1/2/MDATA). Behavior by transfer type:
- Isochronous OUT: int_seqerr_set is raised (ep.int_stat[5]) and the engine proceeds to UPDATEW/UPDATE.
- Non-iso OUT: the core sends ACK and returns to IDLE; int_seqerr_set is not raised.
- IN (device transmit): the host validates sequence; device does not flag a receive-side sequence error.
- Invalid/unsupported tokens (pid_bad): Handshake and other non-token PIDs (ACK/NACK/STALL/NYET/PRE/ERR/SPLIT) are filtered from match; PING is allowed only in HS (in FS it is pid_bad). These are not explicitly reported as errors unless they lead to NSE via match suppression.

Interrupt reporting, masking, and clear:
- Device-level error sources (rf.int_srcb[8:0]):
- [8] usb_reset, [7] rx_err (UTMI RxError), [6] detach, [5] attach, [4] suspend_end, [3] suspend_start,
- [2] nse_err, [1] pid_cs_err, [0] crc5_err.
- Read of RF main interrupt source register (adr==7'h3 with re) clears all bits (int_src_re handshake).
- inta_rf/intb_rf are the masked OR of int_srcb via inta_msk/intb_msk.
- Endpoint-level error/condition bits (ep.int_stat per endpoint, cleared on read of that endpoint’s INT register adr[1:0]==2'h1 with re):
- [6] out_to_small, [5] int_seqerr_set (data PID sequence error, iso OUT only), [4] int_buf1_set, [3] int_buf0_set,
- [2] int_upid_set (unexpected/changed PID context), [1] int_crc16_set (data CRC error), [0] int_to_set (protocol timeout from usbf_pe: ACK timeout on IN, DATA timeout on OUT).
- Endpoint inta/intb lines are the masked ORs of int_stat (iena/ienb).

Protocol reactions summary:
- Bad PID checksum: no automatic transaction suppression; software notified via rf.int_srcb[1].
- Bad token CRC5: transaction is ignored (no endpoint match, no handshake), rf.int_srcb[0]=1, and nse_err also asserts because !match.
- Bad data CRC16 on OUT: drop/abort with no ACK; ep.int_stat[1]=1 so software can recover.
- No-match/NSE (wrong address/endpoint or CRC5 failure): no handshake; rf.int_srcb[2]=1.
- Data PID sequence error: iso OUT raises ep.int_stat[5]; non-iso OUT sends ACK and does not raise the bit.

Software guidance:
- Use rf.int_srcb to catch device-level stream integrity issues quickly (pid_cs_err, crc5_err, nse_err) and clear them by reading 7'h3.
- Use per-endpoint int_stat for data-plane errors (CRC16, sequence) and timeouts; clear by reading the endpoint’s INT register.
- On CRC16/NSE/CRC5 errors, expect the core to avoid acknowledging faulty OUTs or malformed tokens so the host retries; firmware should inspect and reset/prepare buffers as needed.

-- Interrupt Generation, Masking, and Clearing --

Overview
- The core exposes two interrupt outputs: inta_o and intb_o.
- Each is the OR of: (a) per-endpoint masked interrupt events and (b) global (RF-level) masked interrupt events.

Per-endpoint interrupts (generated in usbf_ep_rf)

- Status bits (latched until cleared):
0: Timeout (int_to_set)
1: CRC16 error on data payload (int_crc16_set)
2: Unexpected/invalid PID sequence (int_upid_set)
3: Buffer 0 update/commit event (int_buf0_set)
4: Buffer 1 update/commit event (int_buf1_set)
5: Data PID sequence error (int_seqerr_set)
6: OUT packet size smaller than programmed (out_to_small)
- Masking to INTA/INTB (two independent mask groups per endpoint):
- iena[5:0] masks endpoint sources into INTA (inta)
- ienb[5:0] masks endpoint sources into INTB (intb)
- Mapping (both iena and ienb):
• bit0 → status[0] Timeout
• bit1 → status[1] CRC16 error
• bit2 → status[2] Unexpected PID
• bit3 → status[3] Buffer0 and status[4] Buffer1 (shared mask bit)
• bit4 → status[5] PID sequence error
• bit5 → status[6] OUT too small
- Assertion
- Endpoint inta/intb are asserted when (int_stat & mask) has any bit set.
- Clearing
- Reading the endpoint INT register (EPx.INT) clears all seven status bits (int_stat) for that endpoint in a single operation.
- Register access (EP window, adr[1:0]):
- 0x0: CSR (control/status)
- 0x1: INT: read returns {zeros, iena, zeros, ienb, zeros, int_stat}; write updates iena (bits [29:24]) and ienb (bits [21:16]); read side-effect clears int_stat.
- 0x2: BUF0
- 0x3: BUF1

Global (RF-level) interrupts (generated in usbf_rf)
- Sources (int_srcb[8:0], latched until cleared):
8: USB reset detected
7: RX error (UTMI RxError)
6: De-attach (usb_attached falling edge)
5: Attach (usb_attached rising edge)
4: Suspend end (exit suspend)
3: Suspend start (enter suspend)
2: NSE (Non-SEQ) error
1: Token PID checksum error (pid_cs_err)
0: SOF token CRC5 error (crc5_err)
- Masks
- inta_msk[8:0] masks RF sources into inta_o.
- intb_msk[8:0] masks RF sources into intb_o.
- inta_rf = OR reduction of (int_srcb & inta_msk);
intb_rf = OR reduction of (int_srcb & intb_msk).
- Clearing
- Reading the global INT_SRC register clears all bits in int_srcb[8:0].
- Note: int_srca[15:0] (endpoint summary) reflects epX_inta | epX_intb for each endpoint and is not latched/cleared; it updates continuously.

Top-level interrupt outputs

- inta_o = OR of all endpoint inta signals and inta_rf (masked global sources).
- intb_o = OR of all endpoint intb signals and intb_rf (masked global sources).

Address map within RF space (adr[6:2] page, adr[1:0] offset)
- Global pages (adr[6:2] = 0x00 or 0x01):
- 0x00/0x01, offset 0x2 (adr[6:0] == 0x02): INT_MASK
• Read: {zeros, intb_msk[8:0] at [24:16], zeros, inta_msk[8:0] at [8:0]}
• Write: updates intb_msk (din[24:16]) and inta_msk (din[8:0])
- 0x00/0x01, offset 0x3 (adr[6:0] == 0x03): INT_SRC
• Read: {zeros[31:29], int_srcb[8:0] at [28:20], zeros[19:16], int_srca[15:0] at [15:0]}
• Read side-effect: clears int_srcb[8:0]
- Endpoint windows (adr[6:2] = 0x04..0x13 for EP0..EP15):
- offset 0x1 (INT):
• Read: returns masks (iena/ienb) and status (int_stat[6:0])
• Write: updates iena (din[29:24]) and ienb (din[21:16])
• Read side-effect: clears int_stat[6:0]

Notes
- All masks default to 0 after reset (interrupts disabled until enabled by software).
- Endpoint status bits are edge/condition latched and remain set until software reads EPx.INT.
- Global RF sources are latched and remain set until software reads INT_SRC.
- Endpoint summary (int_srca) is a live OR of endpoint inta|intb and is not affected by reads.
- inta/intb are registered on the WISHBONE clock (wclk); endpoint status generation occurs on the PHY clock (clk). Software should follow standard read-clear procedures when crossing clock domains.

-- Suspend/Resume Entry and Exit --

This core implements USB suspend and resume in the UTMI low-speed interface state machine (usbf_utmi_ls) and exposes status/control at the top level (usbf_top).

Entry into Suspend
- Full-Speed (FS): While in NORMAL, if the FS idle (J) state persists for >3.0 ms (T1_gt_3_0_mS), the controller asserts usb_suspend (suspend_set) and enters SUSPEND.
- High-Speed (HS): While in NORMAL, if inactivity persists for >3.0 ms (T1_gt_3_0_mS), the controller switches to FS transceiver/termination (xcv_set_fs, fs_term_on) and enters RES_SUSP. After >100 µs (T2_gt_100_uS) of stable J (j_long), it asserts usb_suspend and enters SUSPEND. If instead SE0 persists, a reset is detected (see below).
- Top-level export: usb_suspend is driven out on susp_o.

Exit from Suspend (Host-initiated resume)
- If a K state is detected while in SUSPEND (k_long), the state machine transitions to RESUME and immediately deasserts suspend (suspend_clr). Once SE0 is observed, it configures bit stuffing and (if applicable) returns to HS (xcv_set_hs, fs_term_off), then waits >100 µs in RESUME_WAIT (T2_gt_100_uS) before returning to NORMAL.
- UTMI SuspendM output is gated as: SuspendM = (usb_suspend & !resume_req_s) | (LineState == K). While a resume request is pending from the device, SuspendM is cleared to allow signaling.

Exit from Suspend (Device-initiated remote wakeup)
- A device resume request is provided on resume_req_i (latched internally as resume_req_r until suspend_clr). While in SUSPEND, if usb_suspend has been asserted for >5.0 ms (T1_gt_5_0_mS) and resume_req_s is set, the controller enters RESUME_REQUEST.
- After an additional wakeup delay (T2_wakeup ≈ 5.0 ms), the controller enters RESUME_SIG and

actively drives K on the bus (drive_k_d) for ≥1.0 ms (T2_gt_1_0_mS).
- It then transitions to RESUME (which asserts suspend_clr to clear resume_req_r at the top level), configures the link (bit_stuff_on, and HS re-enable if applicable), waits >100 µs in RESUME_WAIT, and returns to NORMAL.

Exit to Reset (from Suspend or Normal)
- From SUSPEND: If SE0 persists for >2.5 µs (T1_gt_2_5_uS with se0_long), the controller clears suspend (suspend_clr), asserts usb_reset, and enters RESET.
- From NORMAL (FS): If SE0 duration is between >2.5 µs and <3.0 ms and not an idle long condition, a RESET is recognized.
- After RESET, the controller performs speed negotiation (SPEED_NEG → SPEED_NEG_K/J transitions), then settles in HS or FS NORMAL as determined by chirp detection.

Timing Constants (implemented by counters)
- T1 thresholds: 2.5 µs, 3.0 ms, 3.125 ms, 5.0 ms (used for idle/suspend detection and minimum suspend time before remote wake).
- T2 thresholds: 100 µs (post-resume settle), ~5.0 ms (remote-wake readiness and wakeup delay), ≥1.0 ms (resume K signaling), 1.2 ms, and 100 ms (attach debounce). Exact counts are defined by macros and internal prescalers.

Software/Top-level Controls and Observability
- Input resume_req_i requests device-initiated resume. It is latched (resume_req_r) and automatically cleared when suspend_clr pulses from the UTMI state machine upon entering RESUME/RESET.
- Output susp_o reflects usb_suspend.
- Internal rf_resume_req exists in the register file, but in this top-level it is not wired into the UTMI resume path; only resume_req_i controls remote wake signaling.

UTMI and Transceiver Behavior During Transitions
- During suspend entry/exit the core switches transceiver settings and terminations appropriately (XcvSelect, TermSel) to meet FS/HS requirements, and manages OpMode (bit stuffing on/off) in alignment with signaling phases.

Summary
- Suspend entry: FS idle J >3.0 ms; HS inactivity >3.0 ms then FS-J >100 µs.
- Host resume: Detect K in SUSPEND → RESUME → wait SE0 → RESUME_WAIT >100 µs → NORMAL.
- Device remote wake: Allowed after ≥5.0 ms in SUSPEND if resume_req_i set; drive K for ≥1.0 ms; then RESUME and return to NORMAL.
- Reset detection from SUSPEND or NORMAL is handled per SE0 timing; speed renegotiation follows reset.

-- SOF Handling and Frame Timing --

Start-of-Frame (SOF) reception and frame timing are handled in the packet decoder (usbf_pd) and protocol layer (usbf_pl), with a software-visible summary provided in the register file (usbf_rf).

1) SOF token detection and validation
- The packet decoder (usbf_pd) captures the USB PID and classifies it. For SOF tokens, it latches the two token bytes (token0, token1) and computes CRC5 over {frame_address[6:0], endpoint[3:0]}.
- token_valid asserts when the token is fully received; crc5_err asserts if the inverted CRC5 field in the token (token1[7:3]) does not match the computed CRC5.
- The 11-bit frame number is reconstructed as frame_no = {token1[2:0], token0}.

2) Frame number update and microframe counting
- In usbf_pl, a valid SOF that passes CRC5 (token_valid && !crc5_err && pid_SOF) generates frame_no_we, then (registered to frame_no_we_r) updates frame_no_r with the newly received frame_no.
- mfm_cnt (4 bits) counts consecutive identical SOF frame numbers: it resets to 0 when the frame number changes (frame_no_we_r && !frame_no_same) and increments when successive SOFs carry the same frame number (frame_no_same). This provides a microframe index for high-speed operation (typically 0–7), though the counter is 4 bits wide and does not explicitly saturate.

3) Time-since-SOF measurement (sof_time)
- usbf_pl generates a periodic tick hms_clk from the PHY clock by counting cycles up to USBF_HMS_DEL (default 0x1C = 28). When hms_cnt equals this constant, hms_clk pulses and the counter resets.
- sof_time (12 bits) increments on each hms_clk pulse and is cleared on every new valid SOF (clr_sof_time = frame_no_we). Thus sof_time measures time elapsed since the last accepted SOF in units of hms_clk.
- Tick resolution depends on the PHY clock and USBF_HMS_DEL. With a 60 MHz UTMI clock and USBF_HMS_DEL=28, the tick is approximately 0.47–0.5 µs. The 12-bit width yields up to 4095 ticks (~2 ms at ~0.5 µs resolution), sufficient to cover the full-/high-speed inter-SOF intervals.

4) Software-visible framing data (frm_nat)
- usbf_pl packages a 32-bit framing/timing word frm_nat = {mfm_cnt[3:0], 1'b0, frame_no_r[10:0], 4'h0, sof_time[11:0]}.
• Bits [31:28]: mfm_cnt (microframe count within the current frame)
• Bit [27] : Reserved 0
• Bits [26:16]: frame_no_r (11-bit SOF frame number)
• Bits [15:12]: Reserved 0
• Bits [11:0] : sof_time (time since last valid SOF in hms_clk ticks)
- frm_nat is exposed through the main register bank in usbf_rf (select the main CSR window; within that window, use the sub-address that returns frm_nat). This allows firmware to read frame number, microframe index, and fine-grain time since SOF for scheduling and diagnostics.

5) Robustness and gating
- SOF-driven updates occur only when crc5_err is deasserted, ensuring corrupted SOFs do not disturb framing.
- The timebase (hms_cnt) is reset on each SOF boundary to keep sof_time aligned to the last accepted SOF.

6) Integration notes
- The effective time resolution depends on the UTMI PHY clock and the USBF_HMS_DEL macro; adjust USBF_HMS_DEL to change sof_time granularity if needed.
- mfm_cnt increments only when consecutive SOFs carry the same 11-bit frame number (typical of high-speed microframes); in full-speed operation it will remain at or near zero as frame numbers increment every SOF.

REGISTER MAP

-- Address Map Overview and Decoding --

Top-level Wishbone address map and decode (32-bit data, byte addressing). Address bus width: A[12:0]. Decoding is performed on A[12].

Region select
- Register File (RF): A[12] = 0
- Select expression: RF_SEL = !wb_addr_i[12]
- Addressing: only A[8:2] are used inside the RF (word index W = A[8:2]); A[11:9] and A[1:0] are ignored, so the RF is mirrored over A[11:9].
- Buffer Memory (SSRAM): A[12] = 1
- Select expression: MEM_SEL = wb_addr_i[12]
- Addressing: A[11:2] map to sram_adr_o[9:0] (10-bit word address). Size = 1024 words x 4 B = 4096 B.

RF region layout (word offsets W = A[8:2], byte offset = 4*W)
- W = 0x00..0x01: Main block (mirrored over two pages; lower 3 bits A[4:2] select register):
- A[4:2] = 0x0 (byte 0x00): Main CSR (RO): {line_stat[1:0], usb_attached, mode_hs, suspend}. Write to this same word (full 7-bit RF address == 0x00) with DIN[5]=1 issues a resume request (self-cleared by core).
- A[4:2] = 0x1 (byte 0x04): Function Address (FA) (R/W) DIN[6:0].
- A[4:2] = 0x2 (byte 0x08): Interrupt mask (R/W): inta_msk[8:0] (DIN[8:0]) and intb_msk[8:0] (DIN[24:16]).
- A[4:2] = 0x3 (byte 0x0C): Interrupt source summary (R/C): {int_srcb[8:0], int_srca[15:0]}. Read clears sources.
- A[4:2] = 0x4 (byte 0x10): Frame/microframe/time (frm_nat) (RO).
- A[4:2] = 0x5 (byte 0x14): UTMI vendor status (RO). Writing this same word sets utmi_vend_ctrl[3:0] and pulses utmi_vend_wr.
- A[4:2] = 0x6..0x7 (bytes 0x18..0x1C): Reserved.
- W = 0x02..0x03 (bytes 0x20..0x3C): Reserved (reads return 0).
- Endpoint windows (EP0..EP15): W pages 0x04..0x13, one page per endpoint, 4 words each. For endpoint i (i = 0..15):
- Base word index W_base = (0x04 + i) << 2. Byte base = 4*W_base = 0x40 + 0x10*i.
- Offsets within the EP window (A[3:2] = 0..3):
- +0x0 (byte +0x00): EPi CSR (R/W).
- +0x1 (byte +0x04): EPi INT status (R/C).
- +0x2 (byte +0x08): EPi BUF0 descriptor (R/W).
- +0x3 (byte +0x0C): EPi BUF1 descriptor (R/W).
- Notes: EP presence is compile-time configurable. If an endpoint is not implemented, its window reads as 0 and has no effect on write.
- Unused RF locations beyond W = 0x13 (bytes > 0x13C) are undefined.

Buffer Memory (SSRAM) region (A[12] = 1)
- Word address: sram_adr_o[9:0] = A[11:2]. Byte addressing within a word uses Wishbone byte selects (A[1:0] are ignored in address decode).
- Size: 4 KiB total (1024 x 32-bit words).

Notes
- RF aliasing: Because only A[8:2] are used for RF, A[11:9] are ignored—RF registers are mirrored across these upper RF bits.
- All RF and MEM accesses are 32-bit word aligned; A[1:0] are ignored by internal decode.

-- Global Control and Status --

Global Control and Status (GCSR)

- Address: RF base + 0x00 (internal rf adr[6:0] == 0x00)
- Width: 32-bit
- Access: Read returns PHY/USB status. Writes are ignored except as noted for bit 5.

Read fields (bits):
- [31:5] Reserved. Reads 0.
- [4:3] LineState (UTMI line state): 00=SE0, 01=J, 10=K, 11=SE1. Interpretation depends on mode:
- Full-speed: J=idle, K=resume signaling.
- High-speed: SE0 is idle between packets.
- [2] USB Attached: 1=Device attached/present; 0=not attached.
- [1] Mode HS: 1=High-speed mode; 0=Full-speed mode.
- [0] Suspend: 1=Core in suspend; 0=active.

Write behavior (bit 5 only):
- [5] Resume Request (write-only side-effect): Writing 1 generates an internal pulse to request resume (rf_resume_req). Auto-clears internally; reading GCSR does not reflect this bit (always reads 0). All other written bits are ignored.

Reset values:
- All defined status bits default to 0 after reset. LineState reflects UTMI input and may change as the PHY initializes.

Clocking/consistency notes:
- Read data is produced in the system (wclk) domain from status signals sourced in the PHY domain; values may change asynchronously. For stable observation, software may read twice and compare.

Dependencies:
- LineState, Attached, Mode HS, and Suspend are driven by the UTMI/link state machine (usbf_utmi_ls). Resume request via bit 5 is generated in the RF block; system integration must route it to PHY control if used.

-- Function Address Register --

Function Address Register (FAR)

Purpose:
- Holds the 7-bit USB device address assigned by the host (per USB Chapter 9). The packet layer compares incoming token addresses against this value to determine if the transaction is addressed to this device.

Location and Access:
- Address space: Register File (RF) region (i.e., Wishbone address bit 12 = 0 per `USBF_RF_SEL`).
- Word offset within RF: 0x01 (adr[6:0] == 7'h1). With 32-bit word addressing, this corresponds to byte offset 0x04 from the RF base.
- Bus width: 32 bits; only bits [6:0] are implemented.
- Access type: Read/Write on the Wishbone clock domain (wclk).

Reset Value:

- 0x00 (function address = 0). Note: This register is NOT automatically cleared on USB bus reset; software must update it upon handling a bus-reset event.

Read Behavior:
- Returns zero-extended function address: bits [6:0] = current device address; bits [31:7] = 0.

Write Behavior:
- On write, bits [6:0] of the data word are latched into the function address; bits [31:7] are ignored.
- No side effects; no interrupts are generated by writes.

Functional Use:
- The value drives funct_adr[6:0], which is fed to the packet layer (usbf_pl) as "fa" and compared against incoming token_fadr. Matching contributes to the internal match signal for endpoint processing.

Clock/Domain Notes:
- FAR is written in the Wishbone clock domain (wclk) and consumed in the PHY/UTMI clock domain via usbf_pl. Software should update FAR at safe times (e.g., when the bus is idle or immediately after a USB reset) to avoid transient mismatches.

Software Guidance:
- After a SET_ADDRESS request is accepted, write the assigned 7-bit address to FAR.
- On USB bus reset (indicated via RF interrupt sources — int_srcb[8] in this design), software should reinitialize FAR (typically to 0) before enumeration resumes.

-- UTMI Vendor Control/Status --

UTMI Vendor Control/Status provides a simple sideband bridge between software (WISHBONE/register-file domain) and vendor-specific UTMI PHY signals. It exposes a 32-bit register at RF address 0x05 (word address in the USB Function Core register file) with the following behavior:

- Read (RF[0x05]):
- Bits [7:0] return the most recently sampled vendor status bus VStatus[7:0].
- Bits [31:8] read as 0.
- Sampling/clocking: VStatus[7:0] originates in the PHY clock domain and is sampled into the WISHBONE clock domain each wclk cycle for reads. There is no handshake; software should tolerate asynchronous updates.

- Write (RF[0x05]):
- Bits [3:0] are taken as the new vendor control value VControl[3:0].
- Writing the register also generates a load/strobe pulse VControl_Load to inform the PHY that a new control value is available.
- Bits [31:4] are ignored on write.
- Clocking and pulse generation: the write occurs in the WISHBONE clock domain; the control value and a single update pulse are then registered into the PHY clock domain. Each write produces exactly one load pulse in the PHY clock domain.

Top-level signal mapping (usbf_top ports):
- VControl_pad_o[3:0] <= UTMI vendor control (driven from writes to RF[0x05], bits [3:0]).
- VControl_Load_pad_o <= UTMI vendor control load strobe (asserted once per write to RF[0x05]).
- VStatus_pad_i[7:0] => UTMI vendor status (reflected in reads of RF[0x05], bits [7:0]).

Clock/Reset domains:

- Software access (read/write of RF[0x05]) uses the WISHBONE clock (wclk).
- VControl[3:0] and VControl_Load are registered in the PHY/UTMI clock domain (clk = phy_clk).
- VStatus[7:0] is captured from the PHY clock domain into WISHBONE clock domain for readback. Initial value after reset is implementation-defined (no synchronous reset applied to the sampled status).

Software usage pattern:
1) To issue a vendor-specific command to the PHY, write RF[0x05] with the desired control nibble in bits [3:0]. This latches VControl[3:0] and produces a single VControl_Load pulse in the PHY clock domain.
2) To read vendor-specific status from the PHY, read RF[0x05] and examine bits [7:0]. Because the status is sampled without handshake, software may need to poll or read twice for stable values if the PHY updates the status near the read.

Notes and constraints:
- The meaning of VControl[3:0] and VStatus[7:0] is vendor-/PHY-specific and outside the USB core's standard operation; the core simply forwards control and mirrors status.
- Consecutive writes to RF[0x05] will generate a VControl_Load pulse per write.
- After reset, VControl[3:0] has no explicit default value until first write; VControl_Load is deasserted.
- This interface does not affect standard USB protocol handling and can be used for PHY diagnostics, test features, or vendor extensions.

-- Frame and Microframe Counters --

The core exposes a consolidated 32-bit frame/microframe/time register (frm_nat) derived from SOF tokens on the bus. It is generated in the PHY clock domain inside usbf_pl and made software-visible through the RF register block (usbf_rf) at the main register bank with adr[2:0] = 3'b100. Behavior and fields: 1) Source and qualification: frm_nat updates only on reception of a valid SOF token (token_valid asserted and CRC5 correct). SOFs are not filtered by device address; all observed SOFs on the bus are considered. 2) Frame number latch: On each qualified SOF, the 11-bit frame number from the token is captured into frame_no_r one cycle after the SOF detection. 3) Microframe counter (mfm_cnt[3:0]): Incremented when consecutive qualified SOFs carry the same 11-bit frame number (frame_no_same). Reset to 0 when the incoming SOF's frame number differs from the last latched frame_no_r. In full-speed operation this typically remains 0 (since FS SOFs increment the frame number each millisecond). In high-speed operation, if the host issues multiple SOFs per 1 ms frame that reuse the same frame number, this field counts those microframes. 4) SOF time-stamp (sof_time[11:0]): A coarse time counter that resets to 0 on each qualified SOF and then increments by one on a periodic tick (hms_clk). The tick is generated by a small prescaler driven by phy_clk; its period is set by the macro USBF_HMS_DEL (default 0x1C). sof_time thus measures elapsed time since the last SOF in units of the hms_clk period and wraps on 12-bit overflow. 5) Register layout (frm_nat[31:0]): [31:28] mfm_cnt (microframe count), [27] reserved (0), [26:16] frame_no_r (11-bit frame number), [15:12] reserved (0), [11:0] sof_time. 6) Reset behavior: On reset, frame_no_r, mfm_cnt, and sof_time are cleared to 0. sof_time is also cleared on each new qualified SOF. No updates occur on PID or CRC errors. This mechanism provides software with the current frame number, a microframe index (when applicable), and a coarse time since the last SOF for scheduling/diagnostics.

-- Interrupt Masks and Sources --

Function-level interrupting follows a two-source model: (1) endpoint-specific interrupts generated and masked inside each endpoint register file (EP0–EP15), and (2) core/global interrupts generated in the RF block. The top-level interrupt outputs are asserted when either source is active.

Top-level interrupt composition

- inta_o = (OR of epN_inta for N=0..15) OR (|(int_srcb & inta_msk))
- intb_o = (OR of epN_intb for N=0..15) OR (|(int_srcb & intb_msk))
- The endpoint contribution (epN_inta/intb) is already masked per-endpoint internally (via endpoint iena/ienb). The RF masks (inta_msk/intb_msk) apply only to the core/global sources (int_srcb).

Registers (RF space, main page)
- Interrupt Mask Register (INT_MASK) at main RF offset 0x02 (adr[2:0] == 3'h2)
- [8:0] INTA_MASK = inta_msk (din[8:0])
- [24:16] INTB_MASK = intb_msk (din[24:16])
- Reset: 0x0000_0000
- Function: Enables which core/global sources (int_srcb bits) are allowed to raise inta_o/intb_o. A masked-off bit will not contribute to the corresponding top-level interrupt.

- Interrupt Source Register (INT_SRC) at main RF offset 0x03 (adr[2:0] == 3'h3)
- Read returns: { [31:29]=0, [28:20]=int_srcb[8:0], [19:16]=0, [15:0]=int_srca[15:0] }
- int_srca[15:0]: Endpoint interrupt summary (read-only, not sticky, not clear-on-read)
- Bit n = epn_inta OR epn_intb (n=0..15). Reflects whether endpoint n currently asserts either A or B interrupt. Detailed reason and per-endpoint masking are contained in each endpoint's register file.
- int_srcb[8:0]: Core/global interrupt sources (sticky, clear-on-read)
- Reading INT_SRC clears all bits in int_srcb simultaneously.
- Reset: all bits 0.

Core/global interrupt sources (int_srcb bit definitions)
- [8] USB reset detected
- [7] Receive error (UTMI RxError observed)
- [6] Detach event (transition from attached to de-attached)
- [5] Attach event (transition to attached)
- [4] Suspend end (resume from suspend)
- [3] Suspend start (entering suspend)
- [2] Endpoint/address mismatch (token for non-matching function address/endpoint)
- [1] PID checksum error (PID CS failure)
- [0] Token CRC5 error

Operational notes
- Writing INT_MASK selects which core/global events can assert inta_o/intb_o via their respective masks; endpoint-generated interrupts are unaffected by these masks.
- Reading INT_SRC provides a snapshot of endpoint interrupt presence (int_srca) and the sticky core/global events (int_srcb). The read operation clears all int_srcb bits (read-to-clear). There is no per-bit clear; software should read INT_SRC after servicing recorded events.
- All masking and source latching for core/global events occurs in the wclk domain. Masks reset to 0 (disabled).

-- Endpoint Window (CSR, INT, BUF0, BUF1) --

Endpoint Window provides a 4x32-bit register set per endpoint (EP0..EP15): CSR, INT, BUF0, BUF1. Addressing within the register-file region: each endpoint window is selected by adr[6:2] = 0x04+EPn (EP0 at 0x04, EP1 at 0x05, ...), and adr[1:0] selects the register within the window: 0:CSR, 1:INT, 2:BUF0, 3:BUF1.

CSR (offset 0): 32-bit, R/W (some fields are HW-updated only)
- [31:30] UC_BSEL (read-only): user/current buffer selector mirrored by HW.
- [29:28] UC_DPD (read-only): data PID state mirrored by HW.

- [27:26] Endpoint Type/Direction: 00=Control, 01=IN, 10=OUT.
- [25:24] Transfer Type: 01=Isochronous, 10=Bulk, others reserved.
- [23:22] Endpoint State: 00=Enabled, 01=Disabled, 10=Stall, 11=Reserved. If OTS_STOP=1 and an OUT-too-small occurs, HW forces this field to 01 (Disabled).
- [21:18] Endpoint Number (EP address) compared against token EP.
- [17] Allow Large (> MaxPacketSize) OUT packets (1=allow, 0=NACK).
- [16] Allow Small (< MaxPacketSize) OUT packets (1=allow, 0=NACK).
- [15] DMA Enable (ignored for Control endpoints).
- [14] Reserved, reads 0.
- [13] OTS_STOP: if set, HW disables EP (CSR[23:22]=01) on OUT-too-small.
- [12:0] Reserved.
Notes: SW writes CSR update only [27:15], [13], [12:0]. UC_BSEL/UC_DPD ([31:28]) are updated by HW (read-only to SW).

INT (offset 1): 32-bit; read returns masks and sticky status; write updates masks; read clears status
- Readback layout: [31:30]=0, [29:24]=IENA[5:0] (mask for INTA), [23:22]=0, [21:16]=IENB[5:0] (mask for INTB), [15:7]=0, [6:0]=INT_STAT[6:0].
- Write: din[29:24] -> IENA, din[21:16] -> IENB. Other bits ignored on write.
- INT_STAT bits (set by HW, cleared on read of INT):
- [6] OUT_TOO_SMALL: OUT packet size != MaxPacketSize during DMA.
- [5] PID_SEQ_ERR: data PID sequencing error.
- [4] BUF1 Event (buffer update/completion).
- [3] BUF0 Event (buffer update/completion).
- [2] UNEXPECTED_PID: token PID not matching EP direction/type.
- [1] CRC16_ERR: data CRC error.
- [0] TIMEOUT: IN ACK timeout or OUT data transmit timeout.
- Interrupt outputs: INTA and INTB are asserted when any (INT_STAT[i] & IEN? mask) is set. Note: BUF1 ([4]) and BUF0 ([3]) share the same mask bit index 3 in both IENA and IENB.

BUF0 (offset 2), BUF1 (offset 3): 32-bit buffer descriptors, R/W (HW updates during transfers)
- [31] NA (Not Available): 1 marks buffer unavailable (SW convention). HW also treats an all-ones address as not allocated.
- [30:17] SIZE (14 bits): buffer size in bytes.
- [SSRAM_HADR+2:4] ADR[high]: start address high bits into external SSRAM window.
- [3:0] Mixed-use low nibble:
- On SW write (initialization) and on HW address updates (buf_set): holds ADR[3:0].
- On HW UC status updates: holds {UC_DPD[1:0], UC_BSEL[1:0]}.
Notes/behavior:
- BUF0_ORIG latches the last SW-written BUF0; when HW asserts buf0_rl, BUF0 is reloaded from BUF0_ORIG.
- During operation, HW updates BUF0/BUF1 on buffer completion or special conditions (e.g., OUT-too-small).
- Software should program SIZE and ADR fields; bit[31] may be used to mark unavailability; an address of all ones also denotes not allocated.

Reset values
- CSR: all zeros (UC_BSEL/UC_DPD=0; OTS_STOP=0; EP disabled until configured).
- INT: IENA=0, IENB=0, INT_STAT=0.
- BUF0/BUF1: 0xFFFF_FFFF (treated as not allocated).

-- DMA Request/Acknowledge Mapping --

DMA request/acknowledge is exposed as a 16■bit vector per endpoint at the top level. Mapping and handshake are as follows:

- Buses
- dma_req_o[15:0]: One request bit per endpoint. Bit N corresponds to endpoint N (0..15).
- dma_ack_i[15:0]: One acknowledge bit per endpoint. Bit N acknowledges endpoint N.
- For endpoints not compiled in (USBF_HAVE_EPx undefined), the corresponding dma_req_o bit is permanently 0 and the dma_ack_i bit is ignored.

- Clocking and polarity
- Request and acknowledge are active■high and synchronous to the WISHBONE/system clock domain wclk (top-level clk_i).
- Internally, each dma_ack_i[N] is also synchronized into the PHY clock domain for counter bookkeeping.

- When requests are generated
- Requests are emitted only if the endpoint's DMA enable (CSR bit csr[15]) is set and the endpoint is not a control endpoint (csr[27:26] != 2'b00).
- OUT endpoints (csr[27:26] == 2'b10): dma_req_o[N] asserts when there is at least one 32■bit word of received packet data to be transferred out by the system DMA.
- IN endpoints (csr[27:26] == 2'b01): dma_req_o[N] asserts when the endpoint needs data to be filled by the system DMA (i.e., current filled-word count is below the programmed packet length).

- Acknowledge semantics (what a single ack means)
- The system must pulse dma_ack_i[N] high for exactly one wclk cycle for each 32■bit word transferred by the system DMA on behalf of endpoint N.
- For OUT endpoints, an ack pulse denotes that the system has consumed one 32■bit word from the endpoint buffer.
- For IN endpoints, an ack pulse denotes that the system has provided one 32■bit word to the endpoint buffer.

- Request deassertion and burst/hold behavior
- While dma_req_o[N] is high, the endpoint may request a burst: a hold condition keeps the request asserted across multiple acks until the internal word counters reach their termination conditions.
- Deassertion rule: dma_req_o[N] will deassert after an ack pulse when the endpoint's internal hold condition is false; otherwise it remains asserted to continue the burst.
- Hold conditions (internal):
- OUT: request stays asserted while there are outstanding words to be serviced (roughly, when the remaining OUT word-count is non■zero; internally held for more than a few words to encourage bursts).
- IN: request stays asserted until the filled-word count approaches the programmed buffer length (internally held while fewer than (length–3) words have been provided).

- Correct usage requirements for external DMA
- Only assert dma_ack_i[N] when dma_req_o[N] is currently high.
- Provide exactly one ack pulse per 32■bit word moved.
- Ack pulses must be synchronous to wclk and at least one clock wide; multi■cycle high is allowed but is treated as a single handshake per cycle high.
- Do not generate acks when DMA is disabled (csr[15]==0) or for control endpoints; no requests will be generated in those cases.

- Reset behavior

- On reset, dma_req_o[*] are deasserted and internal handshake/counter state is cleared.

Summary mapping
- dma_req_o[0] ↔ dma_ack_i[0]: Endpoint 0
- dma_req_o[1] ↔ dma_ack_i[1]: Endpoint 1
- …
- dma_req_o[15] ↔ dma_ack_i[15]: Endpoint 15

Note: If an endpoint is not synthesized (USBF_HAVE_EPx undefined), its request line remains 0 and its ack input has no effect.

-- Access Semantics (R/W, W1C, Side Effects) --

- Address decode: wb_addr_i[12]==0 selects the Register File (RF) space exposed by usbf_rf (adr = wb_addr_i[8:2]); wb_addr_i[12]==1 selects external buffer memory (SSRAM) with normal RW semantics (no special side effects beyond memory read/write).
- RF global registers (adr[6:2]==0x00/0x01, subselected by adr[2:0]):
- 0x00.main_csr (RO): {line_state[1:0], usb_attached, mode_hs, suspend}. Read-only; no side effects.
- 0x00.funct_adr (RW): 7-bit device address. Write sets the value; read returns it; no side effects.
- 0x00.int_mask (RW): inta_msk[8:0] and intb_msk[8:0]. Write updates masks; read returns; no side effects.
- 0x00.int_src (int sources): returns {int_srcb[8:0], int_srca[15:0]}.
- int_srcb (R1C): sticky global source bits set by events (usb_reset, rx_err, attach/deattach, suspend start/end, nse_err, pid_cs_err, crc5_err). Read of this register (re at adr==0x03) clears int_srcb bits (Read-to-Clear) and may deassert global interrupts.
- int_srca (RO): per-endpoint summary (epN_inta|epN_intb). Read-only; not cleared by read.
- 0x00.frm_nat (RO): frame/microframe/timestamp info. Read-only; no side effects.
- 0x00.utmi_vendor (RW/WO/RO combo):
- Read returns UTMI vendor status (RO).
- Write din[3:0] updates utmi_vend_ctrl and generates a single-cycle utmi_vend_wr strobe (side effect). No defined readback of control.
- 0x00.resume_req (WO, self-clearing): write to adr==0x00 with din[5]==1 sets rf_resume_req; it auto-clears when rf_resume_req is observed by the UTMI/link (side effect). Reading returns only main_csr (no readback of this bit).
- Endpoint register blocks (adr[6:2]==0x04..0x13 map EP0..EP15; subregister selected by adr[1:0]):
- +0: EP_CSR (mixed RW/RO):
- Writable fields: csr0[12:0], ots_stop (bit13), csr1[27:15].
- Read-only within CSR image: uc_bsel[31:30], uc_dpd[29:28] (updated only by hardware via uc_*_set from protocol engine).
- Side effects: if ots_stop=1 and an out_to_small event occurs, hardware forces csr1[8:7]=2'b01.
- +1: EP_INT (RW + R1C): read returns {iena, ienb, int_stat}.
- iena[29:24], ienb[21:16] (RW): write updates enables; read returns current.
- int_stat[6:0] (R1C): sticky per-EP event bits (timeout, CRC16, PID/seq error, buffer events, out_to_small). A read of EP_INT (re at adr sub==+1) clears int_stat (Read-to-Clear) and may deassert ep interrupts.
- +2: BUF0 (RW, HW-updated): software write sets buf0 and also latches buf0_orig to the same value (side effect). Hardware may update buf0 on buf0_set/buf0_rl events.
- +3: BUF1 (RW, HW-updated): software write sets buf1. Hardware may update buf1 on buf1_set or out_to_small.
- Notes on BUFx side effects: hardware modifies addresses/sizes and toggles uc_bsel/uc_dpd via uc_*_set; these affect DMA/buffer sequencing. Reading BUFx has no side effects.
- Interrupt outputs:

- inta_o/intb_o are asserted from per-EP inta/intb and masked RF sources. Reading EP_INT (per-EP) clears that EP's int_stat (R1C). Reading global int_src clears only int_srcb (R1C). No W1C fields are implemented.
- Wishbone interface timing/acks: wb_ack_o pulses after internal state machine completes. RF reads assert rf_re; RF writes assert rf_we. No additional side effects beyond those listed above.

-- Reset Values and Field Definitions --

Register file is selected when wb_addr_i[12] == 0. All offsets below are word indices taken from wb_addr_i[8:2]. Unless stated, fields are 0 after reset. Endpoint blocks (EP0..EP15) are replicated; reset values are identical for all endpoints.

Top-level RF registers
- 0x00 MAIN_CSR (R; W has side-effect on bit 5 only)
- [31:5] R: 0
- [4:3] R: LineState[1:0] (mirrors UTMI line state)
- [2] R: usb_attached (1 when attached)
- [1] R: mode_hs (1=High-Speed, 0=Full-Speed)
- [0] R: suspend (1 when suspended)
- Write side-effect: writing a 1 to bit [5] latches a resume request (rf_resume_req). Cleared automatically when consumed. All other bits are ignored on write. Reset: readback shows [4:0]=0; write-side latch reset to 0.

- 0x01 FUNCT_ADR (R/W)
- [31:7] 0
- [6:0] Function address. Reset: 0x00.

- 0x02 RESERVED (R=0, W ignored)
- 0x03 RESERVED (R=0, W ignored)

- 0x04 FRAME_NAT (R)
- [31:28] Microframe count within current frame (increments on repeated SOF of same frame number)
- [27] 0
- [26:16] Last latched frame number
- [15:12] 0
- [11:0] SOF timing counter (host milli/µs tracking)
Reset: 0x00000000.

- 0x05 UTMI_VENDOR (R: status; W: control)
- Read: [31:8]=0, [7:0]=utmi_vend_stat (mirrors VStatus_pad_i). Reset readback depends on external pad; unspecified in RTL.
- Write: [3:0] utmi_vend_ctrl; write also asserts a one-shot utmi_vend_wr strobe. Reset value of utmi_vend_ctrl is unspecified (no explicit reset); write before use.

- 0x06..0x07 RESERVED (R=0, W ignored)

- 0x08..0x25 EPx blocks (EP0..EP15)
Each endpoint x has 4 registers at base = 0x04 + x:
base+0: EPx_CSR (R/W)
- Layout (read):
- [31:30] uc_bsel (R/O): internal user buffer select (updated by core)
- [29:28] uc_dpd (R/O): internal DATA PID state (updated by core)

- [27:26] EP direction/type (R/W): 00=Control, 01=IN, 10=OUT, 11=Reserved
- [25:24] Transfer type (R/W): 01=Isochronous, 10=Bulk, 00/11=Other/Reserved
- [23:22] Endpoint status (R/W): 00=Normal, 01=Disabled, 10=Stall, 11=Reserved
- [21:18] Endpoint address (R/W): 4-bit endpoint number this CSR controls
- [17] lrg_ok (R/W): allow OUT larger than MaxPacket (1=allow)
- [16] sml_ok (R/W): allow OUT smaller than MaxPacket (1=allow)
- [15] dma_en (R/W): enable DMA for non-control endpoints (ignored for Control)
- [14] Reserved, reads as 0, writes ignored
- [13] ots_stop (R/W): if set and an OUT packet is smaller than MaxPacket, core auto-sets status to Disabled (01)
- [12:11] tr_fr (R/W): data PID sequencing mode (used by core for PID selection)
- [10:0] max_pl_sz (R/W): MaxPacketSize
- Writes: only csr1 ([27:15]), ots_stop ([13]), and csr0 ([12:0]) are writable; uc_bsel/uc_dpd ([31:28]) are read-only and updated by hardware. Reset: all zeros (uc_bsel=0, uc_dpd=0, dir=Control, type=0, status=Normal, ep_addr=0, lrg_ok=0, sml_ok=0, dma_en=0, [14]=0, ots_stop=0, tr_fr=0, max_pl_sz=0).

base+1: EPx_INT (R/WM)
- Read returns: { [31:30]=0, [29:24]=IENA[5:0], [23:22]=0, [21:16]=IENB[5:0], [15:7]=0, [6:0]=INT_STAT }
- INT_STAT (latched, W1C-on-read by entire field):
- [6] OUT_TOO_SMALL (out_to_small)
- [5] SEQ_ERR (int_seqerr_set)
- [4] BUF1_DONE (int_buf1_set)
- [3] BUF0_DONE (int_buf0_set)
- [2] UNEXPECTED_PID (int_upid_set)
- [1] CRC16_ERR (int_crc16_set)
- [0] TIMEOUT (int_to_set)
- IENA/IENB (masks for inta/intb aggregation): asserting a bit enables that source to contribute to inta/intb respectively.
- Write: only masks are writable: IENB <= din[21:16], IENA <= din[29:24]. Reading this register clears INT_STAT (all 7 bits). Reset: IENA=0x00, IENB=0x00, INT_STAT=0x00.

base+2: EPx_BUF0 (R/W)
- Buffer descriptor for buffer 0.
- [31] NA (1=Not allocated/Not available)
- [30:17] SIZE (bytes)
- [SSRAM_HADR+2:0] BYTE_ADDRESS into shared SRAM (default build SSRAM_HADR=9 → bits [11:0])
- Note: core updates this descriptor during transfers (address, size, and low nibble may be updated to track progress and PID/buffer select). Software writes initial descriptor; hardware maintains it thereafter. Reset: 0xFFFF_FFFF (NA=1, address undefined).

base+3: EPx_BUF1 (R/W)
- Same format and behavior as BUF0 for buffer 1. Reset: 0xFFFF_FFFF.

Interrupt source aggregation (top-level RF register 0x03)
- Read: { [31:29]=0, [28:20]=INT_SRCB[8:0], [19:16]=0, [15:0]=INT_SRCA[15:0] }
- INT_SRCB (latched; cleared on read of 0x03):
- [8] USB_RESET detected
- [7] RX_ERROR (UTMI RxError)
- [6] DETACH (usb_attached falling edge)

- [5] ATTACH (usb_attached rising edge)
- [4] SUSPEND_END (usb_suspend falling edge)
- [3] SUSPEND_START (usb_suspend rising edge)
- [2] NO_SUCH_ENDPOINT (nse_err)
- [1] PID_CHECKSUM_ERR (pid_cs_err)
- [0] TOKEN_CRC5_ERR (crc5_err)
- INT_SRCA (live OR, not latched/cleared): bit i reflects (EPi_INTA | EPi_INTB). Reset readback = 0 until endpoints raise interrupts.

Address decoding summary
- RF region: wb_addr_i[12] == 0
- Register index within RF: wb_addr_i[8:2]
- Endpoint block x (0..15): base = 0x04 + x; subregister offsets: +0 CSR, +1 INT, +2 BUF0, +3 BUF1

Notes
- uc_bsel and uc_dpd in EPx_CSR are managed by hardware; software reads them but does not write them.
- EP INT_STAT and top-level INT_SRCB are cleared by read (whole-field clear on read access to their respective registers); masks retain their values.
- UTMI vendor control (0x05 write) has no defined reset value; write before use.
- Default build parameters in this image: SSRAM_HADR=9 (12-bit byte address in BUFx), WB RF/MEM select bit at wb_addr_i[12].

-- Endianness and Alignment --

Data path endianness is little■endian for buffer memory: the first byte of a stream maps to D[7:0] of the first 32■bit word, the second to D[15:8], third to D[23:16], and fourth to D[31:24]. UTMI is an 8■bit byte stream (no byte swapping on the UTMI interface); CRC blocks handle any bit order internally. Buffer descriptors store byte addresses: addr[SSRAM_HADR+2:4] hold high address bits and, when a buffer update occurs (buf*_set), addr[3:0] captures the low nibble. The low nibble [3:0] is also used to convey UC_DPD/UC_BSEL when uc_*_set is asserted, so software must write the address low nibble via buf*_set operations.

Alignment rules:
- Internal DMA supports unaligned buffer addresses at byte granularity. It performs read■modify■write on the first/last partial words to preserve untouched bytes.
- External SRAM interface is 32■bit word■addressed; sram_adr_o is a word address and writes are full 32■bit words (no byte enables).
- Wishbone memory/register accesses are 32■bit word accesses only; lower address bits [1:0] are ignored (memory via ma_adr[SSRAM_HADR+2:2], registers via ma_adr[8:2]). Software must not rely on byte■selects.

Implication: software reading/writing packet data via the memory window should interpret 32■bit words in little■endian byte order; unaligned buffers are allowed and handled by the core, but CPU■side memory writes must be full words.

INTERFACE SPECIFICATIONS

-- WISHBONE Slave Interface --

32-bit WISHBONE B3/B4 classic slave, synchronous to wb_clk.

Signals
- Inputs: wb_clk, rst (active-high), wb_addr_i[12:0], wb_data_i[31:0], wb_we_i, wb_stb_i, wb_cyc_i
- Outputs: wb_data_o[31:0], wb_ack_o
- No SEL_I/ERR_O/RTY_O/STALL_O; only single-word classic cycles supported.

Addressing and map (USBF_UFC_HADR=12)
- 13-bit address bus (A12..A0), 32-bit data.
- Region select by A[12]:
- A[12]=0: Register space (RF)
- Effective register index = A[8:2] (word-aligned). A[1:0] ignored. Space mirrored over A[11:9].
- 0x00.. word map drives core control/status, endpoint windows, etc. (see core register map).
- A[12]=1: Buffer memory window (SSRAM)
- Effective memory index = A[11:2] (word-aligned). A[1:0] ignored.
- Depth set by USBF_SSRAM_HADR=9 → 1024 words (4 KiB) accessible.
- Full 32-bit reads/writes only; no byte enables.

Data routing
- Reads: wb_data_o is muxed from RF or buffer memory by A[12].
- Writes: RF writes assert rf_we; memory writes assert ma_we and target SSRAM via arbiter.
- Byte ordering in buffer memory: the byte at address +0 maps to wb_data_o[7:0] (little-endian within 32-bit word).

Handshake and timing
- wb_ack_o is a single-cycle pulse per transfer.
- Typical latency: 1–3 wb_clk cycles from STB/CYC assertion to ACK, depending on clock-domain crossing and target (RF vs. memory).
- No STALL_O: master must hold CYC/STB until ACK.

Clock domains and arbitration
- Requests are sampled into the PHY clock domain for servicing; ACK is returned in wb_clk domain.
- Buffer memory accesses arbitrate with internal DMA on the PHY side. DMA has priority; Wishbone memory accesses may be delayed until DMA releases the SRAM.

Reset
- rst_i synchronously clears internal state; do not access bus while in reset.

-- UTMI+ PHY Interface --

UTMI+ PHY Interface provides the physical-layer attachment for the USB Function core. It wraps a standard 8-bit UTMI+ data path and control/status sideband and implements line-state management, speed negotiation, suspend/resume, and PHY handshakes in the PHY clock domain.

Clock and reset
- phy_clk_pad_i is the only clock for the UTMI interface logic; all UTMI signals are registered on this clock.
- phy_rst_pad_o is a direct copy of rst_i and should be used to reset the external PHY as required.

UTMI+ data and handshakes
- Transmit (device to host): DataOut_pad_o[7:0], TxValid_pad_o, TxReady_pad_i.
- DataOut is driven with the first byte when either TxReady is asserted or a new packet starts (tx_first). Subsequent bytes are presented each cycle while TxReady is high.
- TxValid is asserted for the duration of a packet and remains asserted until the PHY acknowledges (TxReady) or while a K-chirp is being driven during resume signaling.
- Receive (host to device): DataIn_pad_i[7:0], RxValid_pad_i, RxActive_pad_i, RxError_pad_i are sampled and synchronized into rx_data, rx_valid, rx_active, rx_err.

Line-state, mode, and termination control (managed by usbf_utmi_ls)
- LineState_pad_i[1:0] is decoded each cycle to detect J/K/SE0 conditions and idle.
- XcvSelect_pad_o selects transceiver mode (FS vs HS) during and after speed negotiation.
- TermSel_pad_o controls FS terminations (enabled when operating FS as per state machine).
- OpMode_pad_o drives the UTMI OpMode: normal operation when bit-stuffing is enabled; non-driving when disabled (used during reset/suspend/negotiation windows).
- SuspendM_pad_o controls PHY suspend per UTMI+ convention; it is asserted during suspend and deasserted when resuming/active (the wrapper internally gates resume requests and line-state conditions).
- usb_vbus_pad_i is monitored to gate attach/por behavior.

Suspend and resume
- The core accepts a resume request (resume_req_i). usbf_utmi_ls generates a resume sequence (RESUME_REQUEST → RESUME_SIG → RESUME) observing T2 timing; during RESUME_SIG a K-chirp is driven via an internal drive_k signal, which forces TxValid and a defined DataOut pattern.
- suspend_clr is produced to clear the latched resume request once the PHY exits suspend.
- USB suspend, reset, and attach status are exported internally (usb_suspend, usb_reset, usb_attached) and used by higher layers.

High-/Full-speed negotiation
- After SE0 reset, the state machine performs HS detection using alternating K/J chirps; on success it asserts mode_hs and configures XcvSelect/TermSel/OpMode for HS. Otherwise it selects FS and enables FS terminations.

Vendor sideband
- VStatus_pad_i[7:0] is sampled each wclk cycle into the register file for software visibility.
- VControl_pad_o[3:0] and VControl_Load_pad_o provide a programmable vendor control nibble and write strobe to the PHY.

Integration notes
- All UTMI signals are synchronous to phy_clk.
- The wrapper ensures UTMI timing requirements on DataOut/TxValid relative to TxReady at packet start.
- Line-state and timing constants (T1/T2) are parameterized via macros and implement USB 2.0 suspend, reset, resume, and negotiation timings.

-- External SSRAM Interface --

- Purpose: External single-port buffer memory used for endpoint data (RX/TX) and Wishbone debug/host access. The interface is arbitrated between the USB packet engine (IDMA) and the Wishbone side.
- Signals (top-level):
- sram_adr_o[SSRAM_HADR:0]: Word address (32-bit word indexing; byte lanes are not exposed).

- sram_data_o[31:0]: Write data to SRAM.
- sram_data_i[31:0]: Read data from SRAM.
- sram_we_o: Active-high write enable.
- sram_re_o: Read enable, held high continuously.
- Parameterization and size:
- Address width is parameter SSRAM_HADR = `USBF_SSRAM_HADR (effective value in this build: 9). This yields A9:0 = 1024 words x 32 bits = 4 KiB total.
- Addressing and endianness:
- External addressing is word-based: sram_adr_o = internal_byte_address[SSRAM_HADR+2:2].
- No byte enables; all writes are full 32-bit words.
- Little-endian mapping within each word (byte 0 at bits [7:0]).
- Clocking and timing expectations:
- No explicit SRAM clock signal is provided; the arbiter (usbf_mem_arb) runs in the PHY clock domain.
- Read behavior: present sram_adr_o with sram_we_o=0; read data must be valid on sram_data_i one phy_clk cycle later (assumes 1-cycle synchronous read latency or compliant flow-through timing).
- Write behavior: drive sram_adr_o and sram_data_o with sram_we_o=1 for one cycle for a full-word write.
- sram_re_o is tied high; use it as a constant chip/read-enable as appropriate for the memory device.
- Arbitration and access model:
- usbf_mem_arb multiplexes two masters onto the single SRAM port:
- M (USB IDMA, PHY clock): fixed priority; mack asserts immediately when mreq is high; address must be held to return data next cycle.
- W (Wishbone via usbf_wb): granted only when M is idle; wack is generated in PHY clock and resynchronized to WB clock.
- Both masters see read data directly from sram_data_i.
- Integration notes:
- Implement the memory as a 32-bit, single-port synchronous RAM with 1-cycle read latency (or wrap a device-specific SSRAM to match this handshake). Map its clock to the PHY clock domain.
- No chip-select or byte-enable signals are provided; if required by the memory, derive them from sram_re_o (always 1) and sram_we_o.
- Wishbone memory window selection is done above this interface (wb_addr_i[12] selects memory space); external mapping does not affect the SRAM pins.

-- Interrupt Outputs (inta_o, intb_o) --

inta_o and intb_o are the two external, active-high, level interrupts generated by the USB function core. They are synchronous to the WISHBONE clock (wclk = clk_i) and are asserted whenever any enabled interrupt source is pending. Each line is the logical OR of:
- Endpoint-level masked interrupt requests (from EP0..EP15)
- Core-level (RF) masked interrupt events

Aggregation and masking
- inta_o = (OR over all enabled endpoint-A interrupts) OR (OR over enabled RF-A events)
- intb_o = (OR over all enabled endpoint-B interrupts) OR (OR over enabled RF-B events)
- Outputs update on the rising edge of wclk and remain asserted while any contributing source remains set and masked in.

Endpoint-level interrupts (per endpoint)
- Each endpoint (EP0..EP15; only instantiated endpoints contribute) has two interrupt outputs (A and B) that feed the aggregate inta_o/intb_o.
- Per-endpoint event sources (latched in the PHY clock domain, reported at wclk):
0) Timeout (ACK timeout after IN, or TX data timeout during OUT)

1) CRC16 error on received data
2) Unexpected PID (UPID)
3) BUF0 update/completion
4) BUF1 update/completion
5) PID sequence error
6) OUT packet smaller than max payload (out_to_small)
- Per-endpoint mask registers (written via the endpoint register space) control which events drive the A and B lines:
- iena[5:0] masks EP-A; ienb[5:0] masks EP-B
- Event-to-mask mapping:
• [0] Timeout -> mask bit 0
• [1] CRC16 error -> mask bit 1
• [2] Unexpected PID -> mask bit 2
• [3] BUF0 -> mask bit 3
• [4] BUF1 -> mask bit 3 (shares mask with BUF0)
• [5] Sequence error -> mask bit 4
• [6] OUT too small -> mask bit 5
- Clearing endpoint events: reading the endpoint INT register (adr[1:0] == 2) clears that endpoint's latched event bits.
- Reset state: iena and ienb default to 0 (endpoint interrupts disabled until enabled by software).

Core-level (RF) interrupts
- RF event sources (latched, cleared on read of the RF INT source register):
[8] USB reset detected
[7] RX error (UTMI RxError)
[6] Detach (usb_attached falling)
[5] Attach (usb_attached rising)
[4] Exit suspend (suspend end)
[3] Enter suspend (suspend start)
[2] NSE error (protocol layer)
[1] PID check error (pid_cs_err)
[0] Token CRC5 error
- Separate 9-bit masks for A and B:
- inta_msk[8:0] controls which RF events contribute to inta_o
- intb_msk[8:0] controls which RF events contribute to intb_o
- Register access (RF register space, wclk domain):
- Write masks at RF address 0x02: intb_msk <= din[24:16], inta_msk <= din[8:0]
- Read INT source at RF address 0x03: returns {int_srcb, int_srca}; reading clears only RF event latches (int_srcb), not endpoint latches
- Reset state: inta_msk and intb_msk default to 0 (RF interrupts disabled until enabled by software).

Software visibility and clearing
- The RF INT source readback (RF addr 0x03) returns:
- int_srca[15:0] = per-endpoint summary bits (epN_inta OR epN_intb), one bit per endpoint
- int_srcb[8:0] = RF event bits described above
- Reading RF addr 0x03 clears int_srcb but does not clear any endpoint event bits; endpoint events must be cleared by reading each endpoint's INT register.

Summary
- Both inta_o and intb_o are level, synchronous, and represent the OR of enabled endpoint and RF sources.
- All masks default to 0 after reset; no interrupts will assert until software programs masks.

- RF event latches clear on read of RF INT source; endpoint event latches clear on read of the per-endpoint INT register.
- Only instantiated endpoints contribute to the aggregate outputs; dummy (non-present) endpoints contribute zero.

-- DMA Handshake (dma_req_o, dma_ack_i) --

Purpose: Provide a per-endpoint DMA service request/acknowledge handshake between the USB function core and an external DMA (or software DMA) agent for moving 32-bit words to/from the core's endpoint buffers in the shared SSRAM. Interface: dma_req_o[15:0] (output, active-high, wclk domain = clk_i) and dma_ack_i[15:0] (input, active-high pulse, wclk domain = clk_i). Bit mapping: bit 0 = EP0, bit N = EPN (0..15). Reset: On rst, requests deassert; acks should be inactive.
Operation: While an endpoint is DMA-enabled (csr[15]=1) and configured as IN (csr[27:26]=2'b01) or OUT (2'b10), the core asserts dma_req_o[n] when service is needed:
- OUT endpoint (device receives data from USB host): dma_req_o[n] asserts when there is received data in the endpoint buffer that must be drained to the system. Each dma_ack_i[n] pulse indicates the external DMA has removed one 32-bit word from the buffer; the core decrements its internal OUT word counter. When the counter reaches zero, the request can deassert.
- IN endpoint (device sends data to USB host): dma_req_o[n] asserts when the endpoint buffer needs to be filled by the system. Each dma_ack_i[n] pulse indicates one 32-bit word has been written into the buffer; the core increments its internal IN word counter. When sufficient data is available (up to the programmed buffer/packet size), the request can deassert.
Handshake semantics: dma_req_o is level-sensitive; it may remain asserted across multiple acks to allow streaming bursts. The external DMA may pulse dma_ack_i[n] on any wclk cycle while dma_req_o[n] is high; each pulse must correspond to exactly one completed 32-bit data transfer to/from the endpoint buffer. The core includes hold logic that can keep dma_req_o[n] asserted after an ack if more words are immediately needed/available (no need to wait for a reassert between beats). Ack pulses should be at least one wclk cycle wide and deasserted between pulses. Acks received while the corresponding request is low are ignored.
Clocking and CDC: Both dma_req_o and dma_ack_i are in the wclk (clk_i) domain. The core internally synchronizes the ack into the PHY clock domain to update counters safely; no special CDC handling is required by the external DMA.
Direction and sizing: The handshake itself does not encode direction. Software/driver or the DMA engine should use endpoint CSR fields to determine direction (csr[27:26]) and sizes (max payload size csr[10:0] and buffer descriptors buf0/buf1). Transfers are 32-bit word granular; one dma_ack_i pulse == one 32-bit word.
Servicing data: Actual data movement occurs via the core's buffer memory (SSRAM) through the Wishbone/memory port; the DMA handshake only signals demand and completion. The external agent must ensure each ack corresponds to a successful 32-bit read (OUT) or write (IN) at the correct buffer address.
Throughput and flow control: The core exposes additional status (e.g., dma_in_buf_sz1, dma_out_buf_avail via internal paths) to manage flow, but these are internal to the core. The external DMA should continue issuing ack pulses until dma_req_o deasserts, indicating the current DMA need is satisfied.

-- Clocks and Resets (clk_i, phy_clk_pad_i, rst_i) --

- Overview: The core operates with two independent clock domains and a single active-low reset:
- clk_i: System/WISHBONE clock domain (register/file access, interrupts, DMA request handshake, system-side memory access requests).
- phy_clk_pad_i: UTMI PHY clock domain (UTMI interface, packet layer, endpoint engines, internal DMA, and the physical SSRAM port).

- rst_i: Active-low reset distributed to all submodules; also driven out as phy_rst_pad_o to the PHY.

- Reset characteristics:
- Polarity: Active-low (logic 0 asserts reset).
- Default behavior: Synchronous reset inside modules (reset checked as if(!rst)). If compiled with USBF_ASYNC_RESET, some flops use asynchronous, active-low reset (posedge clk or negedge rst).
- External PHY reset: phy_rst_pad_o mirrors rst_i.
- Reset recommendation: Assert rst_i low long enough to be sampled by at least one rising edge of both clk_i and phy_clk_pad_i to guarantee clean reset in both domains (more conservative: a few cycles of the slower clock).

- Clock domain usage and crossings:
- UTMI/PHY domain (phy_clk_pad_i): usbf_utmi_if, UTMI line-state and speed negotiation (usbf_utmi_ls), packet layer (usbf_pl), packet decoder/encoder (usbf_pd/usbf_pa), protocol engine (usbf_pe), internal DMA (usbf_idma), endpoint control/state in usbf_rf (clk port), and the memory arbiter's PHY side (usbf_mem_arb.phy_clk). The external SSRAM interface (sram_adr_o/sram_data_o/sram_we_o/sram_re_o) is driven in this domain.
- System/WISHBONE domain (clk_i): WISHBONE interface/state (usbf_wb.wb_clk), register file host access side (usbf_rf.wclk), DMA request/ack interface to system (dma_req_o/dma_ack_i), and the memory arbiter's WB side (usbf_mem_arb.wclk).
- CDC mechanisms:
- WISHBONE requests (wb_stb_i & wb_cyc_i) are sampled into the PHY domain in usbf_wb; acknowledge is generated in the PHY domain and returned to clk_i using a small pipeline (wb_ack_* flops) to create a single-cycle ack pulse in clk_i.
- DMA request/ack between usbf_rf (clk_i side) and external DMA uses handshake flops in wclk and clk domains to safely cross.
- Memory arbitration (usbf_mem_arb) arbitrates PHY-side IDMA and WB-side memory accesses; requests from WB side must remain asserted until wack is observed; the actual SSRAM cycles occur in the PHY domain.
- Line state and vendor status from PHY are registered in the PHY domain and read by the RF/WB side; internal registers and handshake logic mitigate CDC hazards.

- Design constraints and expectations:
- No fixed frequency ratio or phase relationship is required between clk_i and phy_clk_pad_i; they may be fully asynchronous.
- WISHBONE master must hold stb/cyc (and address/data) stable until wb_ack_o is observed, which also ensures the PHY-domain sampling in usbf_wb is reliable.
- External SSRAM, if used, should be clocked/qualified to the PHY domain timing, since SSRAM control signals are generated with phy_clk_pad_i.
- Resume/suspend: resume_req_i is latched in clk_i domain and cleared by suspend_clr sourced from the PHY domain logic; internal synchronization is provided.

- Summary: Provide two free-running clocks (clk_i for system/WISHBONE and phy_clk_pad_i for UTMI/PHY) and an active-low reset rst_i (also driven to the PHY as phy_rst_pad_o). The design includes internal synchronizers/handshakes for CDC between these domains and expects standard WISHBONE hold-until-ack behavior.

-- Power Management Signals (usb_vbus, susp_o, resume_req_i) --

This core exposes three power■management related signals that coordinate USB attach/suspend/resume behavior across the system and the UTMI PHY domain:

- usb_vbus (input, phy_clk domain): Indicates VBUS/power condition to the UTMI link-state (LS) machine. When asserted, the LS FSM is held in POR (power-on-reset) state; when deasserted, the FSM proceeds through ATTACH and then NORMAL operation. In ATTACH, the core waits ~100 ms before declaring attached. Provide a stable, debounced level relative to phy_clk. Effect of assertion: clears attachment, exit-suspend flags, and reinitializes to FS defaults; deassertion allows normal attach/operation.

- susp_o (output, clk_i domain): System-level suspend status. Reflects the UTMI LS machine's usb_suspend flag, re-sampled into clk_i. Asserts after bus idle persists for ~3 ms (per UTMI suspend entry rules); deasserts on reset or a successful resume sequence. Use susp_o to gate clocks/power to the function logic. Note: single-flop CDC into clk_i; treat as a quasi-static status with up to one clk_i cycle latency.

- resume_req_i (input, clk_i domain): System request for remote wakeup/resume. Latched internally (resume_req_r) and synchronized into the phy_clk domain, then automatically cleared when the UTMI FSM issues suspend_clr after completing resume. Usage rules: only assert while suspended (susp_o=1) and after at least ~5 ms in suspend (UTMI spec constraint enforced by the FSM). When valid, the LS FSM performs a proper resume signaling sequence (drives K for ≥1 ms, then returns to NORMAL). Software/logic may pulse or level■hold resume_req_i; the core holds the request until it is honored and cleared.

Operational notes:
- Clock domains: usb_vbus is sampled in phy_clk; resume_req_i is accepted in clk_i and safely transferred to phy_clk; susp_o is a clk_i copy of a phy_clk status. Ensure long enough assertion of resume_req_i for recognition (the core latches it until cleared).
- Compliance timing: The FSM enforces approximate UTMI timings (idle ~3 ms to enter suspend; ≥5 ms before allowing remote wakeup; ≥1 ms resume K). No external timing shaping is required beyond meeting these conditions.
- Side effects: Asserting usb_vbus (POR) clears attachment and suspend state; deassertion begins attach timing. Resume requests are ignored unless in suspend and timing preconditions are met.

-- Vendor/Test Interfaces (VControl, VStatus) --

Overview
- The core exposes a simple vendor/test sideband interface intended to interact with PHY- or board-specific test logic: a 4■bit control bus (VControl[3:0]) with a write strobe (VControl_Load), and an 8■bit status input (VStatus[7:0]).
- These signals are surfaced at the top level as:
- VControl_pad_o[3:0] (output): vendor/test control value.
- VControl_Load_pad_o (output): pulse indicating a new control value is available (use as an external latch strobe).
- VStatus_pad_i[7:0] (input): vendor/test status returned by external logic.
- Functionally, they are connected to the register file (usbf_rf) as utmi_vend_ctrl[3:0] (output), utmi_vend_wr (output strobe), and utmi_vend_stat[7:0] (input).

Register map and access
- Address space: USB Function Register File (RF) space (WISHBONE accesses with wb_addr_i[12] == 0).
- Word address within RF: 0x05 (i.e., adr[6:0] == 0x05, which corresponds to byte offset 0x14 assuming 32■bit word addressing).
- Read (RF offset 0x05):
- Bits [7:0] = VStatus snapshot.

- Bits [31:8] = 0.
- Write (RF offset 0x05):
- Bits [3:0] written to VControl.
- Bits [31:4] are ignored (reserved).
- A write generates a pulse on VControl_Load to signal external logic to capture the new VControl value.

Clocking and timing behavior
- Two clock domains are involved:
- wclk (system/WISHBONE clock) for software register accesses.
- phy_clk (UTMI/PHY clock) for the vendor/test pins and internal USB logic.
- Write path (wclk -> phy_clk):
- Software writes RF[0x05] in wclk. The 4■bit value is transferred to phy_clk and presented on VControl.
- VControl_Load pulses in phy_clk as a one-shot notification that a new value is ready; external logic should latch VControl on this pulse.
- Read path (phy_clk -> wclk):
- VStatus is first registered in phy_clk, then sampled into wclk and returned on reads of RF[0x05].
- Because crossings are level/pulse-based and not fully handshaked, software should:
- Avoid back■to■back writes without allowing at least one phy_clk cycle for the load pulse to assert and be cleared.
- Expect VStatus reads to reflect a recent (but not strictly synchronized) snapshot; external logic should hold VStatus stable for at least one wclk cycle to be reliably captured.

Reset and default values
- VControl_Load is deasserted after reset.
- VControl default after reset is undefined until software performs the first write to RF[0x05]. External logic should not rely on a power■on VControl value.
- VStatus readback may be undefined until the first stable sample is captured after reset.

Functional notes and constraints
- The vendor/test interface is orthogonal to normal USB operation; it does not affect packet handling, endpoints, or protocol state machines.
- Reserved bits in the RF[0x05] register must be written as zero and read as zero.
- Typical usage:
1) Software writes a 4■bit control value to RF[0x05] to drive VControl and trigger VControl_Load.
2) External logic latches VControl on VControl_Load and may update VStatus.
3) Software reads RF[0x05] to obtain the current 8■bit VStatus.
- Common applications include PHY vendor■specific register access, board test hooks, or debug feature control via off■core logic.

-- Signal Timing and Latency --

This core has two primary clock domains and explicit handshakes that determine timing and latency from request to observable response. Unless USBF_ASYNC_RESET is defined, all resets are synchronous in their respective clock domains.

Clock domains
- phy_clk (UTMI/packet engine/memory master side): phy_clk_pad_i
- wb_clk (system/Wishbone/CSR side): clk_i (referred to as wclk in several blocks)
- All UTMI receive and transmit signals are registered on phy_clk. Most CSR/interrupt outputs are registered on wb_clk.

UTMI interface timing (usbf_utmi_if / usbf_utmi_ls)
- RX path: RxValid/RxActive/RxError/DataIn are sampled on phy_clk; rx_valid, rx_active, rx_err, rx_data update with 1 phy_clk cycle latency.
- TX path: DataOut updates on phy_clk when (TxReady || tx_first); TxValid is asserted as (tx_valid | drive_k | tx_valid_last | hold) and deasserts after TxReady or the end of a resume K, effectively stretching until the PHY accepts the last byte. Net latency from tx_valid to TxValid is 0–1 phy_clk cycle.
- Line-state/state-machine timing is derived from counters in phy_clk with programmable constants:
- T1 250 ns tick derived by a prescaler (USBF_T1_PS_250_NS=13). Higher-level thresholds: 2.5 µs, 3.0 ms, 3.125 ms, 5.0 ms.
- T2 multi-event timing: 100 µs, 2.5 µs granularity, 0.5 ms step, wakeup (~5 ms by count), 1.0 ms, 1.2 ms, and 100 ms maximum window.
- Suspend/resume and high-speed negotiation transitions occur when the above counters reach thresholds while line states satisfy required conditions (J/K/SE0 persistence checks use two-sample qualification). Outputs XcvSelect/TermSel/OpMode/mode_hs/usb_reset/usb_suspend/usb_attached change on the phy_clk edge that closes the state's condition; typical latency is 0–1 phy_clk cycles after the qualifying event.

Packet decoding/encoding (usbf_pd, usbf_pa)
- PID capture: The first PID byte is latched on rx_valid & rx_active. pid_* flags are valid 0–1 phy_clk cycles after PID sampling. Token fields (address, endpoint, CRC5) become valid after two RX bytes; token_valid asserts one cycle later.
- CRC16 (RX): Running CRC updates each cycle during rx_data_valid; crc16_err is valid at rx_data_done with 0–1 cycle latency.
- Data transmit (usbf_pa):
- send_data rising edge (from PE) starts transmission same phy_clk cycle; tx_valid asserts immediately.
- rd_next pulses when tx_ready & tx_valid_r to fetch next byte; one rd_next per payload byte.
- CRC insertion adds two trailing bytes. With send_zero_length, CRC phase begins immediately.
- send_token causes token PID to be sent in place of data; latency to DataOut is 0–1 phy_clk cycle.

Protocol engine timing (usbf_pe)
- Token/data PID selection and send_token/send_data are combinational from current state and inputs; registered outputs present on the next phy_clk.
- Buffer accounting (sizes, addresses) updates on phy_clk using current CSR and buffer descriptors.
- Timeouts:
- RX ACK timeout (rx_ack_to): counts phy_clk cycles when waiting for ACK; thresholds are 36 cycles (FS) or 22 cycles (HS). Absolute time depends on phy_clk frequency.
- TX DATA timeout (tx_data_to): counts phy_clk cycles while RX is inactive; thresholds 36 (FS) or 22 (HS).
- Abort conditions (e.g., buffer overflow, size violations, mismatched PID sequences) drive early state exits; state transitions occur on the next phy_clk with 0–1 cycle latency.

Internal DMA (usbf_idma)
- Handshake to memory arbiter: mreq asserted when ready; mack is sampled and registered as mack_r. Effective acknowledgement latency is 1 phy_clk cycle after mreq is asserted (mem_arb returns mack combinatorially but idma registers it).
- RX DMA (SSRAM writes):
- Starts after WAIT_MRD sees mack_r, then MEM_WR. Writes are word-packed; word write occurs when a 32-bit word is assembled or at end-of-packet via wr_last.
- idma_done asserts when rx_data_done or size counter reaches zero; latency 0–1 phy_clk cycles after those conditions.

- TX DMA (SSRAM reads):
- Issues alternating reads to fill two 32-bit staging buffers. After first mack_r, rd_first asserts; subsequent reads cycle MEM_RD2/MEM_RD3 when adrb_next[1:0]==3 and rd_next is seen.
- send_data remains asserted until size drains to zero; deasserts on the cycle the last byte is scheduled (or immediately for zero-length).

External SRAM arbitration (usbf_mem_arb)
- Single-port external interface multiplexed between PHY master (IDMA) and WB master. mack is tied to mreq; wack is generated one phy_clk later (wack_r) when no mreq is present.
- Port selection is combinational each cycle: Wishbone side wins only when (wreq | wack) and no mreq. Otherwise PHY side has priority.
- Effective PHY memory access latency: 1 phy_clk cycle from mreq to mack_r; external sram_re_o is held high; sram_we_o asserts in-cycle on write selection; sram_dout/adr reflect the selected master combinationally for the current cycle.

Register file and interrupts (usbf_rf/usbf_ep_rf)
- Endpoint CSR/data accesses are addressed on wb_clk; ep*_re/ep*_we decode from adr[6:2] and re/we with 0–1 wb_clk cycle latency.
- ep_match is generated on phy_clk by comparing ep_sel to CSR address; match propagates to rf outputs in 0–1 phy_clk cycles.
- DMA request generation (per-EP):
- dma_req is produced in wb_clk domain with internal hold logic when buffers are available or transfer remains. It asserts when dma_req_d is seen and clears on dma_ack and no-hold.
- dma_ack crosses to phy_clk through a two-flop synchronizer (r4/r5); dma_ack_i becomes valid after 2 phy_clk cycles typical.
- Interrupt lines (inta/intb):
- Event bits (int_stat) set on phy_clk from protocol/CRC/timeout conditions; reading INT register (re at adr==1) clears on the next clk edge.
- Output aggregation and mask application occur on wb_clk; inta/intb update the wb_clk following the bit set/clear, giving 1–2 clock cycle latency from event to pin depending on domain crossing.

Wishbone interface (usbf_wb)
- Address/data are used directly; wb_data_o returns rf_din or ma_din on the next wb_clk after read mux selection.
- Request detection: wb_req_s1 is sampled on phy_clk from (wb_stb_i & wb_cyc_i); response generation occurs via a state machine in phy_clk.
- Acknowledge timing:
- Register-file (RF) access: next_state=W0 causes wb_ack_d=1 for one phy_clk; wb_ack_o is a single-pulse on wb_clk after a two-flop pipeline (W1/W2), typically 2–3 wb_clk cycles after request sampling plus one phy_clk for cross-domain.
- Memory (MA) access: ma_req asserted until ma_ack; mem_arb returns wack one phy_clk after wreq (if PHY not using memory). wb_ack_o then follows the same two-stage wb_clk pipeline. Contention with PHY side can delay wack until mreq deasserts.
- Data path latency for reads: memory data returns to wb_data_o after arbiter selection and cross-domain; effective latency is at least 1 phy_clk for wack generation + 2 wb_clk for ack pulse formation; additional wait if PHY is occupying the SRAM.

General CDC and latency notes
- Many crossings use single-register sampling or two-flop synchronizers depending on signal direction. Where single-sample captures exist (e.g., wb_req_s1 sampled by phy_clk), design assumes requests are held stable long enough to avoid metastability hazards.
- All state machine outputs are registered; observable outputs (pins and WB ack) change on the clock

following the qualifying condition.
- Time-based behaviors (timeouts, suspend/resume, frame timing) scale with phy_clk. Absolute times depend on the actual PHY clock frequency (e.g., HS typically 60 MHz, FS typically 12 MHz). The provided constants define cycle thresholds; convert to time as threshold_cycles / phy_clk_frequency.

-- Clock Domain Crossing Requirements --

This core uses two primary, potentially asynchronous clock domains and includes multiple clock-domain crossings (CDCs) that must be handled carefully at integration time.

Clock domains
- phy_clk (UTMI domain): Driven by phy_clk_pad_i. All packet-level logic (PD/PA/PE/IDMA in usbf_pl), UTMI link state (usbf_utmi_if/usbf_utmi_ls), and the external buffer memory (via usbf_mem_arb) operate in this domain.
- wb_clk (system/Wishbone domain): Driven by clk_i. The Wishbone slave (usbf_wb) and register-file access side of the register block (usbf_rf) operate in this domain.

Reset
- rst is distributed to both domains. If USBF_ASYNC_RESET is not defined, rst must be synchronous to each domain. If USBF_ASYNC_RESET is defined, rst is used as an asynchronous assertion; deassert rst synchronously within each domain to avoid metastability.

CDC summary and requirements
1) Wishbone request/ack crossing (usbf_wb)
- Direction wb_clk -> phy_clk: wb_req_s1 <= wb_stb_i & wb_cyc_i is sampled directly in the phy_clk domain (single-flop sampling). Requirement: The Wishbone master must hold CYC/STB/ADR/DAT/WE stable until wb_ack_o is observed (classic WB handshake), ensuring the request level is stable long enough to be safely sampled.
- Direction phy_clk -> wb_clk: wb_ack_d (generated in phy_clk) is re-timed through 2+ flops in wb_clk and edge-detected to create wb_ack_o (1-cycle pulse). No additional action required by integrator beyond honoring WB protocol.
- STA: Declare clk_i and phy_clk_pad_i asynchronous (set_clock_groups -asynchronous) and constrain the req/ack crossings as CDC paths.

2) Buffer memory arbitration (usbf_mem_arb)
- All arbitration and SRAM control are clocked by phy_clk. The wclk port is not used internally, and wack is generated in phy_clk. The Wishbone side memory requests presented to usbf_mem_arb also originate from phy_clk (via usbf_wb's MA interface), so there is no CDC within the memory path itself.
- External SSRAM interface: Treat as synchronous to phy_clk. Constrain sram_* signals to phy_clk timing.

3) Register file and endpoint register block (usbf_rf and usbf_ep_rf)
- Write/read strobes and address: rf_we/rf_re/adr originate in the phy_clk FSM of usbf_wb and are consumed in the wb_clk (wclk) domain of usbf_rf. These pulses/levels cross clock domains without explicit synchronizers. Requirement: Either (a) use a common clock (tie clk_i == phy_clk_pad_i), or (b) add proper CDC synchronization for re/we/adr (e.g., re-time strobes with two-flop synchronizers and hold address/data stable for multiple wclk cycles), or (c) wrap usbf_rf with a register-slice CDC bridge.
- Status/interrupts back to wb_clk: Many status bits (e.g., int_stat, endpoint CSR mirrors) are produced in phy_clk and are sampled in wb_clk logic without explicit multi-bit synchronization. Requirement: If clocks are asynchronous, add synchronizers for single-bit status (2FF) and use handshake/gray-code schemes or shadow registers to move multi-bit values. Alternatively, keep both domains on the same clock.

4) UTMI vendor/status and line-state into register file
- LineState and VStatus are captured in the phy_clk domain at top-level, then sampled in wb_clk within usbf_rf. This is a CDC without explicit synchronizers. Requirement: If clocks are asynchronous, re-time these buses into wb_clk using appropriate CDC techniques (2FF for single-bit, bus synchronizers/handshake for multi-bit) or use a shared clock.

5) Resume/suspend handshake across domains
- resume_req (clk_i -> phy_clk): Top-level holds resume_req_r asserted in wb_clk until cleared by suspend_clr generated in phy_clk, then re-sampled back into wb_clk. This level-based handshake mitigates short pulses, but sampling is single-flop. Requirement: Keep resume_req_i asserted until suspend_clr is observed; if clocks are asynchronous, consider adding 2FF synchronizers on both control and acknowledge paths.

General integration guidance
- Preferred: Use a single clock for clk_i and phy_clk_pad_i to avoid CDC hazards, as several paths lack full CDC hardening for multi-bit signals.
- If separate clocks are required:
- Classify clk_i and phy_clk_pad_i as asynchronous in STA and set false-paths or asynchronous clock groups between them.
- Add 2FF synchronizers for single-bit control/status that cross domains (e.g., re/we, interrupt sources, suspend_clr, resume_req).
- For multi-bit buses (addresses, data, CSR images, status vectors), use stable-level handshakes or bus synchronizers; ensure the producer holds values stable over at least two destination clock cycles.
- Ensure WISHBONE master adheres strictly to classic handshake timing (hold CYC/STB/ADR/DAT/WE until ACK) so usbf_wb can safely capture requests.
- Reset deassertion should be glitch-free and synchronized per domain if asynchronous assertion is used.

PARAMETERIZATION AND CONFIGURATION

-- Generic Parameters (SSRAM_HADR, Address Widths) --

- Purpose
- This core exposes two key compile-time sizing knobs that shape bus widths and buffer memory depth:
- SSRAM_HADR: selects the on-chip buffer memory depth (in 32-bit words) and all related address widths to the SSRAM/datapath.
- USBF_UFC_HADR: selects the width of the WISHBONE address bus used by the USB Function Controller (UFC) side and the bit used to split Register-File (RF) vs Buffer-Memory (MEM) accesses.

- Defaults in this file (after preprocessor redefinitions; last definition wins)
- `USBF_SSRAM_HADR = 9
- `USBF_UFC_HADR = 12
- RF/MEM select: `USBF_RF_SEL = !wb_addr_i[12], `USBF_MEM_SEL = wb_addr_i[12]

- SSRAM_HADR (buffer memory depth and address widths)
- Scope/propagation: Parameter SSRAM_HADR is defined in usbf_top and passed down into usbf_pl, usbf_idma, usbf_pe, and usbf_mem_arb.

- External SSRAM address width: sram_adr_o is [SSRAM_HADR:0] (word addressing of 32-bit words).
- Internal DMA address width: IDMA/PE use [SSRAM_HADR+2:0] to include the 2 LSB byte/word-offset bits and span two-word prefetching.
- Wishbone→memory mapping: the memory word address seen from Wishbone is ma_adr[SSRAM_HADR+2:2] (i.e., drop byte lanes [1:0], keep word index up to bit SSRAM_HADR+2).
- Capacity formula: Buffer Memory Size = (2^(SSRAM_HADR+1)) words × 4 bytes/word. Example with default SSRAM_HADR=9 → 2^(10) × 4 = 4096 bytes.
- Data width: 32-bit datapath to SSRAM (sram_data_[i|o] are 32 bits). Addressing is word-based; byte-lane selection is handled internally.

- WISHBONE/UFC address width and mapping (USBF_UFC_HADR)
- WISHBONE address bus width: wb_addr_i is [USBF_UFC_HADR:0]. With default 12 → 13-bit address (A12..A0).
- RF/MEM address-space split: A[12] (by default) selects the target space.
- A[12]=0 → Register File (RF) space (\`USBF_RF_SEL)
- A[12]=1 → Buffer Memory (MEM) space (\`USBF_MEM_SEL)
- Register file addressing into usbf_rf:
- usbf_top passes adr = ma_adr[8:2] (7 bits) to usbf_rf; A[1:0] are the byte lanes of the 32-bit WISHBONE data bus.
- Effective RF map: 128 longword slots (7-bit index), organized in sub-windows per endpoint as implemented inside usbf_rf.
- Memory addressing into SSRAM from WISHBONE:
- Word address is ma_adr[SSRAM_HADR+2:2], producing SSRAM_HADR+1 bits to drive sram_adr_o[SSRAM_HADR:0].

- Consistency constraints and guidance when changing sizes
- Relationship: USBF_UFC_HADR must be large enough to cover the memory word address field up to bit (SSRAM_HADR+2) and still have a higher bit available for RF/MEM selection. With the provided split, a good rule is:
- USBF_UFC_HADR ≥ (SSRAM_HADR + 3)
- Example: SSRAM_HADR=9 → (9+3)=12 (matches default USBF_UFC_HADR=12).
- If you change USBF_UFC_HADR, also ensure `USBF_RF_SEL/`USBF_MEM_SEL use the same split bit. In this file they are defined explicitly on bit 12; adjust them together when moving the split.
- If you change SSRAM_HADR, the following adjust automatically via parameter propagation: sram_adr_o width, internal DMA address widths, and the WISHBONE-to-memory slice ma_adr[SSRAM_HADR+2:2]. Ensure USBF_UFC_HADR and the split bit still satisfy the constraint above.

- Practical examples (with defaults)
- SSRAM_HADR=9 → sram_adr_o[9:0] (1024 words), 4 KB buffer memory.
- USBF_UFC_HADR=12 → WISHBONE A[12:0]; A[12] selects MEM vs RF; RF address passed as A[8:2] to usbf_rf; MEM word address passed as A[11:2] to SSRAM.

- How to override
- Via preprocessor: compile with -DUSBF_SSRAM_HADR= and/or -DUSBF_UFC_HADR= and adjust `USBF_RF_SEL/`USBF_MEM_SEL accordingly.
- Via parameter override (where applicable): set SSRAM_HADR on the usbf_top instance; it propagates to all submodules that parameterize on SSRAM_HADR.

-- Endpoint Enablement Macros (USBF_HAVE_EPn) --

Compile-time macros USBF_HAVE_EPn (where n = 1..15) control which non-control USB endpoints

are instantiated in hardware. Endpoint 0 is always present and is not gated by any macro.

Behavior
- If USBF_HAVE_EPn is defined at compile time, the real endpoint register file usbf_ep_rf is instantiated for EPn.
- If USBF_HAVE_EPn is not defined, a stub usbf_ep_rf_dummy is instantiated for EPn, which ties off functionality: no match, no interrupts, no DMA requests, CSR reads return 0, and buffers appear unallocated (buf0/buf1 = 0xFFFF_FFFF).
- The macros are tested with `ifdef, so only the presence of the macro matters (its value is ignored). Typical use is +define+USBF_HAVE_EPn or `define USBF_HAVE_EPn.

Contiguity requirement
- Endpoints must be enabled contiguously from EP1 upwards. Enabling EP(k) while any lower EP(m
Effects
- Register map: The address decode window for all endpoints (EP0..EP15) remains. Accesses to a disabled endpoint’s window operate on the dummy instance and read back zeros (or the dummy defaults) and produce no side effects.
- DMA: dma_req[n] is driven by the endpoint instance. Disabled endpoints keep their DMA request low; dma_ack[n] is still a port but unused by the dummy.
- Interrupts: inta/intb contributions from disabled endpoints are 0.
- Matching and endpoint selection: Disabled endpoints never match (ep_match = 0) and therefore do not participate in transfers.
- Resource usage: Only enabled endpoints synthesize real logic; disabled endpoints reduce area.

Limits and defaults
- Supported endpoint indices: 0..15 (EP0 fixed on; macros cover EP1..EP15).
- Example/default in this file: USBF_HAVE_EP1, USBF_HAVE_EP2, USBF_HAVE_EP3 are defined; EP4..EP15 are not defined, yielding endpoints 0..3 enabled.

Integration guidance
- Define USBF_HAVE_EP1..USBF_HAVE_EP(K) for the highest endpoint number you require; do not skip indices (e.g., enable EP1..EP6 to get endpoints 0..6).
- Add the defines in your synthesis/simulation toolchain (e.g., Verilog +define+USBF_HAVE_EP4) or via a header.
- Ensure your USB descriptors’ endpoint count matches the enabled hardware and that SRAM/buffer sizing is adequate for the chosen number of endpoints.

-- Timing and Timeout Parameters (T1/T2, ACK/DATA TO) --

Timing and Timeout Parameters govern link-state (T1/T2) timing in the UTMI low-speed controller and handshake/data timeouts in the packet engine. All timers run in the PHY clock domain (phy_clk_pad_i) and are parameterized by macros that can be tuned at synthesis.

1) T1 timing (idle-based, us-ms scale)
- Purpose: Detect reset and suspend entry/exit conditions from continuous line-state idleness.
- Idle definition: ls_idle = (mode_hs ? SE0 : J). An "idle_long" qualifier asserts when two consecutive samples indicate idle.
- Tick generation: A prescaler (ps_cnt) produces a periodic tick (ps_cnt_clr) after USBF_T1_PS_250_NS cycles. On each tick, an 8-bit counter (idle_cnt1) increments while idle_long is true.
- Threshold macros (expressed as counts of the ps_cnt_clr tick):
• USBF_T1_C_2_5_US
• USBF_T1_C_3_0_MS

• USBF_T1_C_3_125_MS
• USBF_T1_C_5_MS
• USBF_T1_C_62_5_US (auxiliary windowing constant)
- Derived flags:
• T1_gt_2_5_uS: idle_cnt1 > USBF_T1_C_2_5_US
• T1_st_3_0_mS: idle_cnt1 < USBF_T1_C_3_0_MS
• T1_gt_3_0_mS: idle_cnt1 > USBF_T1_C_3_0_MS
• T1_gt_3_125_mS: idle_cnt1 > USBF_T1_C_3_125_MS
• T1_gt_5_0_mS: idle_cnt1 > USBF_T1_C_5_MS
- Usage in UTMI LS state machine (usbf_utmi_ls):
• NORMAL: !mode_hs & T1_gt_2_5_uS & T1_st_3_0_mS & !idle_long → RESET
• NORMAL: !mode_hs & T1_gt_3_0_mS → SUSPEND
• NORMAL: mode_hs & T1_gt_3_0_mS → RES_SUSP
• SUSPEND: T1_gt_2_5_uS & se0_long → RESET; k_long → RESUME; T1_gt_5_0_mS & resume_req → RESUME_REQUEST
- Notes: Effective time in seconds depends on phy_clk and the prescaler constant USBF_T1_PS_250_NS; macro names indicate intended nominal durations.

2) T2 timing (microframe/millisecond scale)
- Purpose: Provide coarser time bases for reset, resume, attach, and speed negotiation windows.
- Two-stage counters:
• me_ps counts phy_clk cycles; me_ps_2_5_us asserts at USBF_T2_C_2_5_US cycles (~2.5 µs nominal).
• me_ps2 counts me_ps_2_5_us events; me_ps2_0_5_ms asserts at USBF_T2_C_0_5_MS events (~0.5 ms nominal).
• me_cnt increments each me_ps2_0_5_ms pulse and is compared to:
– USBF_T2_C_100_US (via me_ps2) → T2_gt_100_uS
– USBF_T2_C_WAKEUP → T2_wakeup (resume signaling min)
– USBF_T2_C_1_0_MS → T2_gt_1_0_mS
– USBF_T2_C_1_2_MS → T2_gt_1_2_mS
– USBF_T2_C_100_MS → me_cnt_100_ms (attach delay)
- Usage in UTMI LS state machine:
• RESUME/RESUME_WAIT/RESUME_REQUEST windows driven by T2_gt_100_uS, T2_gt_1_0_mS, T2_wakeup
• RESET entry and speed negotiation staging uses T2_gt_1_0_mS, T2_gt_1_2_mS
• ATTACH delay to NORMAL occurs at me_cnt_100_ms (~100 ms)
- Notes: Actual times scale with phy_clk and macro values; defaults target nominal USB timings.

3) Handshake/Data timeouts in Packet Engine (ACK/DATA TO)
- Domain: phy_clk (same as link logic).
- Mode selection: mode_hs selects HS- vs FS-specific thresholds.
- RX ACK timeout (after IN data):
• Counter: rx_ack_to_cnt increments while rx_ack_to_clr is deasserted; cleared by tx_valid activity.
• Threshold: rx_ack_to_val = (mode_hs ? USBF_RX_ACK_TO_VAL_HS : USBF_RX_ACK_TO_VAL_FS).
• Use: In state IN2, if token_valid & pid_ACK arrives before timeout → proceed to UPDATE; if rx_ack_to asserts → return to IDLE and set timeout interrupt.
- TX DATA timeout (before OUT data arrival):
• Counter: tx_data_to_cnt increments while rx_active is low; cleared while rx_active is high.
• Threshold: tx_data_to_val = (mode_hs ? USBF_TX_DATA_TO_VAL_HS : USBF_TX_DATA_TO_VAL_FS).
• Use: In state OUT, if tx_data_to asserts (no data started) → abort to IDLE and set timeout interrupt.

- Interrupt generation: int_to_set = (state==IN2 & rx_ack_to) | (state==OUT & tx_data_to). This feeds endpoint interrupt status machinery (usbf_ep_rf/usbf_rf).

4) Configurability and integration
- All thresholds are macros defined at compile time; system integrators may retune for specific PHY clock frequencies.
- Speed-dependent behavior is gated by mode_hs (set by the UTMI LS controller during speed negotiation).
- Timers automatically clear on state machine events (e.g., tx activity for ACK TO, rx_active for DATA TO, and explicit clears within utmi_ls transitions) to ensure clean windowing.

Default macro values (for reference) include: USBF_RX_ACK_TO_VAL_FS=36, USBF_RX_ACK_TO_VAL_HS=22, USBF_TX_DATA_TO_VAL_FS=36, USBF_TX_DATA_TO_VAL_HS=22, USBF_T1_PS_250_NS=13, USBF_T1_C_2_5_US=10, USBF_T1_C_62_5_US=250, USBF_T1_C_3_0_MS=48, USBF_T1_C_3_125_MS=50, USBF_T1_C_5_MS=80, USBF_T2_C_2_5_US=148, USBF_T2_C_0_5_MS=200, USBF_T2_C_WAKEUP=10, USBF_T2_C_100_US=40, USBF_T2_C_1_0_MS=2, USBF_T2_C_1_2_MS=2, USBF_T2_C_100_MS=200. Effective real times depend on phy_clk and these prescale factors.

-- Debug and Test Options (USBF_DEBUG, USBF_TEST_IMPL) --

Debug and Test Options control simulation-only diagnostics and (potentially) test hooks. USBF_DEBUG: When defined, compiles in $display-based runtime checks primarily in usbf_pe and usbf_idma. These checks print clear error messages if key control signals are X/unknown or violate expected conditions at specific FSM states (e.g., IDLE/IN/OUT/UPDATE in usbf_pe; IDLE/WAIT_MRD/MEM_WR/... in usbf_idma). This aids in catching X-propagation, illegal handshakes, and timing/sequence issues during simulation. It does not alter functional logic or synthesize into hardware (messages are simulation-only). For additional step-by-step state tracing, an optional USBF_VERBOSE_DEBUG (not defined by default) prints state-entry messages. Recommended: enable for simulation bring-up and X-hunting; disable for quieter regressions or synthesis. USBF_TEST_IMPL: Present but not used in the provided modules; it has no effect in this file set. It is reserved for integrating test-specific implementations elsewhere. Enabling it here does not change behavior.

-- Buffer Memory Sizing and Addressing --

Overview: The USB function core uses a shared, 32■bit wide SSRAM for all endpoint buffers. The physical SSRAM address presented on sram_adr_o is word-addressed (32■bit words). The memory depth and all pointer/length fields are parameterized by SSRAM_HADR. Total buffer memory size is: bytes = 4 × 2^(SSRAM_HADR+1). Example: SSRAM_HADR=14 -> 128 KiB. Addressing domains: 1) External SSRAM port (to SRAM): word addressing [SSRAM_HADR:0], 32■bit data. 2) Endpoint buffer descriptors (buf0/buf1): byte addressing [SSRAM_HADR+2:0] embedded in a 32■bit register, allowing unaligned starts and automatic byte/word boundary handling. 3) Internal DMA: converts byte addresses to word addresses and manages sub■word byte indices. Buffer descriptor format (per endpoint, buf0/buf1 in usbf_ep_rf): - Bit 31: status flag (used by core as not-allocated/done). - Bits [30:17] (14 bits): buffer length in bytes (current/remaining). - Bits [SSRAM_HADR+2:0]: buffer start pointer in bytes (lower 2 bits are byte offset within a word). Core-generated updates write the full 32■bit descriptor atomically. Address and size handling in transfers: - Max payload size is 11 bits (csr[10:0]); the packet size used per transaction is size_next = min(buf_size, max_pl_sz). - New buffer pointer = old pointer + size_next (IN) or +max_pl_sz (OUT with DMA), computed in byte space; the lower 2 bits are preserved/updated to maintain correct sub■word alignment. - Remaining length is decremented

accordingly; when the buffer becomes empty (IN) or cannot fit another packet (OUT), buffer_done is asserted. - For DMA endpoints, when buffer_done is reached, the core pulses buf*_rl to release the buffer instead of writing back a new pointer/length. Byte/word conversion (IDMA): - The IDMA performs packing/unpacking between byte streams and 32■bit SSRAM words. - It tracks a 3■bit byte index (adr_cb) to assemble 4 bytes per write and to walk bytes across words on read; word reads are pre-fetched into two 32■bit staging registers to allow seamless byte sequencing across word boundaries. - The SSRAM word address used on the memory port is adr[SSRAM_HADR+2:2]; the byte index uses adr[1:0]. Wrap/linear behavior: - For DMA operation, the internal word address generator advances sequentially; when the computed next word address reaches start+length (buffer end), it wraps to 0 (enables safe progression even if a buffer spans to the end of memory). Non-DMA updates simply write back the new byte pointer to the descriptor, so no implicit wrap is required. Wishbone access to buffer memory: - The Wishbone master address (ma_adr) is byte-based; the SRAM word address seen by the arbiter is ma_adr[SSRAM_HADR+2:2]. CPU accesses are 32■bit word operations (no byte strobes), matching the IDMA's packing. Constraints and recommendations: - Choose SSRAM_HADR large enough so that buf_size (14■bit) and endpoint payloads (11■bit) comfortably fit; memory is shared across all endpoints. - Buffer pointers may be byte■unaligned; the core handles alignment internally. - For deterministic behavior, software should mark a buffer not-allocated by setting the descriptor to all 1s or by using the status bit scheme consistently before enabling an endpoint.

-- Endpoint CSR Configuration (Direction, Type, Max Packet, DMA Enable) --

Endpoint CSR (EPn, offset 0 within the endpoint's 4-word register bank) packs direction, transfer type, max■packet size, and DMA enable as follows (bit indices are [31:0], MSB..LSB):

- Direction (bits [27:26])
- 00: Control endpoint (CTRL)
- 01: IN (device → host)
- 10: OUT (host → device)
- 11: Reserved
- Notes: Internally decoded as CTRL_ep, IN_ep, OUT_ep.

- Type (bits [25:24])
- 01: Isochronous
- 10: Bulk
- 00/11: Treated as non■iso/non■bulk by the core (typical mapping is 00=Control, 11=Interrupt). The core only checks these for iso/bulk behaviors.
- Effects: For Isochronous, the protocol engine skips ACK/handshake waits (e.g., IN transitions directly to UPDATE; OUT goes via UPDATEW). For non■iso, normal handshakes apply.

- DMA Enable (bit [15])
- 0: DMA disabled (PIO/buffered mode)
- 1: DMA enabled
- Effects/constraints:
- Ignored for Control endpoints (Direction=00); the core forces DMA off for CTRL endpoints.
- When enabled on IN/OUT endpoints, the IDMA engine moves data to/from external SRAM; buffer selection logic changes (endpoint logic fixes selection to buffer 0 for DMA transfers) and flow control uses dma_in_buf_sz1 (IN) and dma_out_buf_avail (OUT). If insufficient data/space is available, the core NACKs/NYETs as appropriate (NYET in HS mode when no space).

- Max Packet Size (bits [10:0])
- Size in bytes (11-bit field).
- Used by the protocol engine for transfer sizing, buffer pointer advancement, and validation against small/large constraints.

- Special case: For IN transfers, a value of 0 forces a zero■length packet (send_zero_length asserted).

Access notes:
- The CSR is 32-bit and written via the EPn register window (adr[1:0]=2’b00 inside each endpoint bank). Reset value is 0 (Direction=Control, Type=00, MaxPkt=0, DMA=0).

-- Default Reset Configuration --

- Reset polarity and timing
- rst_i is active-low. All internal resets are synchronous to their respective clocks unless USBF_ASYNC_RESET is defined (not defined here).

- Global/top-level outputs after reset release (rst_i=1)
- Interrupts: inta_o=0, intb_o=0
- DMA: dma_req_o[15:0]=0
- Suspend: susp_o=0 (usb_suspend cleared)
- UTMI pads:
- XcvSelect_pad_o=1 (Full-Speed transceiver selected)
- TermSel_pad_o=1 (FS termination enabled)
- SuspendM_pad_o=0 unless LineState_pad_i==2'b10 (K); default deasserted if line not K
- OpMode_pad_o: not forced during POR; remains unchanged until later state transitions (normal mode will be set during resume/speed-neg)
- TxValid_pad_o=0; tx_ready mirrors TxReady_pad_i; DataOut_pad_o undefined until a transmit or drive_k occurs
- SRAM interface: sram_re_o=1, sram_we_o=0; sram_adr_o/sram_data_o are don't-care until a request
- Vendor control: VControl_Load_pad_o=0, VControl_pad_o=4'b0000
- phy_rst_pad_o mirrors rst_i

- USB link/PHY controller defaults (usbf_utmi_if/usbf_utmi_ls)
- Mode: mode_hs=0 (FS), usb_reset=0, usb_suspend=0, usb_attached=0
- Line-state driven K signaling (drive_k) inactive
- State machine starts in POR, forcing FS select and FS termination; immediately proceeds to ATTACH state timing

- Packet layer defaults (usbf_pl, usbf_pd, usbf_pa, usbf_idma, usbf_pe)
- All internal state machines in IDLE
- No token/data transmit in progress: tx_valid=0, tx_first=0
- Receive datapath idle: rx_data_valid=0, rx_data_done=0, crc16_err=0
- No DMA activity: rx_dma_en=0, tx_dma_en=0, idma_done=0
- Timing/counters cleared (frame tracking and half-microsecond clocks start from 0)

- Register file and endpoint defaults (usbf_rf, usbf_ep_rf)
- Function address: funct_adr=7'h00
- Interrupt masks: inta_msk=intb_msk=0; all RF and EP interrupt sources cleared
- Resume request: rf_resume_req=0
- UTMI vendor iface regs: utmi_vend_ctrl=4'h0, utmi_vend_wr=0; status is a sampled copy of VStatus_pad_i
- Endpoint configuration (for all instantiated endpoints; non-present endpoints are tied-off dummies with the same visible reset state):
- CSR fields: csr0=0, csr1=0, uc_bsel=0, uc_dpd=0, ots_stop=0 → direction=00 (control), transfer type=00, not disabled, not stalled, DMA disabled (csr[15]=0), max packet size=0, size-match qualifiers

cleared
- Buffers: buf0=32'hFFFF_FFFF, buf1=32'hFFFF_FFFF (unallocated); buf0_orig=32'hFFFF_FFFF
- Interrupt enables: iena=0, ienb=0; interrupt status int_stat=0; inta=intb=0
- DMA sideband: dma_req=0; dma_in_buf_sz1=0; dma_out_buf_avail=0; all internal DMA counters cleared

- Wishbone interface defaults (usbf_wb)
- State machine in IDLE; no memory/rf access pending
- wb_ack_o=0; wb_data_o reflects selected target on reads when issued; rf_we=0; ma_req=0, ma_we=0

- Memory arbiter defaults (usbf_mem_arb)
- wack_r=0; mack follows mreq; sram_we_o=0; sram_re_o=1 constant; SRAM address/data mux idle until a request

- Miscellaneous/status
- Addressing parameters in this build: USBF_UFC_HADR=12 (13-bit WB address), SSRAM_HADR=9 (10-bit word address)
- Error flags: pid_cs_err=0, nse_err=0, crc5_err=0 until traffic is observed
- frm_nat counters start at 0

-- Synthesis and Clocking Assumptions --

- Clock domains
- phy_clk_pad_i: UTMI/PHY clock domain. Drives the UTMI interface (usbf_utmi_if/usbf_utmi_ls), packet layer (usbf_pl, usbf_pd, usbf_pa, usbf_idma, usbf_pe), the PHY-side of the endpoint/register file (usbf_rf clk port), and the external buffer memory (usbf_mem_arb and sram_* signals). Expected rate is UTMI 60 MHz; all UTMI/suspend/reset timing counters are derived from this clock.
- clk_i: System/WISHBONE clock domain. Drives the WISHBONE interface (usbf_wb), the WB-side of the register file (usbf_rf wclk port), and the WB side of the memory arbiter.
- The two clocks are fully asynchronous to each other; there is no defined phase or frequency relationship.

- Reset
- rst_i is active-high and distributed to both clock domains.
- By default resets are synchronous to their respective clocks. If macro USBF_ASYNC_RESET is defined, many always blocks use asynchronous assertion (negedge rst) with synchronous deassert.
- Hold reset long enough for all CDC handshakes to settle across both domains.

- Clock-domain crossing (CDC)
- CDC between clk_i and phy_clk_pad_i is done with simple registered handshakes and two-flop style request/ack schemes (e.g., usbf_wb, usbf_mem_arb, usbf_ep_rf DMA req/ack staging).
- Multi-bit data buses cross domains only when gated by these handshakes; otherwise they are used within a single domain.
- STA constraints: treat clk_i and phy_clk_pad_i as asynchronous clock groups (e.g., set_clock_groups -asynchronous). Do not rely on timing between these domains; optionally add false-paths or CDC exceptions on handshake paths.

- External buffer memory (SRAM/BRAM) interface
- Single-port, 32-bit data, address width is (SSRAM_HADR+1) bits; macro USBF_SSRAM_HADR defines depth (final effective value in this file is 9 $\rightarrow$ 4 KiB total buffer: 2^(9+1) words × 4 bytes).
- Memory is clocked by phy_clk_pad_i only; all sram_* signals are synchronous to phy_clk_pad_i.
- Zero-wait-state assumption: usbf_mem_arb asserts mack combinationally with mreq, and sram_re_o

is tied high. The memory must present read data with a single-cycle latency compatible with this assumption (or be a wrapper that emulates flow-through read and immediate acknowledge).
- Read-during-write and continuous-read behavior must be supported by the memory or wrapper, as sram_re_o is constantly asserted and read data is sampled whenever requested.

- WISHBONE access and address mapping
- WISHBONE runs on clk_i; ack generation is multi-cycle due to CDC. Masters must hold CYC/STB stable until wb_ack_o pulses.
- Address decode uses macros: USBF_RF_SEL = !wb_addr_i[12] (register file), USBF_MEM_SEL = wb_addr_i[12] (buffer memory). Macro USBF_UFC_HADR final value is 12 in this file.

- UTMI interface and timing constants
- UTMI 8-bit interface is assumed at 60 MHz. Timing counters in usbf_utmi_ls are derived from macros (e.g., USBF_T1_PS_250_NS, USBF_T1_C_xxx, USBF_T2_C_xxx). These constants must match the actual phy_clk_pad_i frequency; if the PHY clock differs, adjust the macros accordingly to meet USB spec timing (reset, resume, suspend, chirp, etc.).

- Synthesis notes
- No gated clocks; clock enables and state machines are used throughout.
- Simulation-only code: $display and initial blocks exist (e.g., in usbf_top and under USBF_DEBUG). For clean synthesis, compile without USBF_DEBUG/USBF_TEST_IMPL or ensure your tool ignores system tasks.
- No vendor-specific primitives are instantiated; design is generic RTL.

- I/O and miscellaneous assumptions
- UTMI signals (XcvSelect, TermSel, SuspendM, OpMode, LineState) must connect to a UTMI/UTMI+ compliant PHY. LineState and DataIn/Out are synchronous to phy_clk_pad_i as per UTMI.
- resume_req_i originates in clk_i domain and is synchronized inside the UTMI block before use.
- No tri-stated internal busses; external memory data_in/out are separated.
- Byte ordering on the 32-bit buffer follows little-endian byte lanes as mapped in usbf_idma (adr[1:0] selects byte within word).

MODULE HIERARCHY AND INTEGRATION NOTES

-- Module List and Roles --

- usbf_top: SoC-facing USB function controller top level. Integrates UTMI PHY interface, protocol layer, register file (RF), WISHBONE slave, and SSRAM memory arbiter. Exposes pads (UTMI), WISHBONE bus, interrupt lines, DMA request vector, and external SRAM interface.
- usbf_utmi_if: UTMI PHY front-end bridge. Synchronizes UTMI Rx/Tx handshakes and data to internal signals, manages TxValid generation (incl. resume K drive), and instantiates the UTMI line-state controller.
- usbf_utmi_ls: UTMI line-state and link manager. Detects/reset/suspend/attach states, handles HS/FS speed negotiation (chirp K/J), resume signaling, line termination and transceiver select control, and produces mode_hs/usb_reset/usb_suspend/usb_attached.
- usbf_pl: USB protocol layer integration. Connects packet decoder (PD), packet assembler (PA), internal DMA (IDMA), and protocol engine (PE). Provides address match against function address, exposes frame/sof timing (frm_nat), and bridges payload accesses to external memory via

madr/mdout/mreq.
- usbf_pd: Packet decoder. Parses incoming UTMI RX stream into PIDs, tokens (function address, endpoint), frame numbers, data payload with CRC16 check, and flags sequence/CRC5/CRC16 errors.
- usbf_pa: Packet assembler. Builds outgoing token/data packets, selects proper DATA PID, appends CRC16, manages send_first/last timing toward UTMI Tx.
- usbf_pe: Protocol engine per active endpoint context. Implements USB token/data handshake/state machines for IN/OUT/SETUP/ISO/BULK/CTRL, PID sequencing and checks, timeout handling (ACK/DATA TO), buffer selection/rollover, zero-length handling, and generates RF updates/interrupt set strobes. Coordinates with IDMA for payload moves.
- usbf_idma: Internal DMA engine. Moves payload data between UTMI side and external SSRAM: writes RX data to memory and reads TX data from memory. Generates memory addresses/byte lanes, tracks sizes, and signals send_data/idma_done.
- usbf_crc16: Byte-wise USB CRC16 generator used by PA/PD.
- usbf_crc5: USB CRC5 generator used for token verification.
- usbf_rf: Register file and endpoint aggregator. Hosts top-level RF registers (main status, function address, interrupt masks/sources, frame/diagnostics, UTMI vendor regs) and instantiates up to 16 endpoint RF blocks (ep0..ep15). Aggregates per-endpoint CSRs/buffers/DMA status, produces combined interrupt outputs (inta/intb), endpoint match muxing for PE, and issues per-endpoint DMA request bits.
- usbf_ep_rf: Endpoint register file (real endpoint). Holds endpoint CSR (type/enable/stall/max packet, data toggle, buffer select), buffer descriptors (two buffers with address/size/flags), per-endpoint interrupt enables/status, and DMA request/handshake logic for IN/OUT directions.
- usbf_ep_rf_dummy: Stub endpoint RF used when an endpoint is disabled via compile-time macros; returns safe defaults and no DMA/interrupts.
- usbf_wb: WISHBONE slave interface. Decodes address space into RF vs external memory windows, returns read data from RF or memory, sequences memory access requests toward the arbiter, and generates WISHBONE acknowledgments.
- usbf_mem_arb: External SSRAM arbiter. Time-multiplexes memory interface between protocol-side master (payload DMA) and WISHBONE-side master (CPU access), arbitrating address/data/we and returning read data/ack.
Notes:
- Endpoint instantiation is compile-time selectable via USBF_HAVE_EPx macros (EP0 always present). usbf_rf multiplexes the selected endpoint's CSR/BD and DMA status toward PE.
- External memory is addressed as 32-bit words with parameterizable high address width (SSRAM_HADR). The arbiter bridges protocol (phy_clk) and bus (wclk) domains.
- Interrupts are generated per-endpoint and at RF level (attach/detach/reset/suspend/errors) and presented as inta/intb outputs.
- DMA requests are per-endpoint (dma_req_o[15:0]) with matching dma_ack_i inputs.

-- Clock Domain Partitioning and Crossings --

- Clocks and resets
- Two primary clock domains:
- PHY domain: phy_clk_pad_i (UTMI/packet path, DMA, memory arbiter control, Wishbone transfer FSM)
- WB domain: clk_i (WISHBONE host/programming, interrupt aggregation)
- Reset rst_i is fed to both domains. If USBF_ASYNC_RESET is undefined (as in this file), all resets are synchronous to their local clock. If defined, resets become async (negedge rst) with no additional re-synchronization across domains.

- Domain ownership (top-level instances)
- PHY domain (phy_clk_pad_i): usbf_utmi_if (incl. usbf_utmi_ls), usbf_pl (usbf_pd/usbf_pa/usbf_idma/usbf_pe), usbf_mem_arb sequential logic, usbf_wb main FSM and

RF/Memory selection, usbf_rf "clk" side (per-EP datapath and DMA counters)
- WB domain (clk_i): usbf_rf "wclk" side (host-visible CSRs: funct_adr, masks, vendor ctrl, readback mux; INT aggregation), top-level resume_req_r/susp_o handling

- Key clock domain crossings (CDC)
1) WB -> PHY (programming path via usbf_wb)
- wb_stb_i/wb_cyc_i/wb_we_i sampled into PHY domain (wb_req_s1) without explicit synchronizers; wb_addr_i/wb_data_i used combinationally in PHY FSM
- Return path: wb_ack_d generated in PHY domain, then single-flop pipelined in WB domain (wb_ack_s1/2) to produce wb_ack_o
- Read data: rf_din/ma_din (PHY) used to drive wb_data_o in WB domain; validity tied to ack pipeline rather than CDC synchronizers
2) PHY -> WB (status/errors into RF)
- usb_attached and suspend are double-synchronized in WB domain (attach_r/attach_r1, suspend_r/suspend_r1)
- usb_reset, rx_err, nse_err, pid_cs_err, crc5_err are single-flop sampled in WB domain and used for RF interrupt source bits
- UTMI vendor status (VStatus_pad_i) captured in PHY (VStatus_r), then single-flop sampled in WB domain (utmi_vend_stat_r)
- LineState captured in PHY (LineState_r), read in WB domain without explicit synchronization
- RF per-endpoint int_stat bits are set in PHY domain and read/cleared in WB domain (level-based, no synchronizer)
3) WB -> PHY (RF controls to PHY)
- funct_adr is written in WB domain and used in PHY domain (usbf_pl) without explicit synchronization
- utmi_vend_ctrl and utmi_vend_wr are written in WB domain and single-flop retimed into PHY domain
4) PHY <-> WB (DMA request/ack for endpoints in usbf_ep_rf)
- dma_req_d computed in PHY domain and consumed in WB domain to assert dma_req_r (no explicit synchronizer)
- dma_ack asserted in WB domain and two-flop retimed into PHY domain (r4/r5) to produce dma_ack_i
- Additional hold logic (dma_req_hold) used to keep request asserted across clocks
5) Memory subsystem
- usbf_mem_arb sequential logic uses PHY clock; wclk port is unused internally
- All SSRAM-side control/data are effectively in PHY domain; WB-originated memory ops are evaluated in PHY via usbf_wb FSM
6) Top-level suspend/resume
- suspend_clr generated in PHY and single-flop sampled in WB (suspend_clr_wr)
- resume_req_r generated in WB and sampled in PHY by UTMI LS (no explicit synchronizer)
7) Interrupt outputs
- inta_o/intb_o are generated in WB domain from per-EP and RF sources (which originate in PHY domain but are treated as level signals)

- CDC design intent and constraints
- The design relies on level-stable handshakes and long-lived status bits rather than explicit multi-flop synchronizers on many buses.
- For STA/constraints, treat clk_i and phy_clk_pad_i as asynchronous; add set_clock_groups -asynchronous or equivalent false paths between them.
- For handshake-based paths (WB ack, DMA req/ack), consider multicycle/false-path exceptions aligned with the handshake protocol.
- If targeting ASIC or timing-closure-sensitive FPGA flows, consider adding explicit 2-flop synchronizers for single-bit controls (e.g., funct_adr, usb_reset, error flags, wb_req_s1) and CDC FIFOs or registered staging for multi-bit data (wb_addr_i/wb_data_i, ma_din/rf_din, utmi_vend_stat, ep*_dout), or ensure data is only observed in conjunction with a synchronized handshake.
- If enabling USBF_ASYNC_RESET, add reset deassertion synchronizers per domain.

-- Memory Arbitration and Buffer Integration --

- Scope
- Describes how the USB function core arbitrates access to the on-chip packet buffer memory (SSRAM) and how endpoint buffers are represented, updated, and consumed by the protocol engine and the Wishbone host.

- Buffer Memory Architecture
- Physical memory: external/internal synchronous SRAM interface exposed by usbf_mem_arb.
- Data width: 32-bit. Addressing is word-based on the SRAM bus and byte-based inside the packet engine.
- Size: parameterized by SSRAM_HADR. With SSRAM_HADR=9, the buffer size is 1024 words (4096 bytes).
- Clocks: arbitration and memory interface run in the PHY clock domain (phy_clk). The Wishbone interface runs in wb_clk.

- Masters of the Buffer Memory
- USB master (M-side): internal IDMA (usbf_idma) driven by the protocol engine (usbf_pe/usbf_pl) for packet RX/TX.
- Wishbone master (W-side): CPU/host access via usbf_wb, used to inspect/fill buffers and endpoint register file.

- Arbitration Policy (usbf_mem_arb)
- Priority: USB master has fixed priority over Wishbone.
- Selection: wsel = (wreq | wack) & !mreq. If the USB master is requesting, it owns the bus; otherwise the Wishbone master is serviced.
- Handshakes:
- USB side: mack is asserted combinationally with mreq (zero-wait for M-side). mcyc mirrors mack.
- Wishbone side: wack is generated in phy_clk domain using a registered pulse (wack_r) gated with !mreq to avoid contention.
- Multiplexing:
- sram_adr: driven by USB madr when M has bus; otherwise by W wadr.
- sram_dout (write data): driven by USB mdin or Wishbone wdin per wsel.
- sram_re is held asserted (1) for designs that don’t require explicit read enables; sram_we is qualified per selected master.
- Read data fan-out: sram_din (SRAM read) is broadcast back to both masters (mdout, wdout); only the acknowledged master consumes it.

- Wishbone Integration (usbf_wb)
- Address map select (final macros):
- USBF_RF_SEL = !wb_addr_i[12] selects register file (RF).
- USBF_MEM_SEL = wb_addr_i[12] selects the buffer memory window.
- RF path: direct read/write to usbf_rf (endpoint register file and core CSRs).
- Memory path: forwards ma_adr, ma_dout, ma_we, ma_req to arbiter; data returns via ma_din; ack returned as wb_ack_o after a short pipeline in wb_clk.
- Addressing to memory: word-aligned, using ma_adr[SSRAM_HADR+2:2] into the arbiter (lower 2 bits dropped for 32-bit words).

- Endpoint Buffer Model (usbf_ep_rf)
- Each endpoint has two 32-bit buffer descriptors: buf0 and buf1.
- Field layout:

- [31] NA (Not Allocated/Unavailable) flag. If set or if address is all-ones, buffer is considered not available (bufx_na).
- [30:17] Size (14 bits): remaining or configured payload size for the buffer.
- [SSRAM_HADR+2:0] Buffer start address (byte address). Low nibble is multiplexed contextually (see below).
- Original descriptor shadow: buf0_orig captures CPU-programmed buf0 for release/restore (buf0_rl).
- Endpoint matching and status: csr holds EP type/dir, enable/stall, transfer attributes, max packet size, and SW-controlled DP/BSEL bits.
- Interrupts: per-endpoint interrupt sources latched and masked; combined inta/intb are generated in wb_clk domain.

- Descriptor Update Semantics (driven by usbf_pe)
- Data movement updates are produced on idin and applied via strobes:
- buf0_set / buf1_set: write updated descriptor (address, size, dpid, buffer select) into the selected buffer.
- buf0_rl: release buf0 and restore from buf0_orig (used when a DMA cycle completes and buffer rotation is needed).
- uc_bsel_set / uc_dpd_set: update user buffer select and data PID fields packed into the low nibble when not writing a new address.
- idin packing rules:
- idin[31:17] = {buffer_done, new_size} for normal updates, or {4'h0, sizu_c} when out_to_small is asserted (special short OUT handling).
- idin[SSRAM_HADR+2:4] = new_adr[SSRAM_HADR+2:4] (word-aligned address increment) or old buf0_adr when out_to_small.
- idin[3:0] = new_adr[3:0] when buf_set_d is writing a new address, otherwise {next_dpid[1:0], next_bsel[1:0]} to carry DPID/BSEL status.
- Buffer availability flags:
- bufX_na = bufx[31] | (&bufx;_adr) indicates not available (either flagged or unallocated address).
- no_buf0_dma considers dma_en and in/out buffer thresholds to prevent starting transfers when there is insufficient data/space.

- Protocol Engine to Memory Data Path (usbf_pl → usbf_idma)
- RX (OUT) path:
- usbf_pd parses incoming packets; usbf_pe validates and enables rx_dma_en.
- usbf_idma accumulates 8-bit rx_data_st into 32-bit words in dtmp_r, issues writes when a word completes or on last bytes, and advances adr.
- Byte count sizu_c increments with rx_data_valid; transfer completes on rx_data_done or sized-out.
- After successful reception, usbf_pe computes new_adr/new_size and updates buf descriptors; it may NACK/ACK depending on size and policy.
- TX (IN) path:
- usbf_pe enables tx_dma_en; usbf_idma prefetches two 32-bit words (rd_buf0/rd_buf1) using mreq bursts and serves tx_data_st to usbf_pa.
- sizd_c tracks remaining bytes; send_data is held active until sizd_c reaches zero (or zero-length packet case).
- Addressing and wrap:
- adr is byte-based buffer pointer used by IDMA; madr is word-based address to SRAM.
- last_buf_adr = adr + buf_size; when adrw_next reaches last_buf_adr and dma_en is set, adrw_next wraps to 0 (circular addressing under DMA mode).

- Flow Control, Timeouts, and Acknowledgments
- The protocol engine enforces ACK/NACK/STALL/NYET policy based on endpoint state, buffer availability, max packet size constraints, and high-speed rules.

- RX ACK timeout and TX data timeout counters are parameterized for FS/HS and gate state transitions; on timeout, transfers abort and no descriptor update occurs.

- External DMA Signaling (System-level)
- Each endpoint raises dma_req when internal counters indicate data is ready for system DMA or space is available, honoring direction and max packet sizing.
- dma_ack is synchronized across clk domains (wclk↔clk) inside usbf_ep_rf; hold logic prevents premature deassertion if there is still work to do.

- Clock-Domain Crossings and Safety
- Arbitration and SRAM control are in phy_clk; Wishbone requests/acks cross into this domain (wack_r handshake).
- Endpoint DMA req/ack cross between wclk and clk safely using multi-flop synchronization (r1/r2/r4/r5 sequence).

- Design Guarantees and Assumptions
- USB master always wins arbitration to meet real-time requirements; Wishbone accesses are deferred but eventually serviced when USB is idle.
- Read data is broadcast on both master return paths; only the acknowledged side latches it.
- Word-aligned SRAM accesses; byte steering is handled internally in IDMA.
- Buffer descriptors default to 0xFFFF_FFFF (invalid) after reset and must be programmed by software before use.
- Parameter SSRAM_HADR defines both memory size and descriptor address field width; macros control the address decode between RF and MEM windows.

-- Endpoint Instantiation and Scalability --

The core supports up to 16 hardware endpoints, indexed 0–15. Endpoint 0 is always instantiated (control endpoint). Endpoints 1–15 are instantiated conditionally at compile time via Verilog defines:
- Define USBF_HAVE_EPx (x = 1..15) to instantiate a real endpoint register file (usbf_ep_rf) for that number.
- If a define is absent, a dummy endpoint (usbf_ep_rf_dummy) is instantiated. The dummy never matches tokens, raises no interrupts, and issues no DMA requests; its buffers read back as all 1s (0xFFFF_FFFF) and CSR/INT read as 0.

Contiguity requirement:
- Endpoints must be enabled contiguously starting from EP1. An elaboration-time check in usbf_top flags an error if EP(n+1) is enabled while EPn is not. EP0 is always present and not part of this contiguity check.

Addressing and register map scaling:
- All endpoint register banks are accessed through the shared register-file module (usbf_rf). The Wishbone address slice adr[6:2] selects the endpoint window:
- 0x04 → EP0, 0x05 → EP1, …, 0x13 → EP15.
- Within an endpoint window, adr[1:0] selects the register:
- 0: CSR (control/status), 1: INT (interrupt source/enable), 2: BUF0 descriptor, 3: BUF1 descriptor.
- Adding endpoints increases the accessible register windows linearly; no other address decode changes are required.

DMA and interrupt scalability:
- DMA request is a 16-bit vector dma_req[15:0], one bit per endpoint (index aligns with endpoint number). dma_ack[15:0] is returned per-endpoint. Disabled (dummy) endpoints drive dma_req=0.

- Endpoint interrupt outputs (inta, intb) are generated per endpoint and OR-aggregated across all instantiated endpoints in usbf_rf. Additional RF/global interrupt sources are also ORed in with per-class masks.

Endpoint selection and data path:
- Each real endpoint has a CSR field CSR[21:18] that holds the USB endpoint number it responds to, CSR[27:26] for direction/type (00=CTRL, 01=IN, 10=OUT).
- During token processing, the usbf_rf asserts ep_match for the endpoint whose CSR[21:18] equals the token's endpoint number. Only the matched endpoint's CSR, BUF0/BUF1, and flow-control outputs (dma_in_buf_sz1, dma_out_buf_avail) are forwarded to the packet engine, ensuring a single active endpoint per transaction.

Buffers and memory sharing:
- Every instantiated endpoint exposes two buffer descriptors (BUF0/BUF1): address [SSRAM_HADR+2:0] and size [30:17], with additional status/flags in the low nibble. All endpoints share the same external SSRAM through a single arbiter (usbf_mem_arb); enabling more endpoints does not change the memory interface width or timing, only the number of register banks and potential concurrent DMA requesters.

Build-time configuration guidance:
- To scale the number of endpoints, define USBF_HAVE_EP1 … USBF_HAVE_EPN contiguously for the highest endpoint required. Leave higher-numbered defines undefined.
- No RTL edits are needed beyond defines. The dma_req vector, interrupt aggregation, address decode, and arbitration automatically scale to the highest enabled endpoint.

Example (this build):
- With USBF_HAVE_EP1/2/3 defined, EP0–EP3 are active; EP4–EP15 are dummies. The dma_req width remains 16, but only bits [3:0] can assert.

-- Interrupt Wiring and Aggregation --

Overview
- Two interrupt outputs are provided at the top level: inta_o and intb_o. They are generated in the register file (usbf_rf) and updated on the WISHBONE clock (wclk = clk_i).
- Each endpoint (EP0..EP15) can assert two interrupt lines (epX_inta, epX_intb). usbf_rf ORs all endpoint interrupt lines and combines them with a set of controller (RF-level) interrupt sources that are maskable per top-level line.

Top-level interrupt equations (in usbf_rf)
- inta_o = OR over all epX_inta (X = 0..15) OR any(RF_INT_SRC & INTA_MASK)
- intb_o = OR over all epX_intb (X = 0..15) OR any(RF_INT_SRC & INTB_MASK)
- All terms above are registered on wclk.

RF-level interrupt sources and masks
- RF interrupt sources are collected in a 9-bit sticky vector int_srcb[8:0]. Bits are set on events and cleared on read of the RF INT STATUS register.
• int_srcb[8] = USB reset detected
• int_srcb[7] = RX error
• int_srcb[6] = Detach
• int_srcb[5] = Attach
• int_srcb[4] = Suspend end
• int_srcb[3] = Suspend start

• int_srcb[2] = NSE (no selected endpoint) error
• int_srcb[1] = PID check-sum error
• int_srcb[0] = CRC5 error (token)
- Two 9-bit masks gate RF sources into each top-level line:
• INTA_MASK = inta_msk[8:0]
• INTB_MASK = intb_msk[8:0]
- Mask programming (WISHBONE, usbf_rf address space, wclk domain):
• Write RF INT MASK register (addr 0x02):
- inta_msk <= din[8:0]
- intb_msk <= din[24:16]
- RF source clear mechanism:
• Read RF INT STATUS register (addr 0x03) with re=1 clears all bits of int_srcb. Clearing is per-bit, synchronous to wclk.

RF INT STATUS readback composition (addr 0x03)
- 32-bit read data layout: { 3'b000, int_srcb[8:0], 4'b0000, int_srca[15:0] }
• int_srcb[8:0] are the sticky RF-level sources described above; reading this register clears int_srcb.
• int_srca[15:0] is a live (non-sticky) per-endpoint summary: int_srca[N] = epN_inta | epN_intb.
- Mapping: bit 15..0 correspond to EP15..EP0, respectively.

Endpoint interrupt generation and masking (per usbf_ep_rf instance)
- Each endpoint implements its own 7-bit status vector int_stat[6:0] that latches endpoint-specific events when that endpoint matches the current token.
• int_stat[6] = OUT payload smaller than max packet size while DMA (out_to_small)
• int_stat[5] = PID sequence error
• int_stat[4] = Buffer1 update/complete event
• int_stat[3] = Buffer0 update/complete event
• int_stat[2] = Unexpected PID (UPID) event
• int_stat[1] = CRC16 error on received data
• int_stat[0] = Timeout (IN ACK timeout or OUT data timeout)
- Per-endpoint interrupt mask registers (written on wclk by accessing the endpoint's register window, offset 0x1):
• On write to EP INT MASK (offset 0x1 within the endpoint window):
- ienb[5:0] <= din[21:16] gate epX_intb
- iena[5:0] <= din[29:24] gate epX_inta
- Endpoint interrupt outputs are level-true and updated on wclk:
• epX_inta <= OR over (int_stat[i] & iena[i]) for i in the implemented set
• epX_intb <= OR over (int_stat[i] & ienb[i]) for i in the implemented set
- Clearing endpoint interrupt status:
• Read EP INT STATUS (offset 0x1 within that endpoint's register window) with re=1 clears all bits in that endpoint's int_stat.

Endpoint presence
- EP0 is always present. EP1..EP15 are conditionally instantiated via USBF_HAVE_EP# macros. If an endpoint is not instantiated, a dummy module ties epX_inta/intb low, so it does not contribute to inta_o/intb_o or int_srca.

Clocking and reset domain
- All interrupt status, mask, aggregation, and the top-level inta_o/intb_o are in the WISHBONE clock domain (wclk = clk_i). RF event latching from PHY domain signals is synchronized into wclk before affecting int_srcb.

Software usage summary
- To enable top-level reporting of RF events on inta_o/intb_o: write RF INT MASK (addr 0x02) to set inta_msk/intb_msk.
- To observe and clear RF events: read RF INT STATUS (addr 0x03). Reading clears only the RF portion (int_srcb), not endpoint statuses.
- To enable endpoint-generated interrupts on inta_o/intb_o: program per-endpoint iena/ienb in the endpoint's INT MASK register (offset 0x1) and ensure endpoint int_stat bits can set.
- To clear endpoint interrupt causes: read the endpoint's INT STATUS register (offset 0x1) for that endpoint window; this clears that endpoint's int_stat.
- The top-level lines assert whenever any enabled endpoint or masked RF source is active; they deassert when all enabled sources are cleared.

-- Reset Distribution and Sequencing --

Overview and polarity
- A single reset net rst_i drives the entire core. Reset is active-low: rst_i = 0 asserts reset. The same level is driven out on phy_rst_pad_o to reset the external UTMI PHY.

Clock-domain distribution
- The core has two principal clock domains: wb clock (clk_i) and PHY/packet-processing clock (phy_clk_pad_i). The same rst_i net feeds logic in both domains.
- Compile-time option USBF_ASYNC_RESET selects asynchronous assertion (negedge rst) for most sequential blocks; otherwise resets are synchronous to the local clock. In both cases, deassertion is sampled on the local clock.

Deassertion and sequencing requirements
- Ensure both clk_i and phy_clk_pad_i are running and stable before deasserting rst_i. Hold rst_i low for at least two rising edges of both clocks to guarantee all FSMs and counters initialize deterministically.
- Because reset is distributed into multiple clock domains without additional synchronization, system-level sequencing must avoid glitching rst_i and must meet each domain's recovery/removal timing.
- The external UTMI PHY reset (phy_rst_pad_o) mirrors rst_i; release the PHY reset together with or before the internal logic so that UTMI Rx/Tx handshake signals become valid when internal state machines leave reset.

Module-level effects on reset
- usbf_utmi_if and usbf_utmi_ls (PHY/link state): UTMI-LS FSM enters POR on rst_i and also whenever usb_vbus is asserted. Defaults are forced to Full-Speed: XcvSelect=FS, TermSel=FS pull-up on, OpMode bit-stuffing off until the state machine transitions. usb_suspend is cleared, usb_attached is cleared, mode_hs=0. After reset and with bus conditions present, the FSM proceeds through ATTACH (≈100 ms debounce via internal timer) to NORMAL. suspend_clr is generated during resume paths and is used at top-level to clear any latched resume_req.
- usbf_pl (protocol layer) and sub-blocks:
• usbf_pd (packet decoder): PID, token/data pipelines and CRC engines reset; FSM → IDLE.
• usbf_pa (packet assembler): CRC16 seed set to 0xFFFF; FSM → IDLE; no data/CRC emission pending.
• usbf_idma (internal DMA): Address/counters cleared, send_data deasserted; read/write pipelines idle; FSM → IDLE.
• usbf_pe (packet engine): All control/data-path FSMs → IDLE; buffer selectors, timers (ACK/DATA TO), and error flags cleared; no token/data is driven.
- usbf_mem_arb (SRAM arbiter): wack_r is cleared (PHY clock domain). SRAM read-enable is

statically 1 by design; write-enable is held low while requests are idle.
- usbf_wb (Wishbone interface): Main arbiter FSM (PHY clock domain) → IDLE; Wishbone handshake flops in clk_i domain start in the deasserted state and generate a clean first acknowledge after reset.
- usbf_rf (register file) and usbf_ep_rf (per-endpoint regs): Function address (funct_adr) resets to 0; interrupt masks clear; vendor-control write strobes clear. All endpoints: CSR fields default to 0; buf0/buf1 descriptors initialize to 0xFFFF_FFFF (not allocated), DMA counters/buffer-availability indicators clear, and interrupt status bits clear (and are cleared again upon read of the status register). inta/intb outputs are low until enabled and a source is set.

VBUS interaction
- The UTMI link-state FSM unconditionally re-enters POR while usb_vbus is asserted; after bus presence and timers satisfy attach timing, the FSM proceeds to ATTACH then NORMAL. System designers should ensure usb_vbus reflects actual attach/power state to avoid holding the link FSM in POR.

Cross-domain considerations
- Several handshakes cross domains (e.g., resume_req synchronizers, DMA req/ack, UTMI status). Reset drives all participating flops to benign values; there is no additional reset-release ordering between domains beyond the shared rst_i. Designers should therefore deassert rst_i only when both clocks are running.

Summary
- Assert rst_i low to reset the entire core and the external PHY. Keep it asserted until both clocks are stable. With USBF_ASYNC_RESET defined, assertion is asynchronous per domain; otherwise it is synchronous. On release, all FSMs start in IDLE/POR, endpoints are disabled with invalid buffer descriptors, interrupts are cleared, and the UTMI FSM performs its standard attach/normal transition when VBUS and line conditions permit.

-- PHY and External Memory Integration --

This core integrates a UTMI+ USB PHY and an external 32-bit synchronous SRAM through well-defined interfaces and clock-domain boundaries.

UTMI PHY integration
- Physical interface: The usbf_utmi_if module connects directly to the external PHY pads: DataOut/TxValid/TxReady (TX), DataIn/RxValid/RxActive/RxError (RX), XcvSelect, TermSel, SuspendM, OpMode, LineState, and usb_vbus. All PHY-side logic runs on phy_clk_pad_i; reset is rst_i (also routed to phy_rst_pad_o).
- Link state machine: usbf_utmi_ls performs HS/FS speed negotiation, reset detection, suspend/resume entry and signaling (drive_k), attach/de-attach detection, and line-state based timing. It produces mode_hs, usb_reset, usb_suspend, usb_attached, and suspend_clr. SuspendM is driven from usb_suspend and resume request state.
- Resume handling: Top-level resume_req_i is latched until a suspend_clr from the link state machine and then forwarded to usbf_utmi_ls. During resume signaling, TxValid is held asserted and DataOut forced while drive_k is active.
- Data path coupling: usbf_pl implements packet decode/encode (PD/PA), CRC, and transaction state machine (PE). It consumes rx_* and provides tx_* to usbf_utmi_if. TxFirst is used to stage the first TX byte to DataOut in sync with TxReady.
- Vendor I/O hooks: UTMI vendor status VStatus_pad_i is sampled and exposed via the register file; vendor control VControl_pad_o and load strobe VControl_Load_pad_o are driven by software through usbf_rf.

External memory (buffer SRAM) integration

- Memory organization: External buffer memory is 32-bit wide with address bus sram_adr_o[SSRAM_HADR:0]. With SSRAM_HADR=9, total capacity is 1024 words (4 KiB). sram_data_o/sram_data_i carry data; sram_we_o controls writes; sram_re_o is held high (combinational read enable).
- Masters and arbitration: usbf_mem_arb arbitrates between two masters in the PHY clock domain: (1) the packet-layer IDMA (M-port: madr, mdout, mdin, mwe, mreq, mack) used for line-rate payload transfers; and (2) the system/Wishbone software access (W-port: wadr, wdout, wdin, wwe, wreq, wack) used to inspect/fill endpoint buffers. Grant policy favors the IDMA; the W-port gains access only when no M-port request is active. Address and write-data are muxed based on the current grant; read-data is broadcast to both masters.
- Addressing and alignment: The IDMA operates on byte addresses adr[SSRAM_HADR+2:0] and packs/unpacks to 32-bit words internally. The SRAM is accessed using word addresses; IDMA and WISHBONE word indices are derived by dropping the two LSBs (e.g., ma_adr[SSRAM_HADR+2:2] to W-port). This yields contiguous, word-aligned buffer addressing for software and streaming byte addressing for the IDMA.
- Clock-domain crossing: All SRAM control (arbiter, M/W requests) runs on phy_clk_pad_i. usbf_wb synchronizes Wishbone bus requests from clk_i into the PHY domain (wb_stb_i/wb_cyc_i sampled on phy_clk) before generating memory requests, avoiding CDC hazards. W-port acknowledge (wack) is generated in the PHY domain and observed by usbf_wb in the system domain to complete Wishbone handshakes.
- Software mapping: The Wishbone address map is split by wb_addr_i[12]: 0 selects the register file (usbf_rf), 1 selects buffer SRAM via usbf_wb → usbf_mem_arb. Software accesses buffer memory as 32-bit words using ma_adr[SSRAM_HADR+2:2].
- Packet-layer DMA: usbf_idma performs bursty RX writes and TX reads with word coalescing and sub-word packing, handshaking via mreq/mack and mwe. Buffer sizes and descriptors come from the endpoint register file; the packet engine updates descriptors atomically after transfers.

Reset and status
- rst_i resets both PHY and system domains; phy_rst_pad_o mirrors rst_i. LineState and VStatus are synchronously sampled in the PHY domain. PHY/link status (usb_reset, usb_suspend, usb_attached, mode_hs) is fed into the packet layer and register file for software visibility and interrupt generation.

-- System Bus Address Space Integration --

Bus type and width:
- Wishbone classic, 32■bit data bus, single transfer (no pipeline), one■hot ack. No byte select signals are implemented; all accesses are 32■bit word accesses.
- Local address bus width inside the core is A12:0 (13 bits). As compiled here: USBF_UFC_HADR = 12.

Top■level address map (relative to the core’s base address B):
- The 8 KB local space is split by address bit 12.
- B + 0x0000_0000 … B + 0x0000_0FFF (wb_addr[12] = 0): Register File (RF) region
- B + 0x0000_1000 … B + 0x0000_1FFF (wb_addr[12] = 1): Buffer Memory window (SSRAM)

Address select and alignment:
- Region select:
- RF_SEL = !wb_addr[12]
- MEM_SEL = wb_addr[12]
- All accesses must be 32■bit word aligned. wb_addr[1:0] are ignored by the core (no byte enables).

Register File (RF) region (wb_addr[12] = 0):

- Register addressing uses wb_addr[8:2] (7■bit word index) inside RF. The core internally decodes:
- Main/global registers: word index adr[6:0] = 0x00 … 0x05
- 0x00: Main CSR (read). Also write: bit[5]=1 issues a resume request toward the UTMI layer.
- 0x01: Function Address (read/write). Bits [6:0].
- 0x02: Interrupt mask (read/write): {intb_msk[8:0], inta_msk[8:0]} packed as {7'h0, intb_msk, 7'h0, inta_msk}.
- 0x03: Interrupt sources (read/clear-on-read): packed as {int_srcb[8:0], int_srca[15:0]}.
- 0x04: Frame/time stamp (frm_nat) (read).
- 0x05: UTMI vendor interface:
- Read: Vendor status ({24'h0, utmi_vend_stat}).
- Write: Vendor control (low nibble din[3:0] latched to VControl; write also asserts VControl_Load strobe).
- Endpoint windows (per■endpoint, 16 windows, each 4 words):
- Base word index = 0x04 + ep_num (ep_num = 0…15). Within a window, wb_addr[3:2] selects:
- +0: EP CSR (configuration/status)
- +1: EP INT (interrupt status/masks per EP)
- +2: EP BUF0 descriptor
- +3: EP BUF1 descriptor
- Unimplemented endpoints are present as read■as■zero, write■ignored dummies (no side effects). In this build EP0…EP3 are implemented; EP4…EP15 are stubs.
- Notes:
- RF read data is returned when wb_addr[12]=0; RF writes are performed when wb_addr[12]=0 and wb_we_i=1.
- Only the addresses listed above have defined behavior; other RF word indices return zero and ignore writes.

Buffer Memory window (wb_addr[12] = 1):
- Maps the USB buffer SSRAM to the system bus.
- SSRAM addressing:
- Parameter SSRAM_HADR = 9 (as compiled here). The external SSRAM address bus is sram_adr[9:0] (10 bits), addressing 1024 words.
- Wishbone word address bits wb_addr[11:2] map to sram_adr[9:0] (i.e., wb_addr >> 2, truncated to 10 bits). The window thus covers 1024 x 32■bit words = 4096 bytes.
- Data path:
- Reads/writes are full 32■bit words. There are no byte enables.
- The arbiter (usbf_mem_arb) arbitrates between the PHY DMA master and the Wishbone port. System accesses to the memory window may incur wait states when DMA is active.

Handshake/timing on Wishbone:
- A valid bus request is wb_cyc_i=1 and wb_stb_i=1. The core pulses wb_ack_o=1 for exactly one cycle per completed transfer.
- RF accesses typically acknowledge after one core clock; memory window accesses acknowledge when the internal memory arbiter grants and returns data (wait states possible).

Integration guidelines:
- Map an 8 KB aperture for the core at base B. Use bit[12] of the local offset to select RF vs SSRAM.
- Ensure the bus fabric performs 32■bit word accesses only; no byte writes are supported.
- Treat wb_addr[11:9] as don’t■care in RF space (they are ignored by the RF decoder); software should keep them 0 for forward compatibility.
- Software should program endpoint buffer descriptors (BUF0/BUF1) with word■aligned SSRAM addresses consistent with the 4 KB buffer memory window.
- Interrupts are exposed as two lines (inta_o, intb_o); per■EP and core interrupt sources/masks are in

the RF region at 0x02/0x03.
- Function address is set via RF 0x01. A resume request toward the PHY can be triggered by writing bit[5]=1 at RF 0x00.
- The core reports its configured local widths at reset (A12:0 for Wishbone, SSRAM_HADR for memory). In this build: Wishbone A12:0, SSRAM window size 4 KB.

-- Verification Hooks and Testbench Notes --

This core exposes multiple built■in hooks and behaviors to aid verification and testbench development.

Clocks and reset
- Two clock domains: wb_clk=clk_i (register-file/host) and phy_clk=phy_clk_pad_i (UTMI/link/memory). Many CDC handshakes exist (e.g., request/ack flip-flop pairs). Reset is synchronous by default; define USBF_ASYNC_RESET to switch to async resets on many always blocks. Top drives phy_rst_pad_o=rst_i for convenience.

Compile-time switches (enable in TB via +define+...)
- USBF_DEBUG: enabled here; emits $display error lines when key FSM inputs are X/unknown. Helps catch TB driving issues.
- USBF_VERBOSE_DEBUG: optional; adds state-entry traces in PE/IDMA.
- USBF_HAVE_EP: controls which endpoint register files are real (u1..u15) vs dummy. Top initial block prints errors if gaps exist.
- Timing constants: USBF_T* macros parameterize UTMI low-speed/full/high-speed timing checks (used by UTMI LS state machine). Can be adjusted for accelerated sims.

UTMI interface TB driving
- Connect DataIn/RxValid/RxActive/RxError/LineState and monitor DataOut/TxValid/TxReady.
- Tx path handshake: usbf_utmi_if asserts TxValid while any of: tx_valid (from packet assembler), tx_valid_last, or internal resume drive_k is active. It holds TxValid until TxReady or drive_k_r are observed. TB must model TxReady backpressure and accept DataOut updates on TxReady or on tx_first.
- Rx path: provide byte stream with RxValid/RxActive; PID decode and token/data parsing is in usbf_pd. Inject errors by setting RxError or creating bad PID complement (pid_cks_err), or CRC mismatches; crc16_err and crc5_err are generated internally.
- Suspend/resume: drive resume_req_i (host) and observe suspend_clr pulse from utmi_if. UTMI LS FSM also reacts to LineState (J/K/SE0), usb_vbus_pad_i, and enforces timing windows; TB can emulate resets (SE0), chirp sequences, and HS/FS negotiation.
- Vendor UTMI test registers: write RF address 0x5 (usbf_rf space) to drive VControl_pad_o with VControl_Load_pad_o strobe; read UTMI vendor status via VStatus_pad_i (latched in u4).

Memory/IDMA and arbitration
- usbf_mem_arb connects UTMI-side IDMA and WISHBONE-side memory window to a shared synchronous SRAM; sram_re_o is tied high; sram_we_o pulsed as needed. Arbiter grants write-/read-side on phy_clk; wack asserts when wreq & !mreq & !wack previously, creating simple handshake. TB can use a trivial SRAM model (combinational read, registered write) with address sram_adr_o and 32-bit data.
- usbf_idma generates mreq/mwe and expects mack (granted in arbiter). Word packing/unpacking and last-word completion are handled internally; TB need only provide mack via arbiter and allow continuous access.

WISHBONE access and address map
- usbf_wb decodes wb_addr_i MSB per macros: USBF_RF_SEL(!wb_addr_i[USBF_UFC_HADR]) selects register file (RF), USBF_MEM_SEL(wb_addr_i[USBF_UFC_HADR]) selects memory window.

Note the last macro definitions in this file set USBF_UFC_HADR=12 and USBF_SSRAM_HADR=9; ensure the TB master uses the same widths. Memory path uses ma_req/ma_ack handshake (ack is immediate in arbiter).
- Reading RF vs MEM: wb_data_o is sourced from rf_din or ma_din based on RF_SEL. wb_ack_o is a short pulse derived from internal state; TB WB master should accept single-cycle acks.

Endpoint register file (RF) and DMA request to system
- usbf_rf instantiates EP0 real and EP1..EP15 per USBF_HAVE_EP flags. Each EP has CSR, buffer pointers (buf0/1), interrupt masks/status, and internal DMA bookkeeping.
- External DMA handshake: dma_req_o[15:0] are per-endpoint requests for system DMA (not the internal IDMA). TB can assert dma_ack_i bits to emulate DMA engine acceptance; EP logic keeps requests asserted/held across clock domains and uses holdoffs to prevent underrun/overrun.
- Interrupts: inta_o/intb_o combine EP interrupts and RF-level events (attach/detach, suspend/resume edges, errors). Some RF sources clear on read of the INT_SRC register (addr 0x3 in RF page); masks at addr 0x2. TB should read INT_SRC to clear latched sources.
- EP selection/match: usbf_pe uses token_fadr==funct_adr (RF addr 0x1) and ep_sel field to match endpoints (match_o). TB must program endpoint CSRs (type/dir/max packet/DMA enable) and buf descriptors before exercising traffic.

Error injection and observability
- PID/CRC errors: pid_cs_err, crc5_err, crc16_err outputs available; nse_err indicates token with no EP match. rx_err input forces framing errors.
- State/time behavior: frm_nat bus exposes frame number, SOF timing and microframe count for checking timing; useful for suspend/resume/reset sequences.
- USBF_DEBUG prints: PE and IDMA states emit $display when critical inputs are X; helps assert TB drives. Top initial $display summarizes configured endpoints and bus sizes.

Recommended TB scenarios (high-level)
- Reset and speed negotiation: drive usb_vbus_pad_i, SE0 for reset, then K/J line states per HS/FS negotiation; verify mode_hs, XcvSelect/TermSel/OpMode and usb_reset strobes.
- Enumeration path: program funct_adr; configure EP0 CSR; exercise SETUP/IN/OUT with correct PIDs, including zero-length and data PID toggling; verify token/data paths and ACK/NAK/STALL responses.
- DMA flow: configure a non-control EP with DMA enabled; preload buf0/1 descriptors; observe dma_req_o and respond with dma_ack_i; verify internal IDMA moves data to/from SSRAM and that buf descriptors update (buf*_set, uc_*_set) and INT bits set/clear.
- Backpressure and timeouts: throttle TxReady, assert RxActive without data to trigger timeouts (rx_ack_to, tx_data_to); verify INT_TO and PE transitions.
- Error cases: inject bad PID complement, CRC errors, sequence errors, and unmatched tokens; verify INT sources (crc5_err, pid_cs_err, nse_err, seqerr) and endpoint-side INT bits.

Waveform/coverage focus
- FSM coverage: usbf_pd, usbf_pa, usbf_pe, usbf_idma, utmi_ls state transitions; handshake crossings (dma_req/ack, WB ack strobes).
- Endpoint matrix: directions (IN/OUT/CTRL), transfer types (iso/bulk), DMA vs non-DMA, buf0/buf1 rolling, max packet edge cases (short/long/out_to_small).
- Interrupts and mask behavior, RF read-to-clear side effects, vendor register handshake.

Known behaviors useful in TB
- sram_re_o tied high; memory is effectively always-readable. Arbiter grants based on mreq vs wreq.
- ma_ack is generated combinationally from mreq inside wb/arb, so MEM window behaves 0-wait-cycle; ensure TB WB master samples wb_data_o accordingly.
- Zero-length IN support via send_zero_length path when max packet=0.

- Final macro definitions in this flattened file override earlier ones; ensure TB uses the last values for address MSBs and SSRAM size.

-- Integration Guidelines and Best Practices --

- Clocks and resets
- Two primary clock domains: wb_clk (clk_i) for the system/Wishbone and phy_clk (phy_clk_pad_i) for UTMI PHY and packet logic. Avoid crossing signals directly; use the existing bridge modules (usbf_wb, usbf_utmi_if, usbf_pl) which already include synchronizers.
- By default resets are synchronous (USBF_ASYNC_RESET not defined). If you require asynchronous resets, define USBF_ASYNC_RESET consistently across the project.
- phy_rst_pad_o is a copy of rst_i; ensure PHY reset timing meets your PHY's requirements.

- UTMI+ PHY integration
- Connect UTMI 8-bit interface: DataIn_pad_i/DataOut_pad_o, TxValid_pad_o, TxReady_pad_i, RxValid/Active/Error, LineState_pad_i, XcvSelect_pad_o, TermSel_pad_o, SuspendM_pad_o, OpMode_pad_o, usb_vbus_pad_i.
- Clock phy_clk_pad_i must be the UTMI data clock (typically 60 MHz). Maintain low-jitter and correct duty cycle.
- Speed negotiation and resume/suspend are handled in usbf_utmi_ls. Provide resume_req_i when system intends to remote-wake; it is internally latched and auto-cleared by suspend_clr.
- Verify usb_vbus polarity versus your PHY. The state machine uses usb_vbus in its state reset path; mismatched polarity can keep the link in POR.
- Vendor control/status: VControl_pad_o and VControl_Load_pad_o drive UTMI vendor registers; VStatus_pad_i is sampled. Ensure your PHY exposes these or tie off appropriately.

- Wishbone bus integration
- 32-bit classic Wishbone (no byte selects). Single read/write cycles; wb_ack_o is a single-cycle pulse generated in wb_clk domain.
- Address map split uses macro USBF_RF_SEL/USBF_MEM_SEL based on wb_addr_i[USBF_UFC_HADR]. Final macro definitions in this file set USBF_UFC_HADR=12 (i.e., wb_addr_i[12] selects):
- wb_addr_i[12]==0: Register file (RF) space (accessed via usbf_rf)
- wb_addr_i[12]==1: Buffer memory space (accessed via usbf_mem_arb)
- Ensure wb_addr_i is at least 13 bits wide. Align accesses to 32-bit words.

- External buffer memory (SSRAM) interface
- Signals: sram_adr_o[SSRAM_HADR:0], sram_data_i/o[31:0], sram_re_o, sram_we_o. Data width is 32-bit.
- Final macro sets USBF_SSRAM_HADR=9, yielding 4 KiB buffer RAM ((1<<(SSRAM_HADR+1))*4). Adjust macro as needed.
- usbf_mem_arb expects zero-wait-state behavior for the DMA port (mack=mreq). If your memory has wait states, insert adaptation logic or modify the arbiter.
- sram_re_o is held high; your memory wrapper should ignore it or treat it as permanently enabled.
- Addressing is word-based inside the core: ma_adr[SSRAM_HADR+2:2] maps to sram_adr_o. Provide proper byte ordering; IDMA packs bytes little-endian across the 32-bit word.

- Endpoints and register file
- EP0 is always present. Additional endpoints are controlled by defines USBF_HAVE_EP1..USBF_HAVE_EP15. This top includes defines for EP1–EP3; extend contiguously from EP1 upward (the top checks for gaps and prints errors).
- Register file usbf_rf exposes per-EP CSRs and descriptors and system-level status/interrupt control.

Access via RF address space (wb_addr_i[12]==0). Internal address mapping uses adr[6:2] per-EP windows; keep software consistent with this layout.
- Functional address (device address) is held in funct_adr and must be set by firmware after a SET_ADDRESS request.

- Buffer descriptors and data flow
- Each EP has Buf0/Buf1 descriptors and CSR. Descriptor fields (typical):
- [31]: not-available flag; [30:17]: buffer size in bytes; [SSRAM_HADR+2:0]: start address in words/bytes (see module access). Align buffers to 4-byte boundaries to match 32-bit memory port.
- The packet engine (usbf_pl/usbf_pe/usbf_idma) directly moves data between UTMI and the external buffer RAM based on these descriptors and the EP CSR (direction, max packet size, transfer type).
- Max packet size must match the host configuration and EP CSR; software should ensure descriptor sizes are consistent and large enough to avoid overflow conditions flagged by the core.

- External DMA handshake (system DMA)
- Top-level dma_req_o[15:0]/dma_ack_i[15:0] expose per-EP requests from usbf_ep_rf to system logic. Use these to schedule system DMA transfers between the USB buffer RAM and system memory, if desired.
- Protocol: EP asserts dma_req when it has space/data according to EP direction and CSR. System asserts corresponding dma_ack_i bit to acknowledge service; the EP will deassert request (it may hold the request depending on internal hold logic if more work remains). Design your DMA engine to operate per-bit and avoid missing short pulses (use registered acks).

- Interrupts
- inta_o and intb_o are the combined interrupt outputs. Masking and source registers are in the RF (main and per-EP). Software must read/clear latched status per documented locations in usbf_rf.
- Sources include protocol errors (PID CS, CRC5/CRC16), attach/detach, suspend/resume, USB reset, and EP buffer events.

- Error handling and flow control
- The core autonomously returns ACK/NACK/STALL/NYET based on EP state, buffer availability, transfer type, and HS/FS mode.
- Sequence errors and size mismatches are flagged; software should clear and/or reconfigure EPs accordingly.

- Power management and resume
- usb_suspend is exported (susp_o). To remote-wake, assert resume_req_i; the core manages K-chirp and resume timing, and auto-clears the request via suspend_clr.

- Timing and CDC constraints
- Constrain both clk_i and phy_clk_pad_i domains. Mark inter-domain paths through provided synchronizers as false paths if your STA flow requires it.
- Ensure UTMI timing (TxValid/TxReady and DataOut/DataIn) is met; the core registers outputs and samples inputs in phy_clk domain.

- Configuration macros sanity
- This file redefines macros; ensure your build system sets final intended values: USBF_UFC_HADR (default final 12), USBF_SSRAM_HADR (default final 9), USBF_HAVE_EPx as needed, and optionally USBF_ASYNC_RESET.
- Keep endpoint defines contiguous from 1..N to avoid RF instantiation gaps.

- Software bring-up checklist
- After reset: program EP0 CSR and buffers; write functional address after SET_ADDRESS; configure

masks in RF; set EP CSRs (direction/type/max packet size); allocate and program buf0/buf1 descriptors pointing into buffer RAM; service dma_req_o (if using system DMA); handle interrupts.

- Simulation and synthesis notes
- The design prints configuration info at sim start. Disable USBF_DEBUG/USBF_TEST_IMPL in production to reduce logic and messages.
- Verify memory wrapper semantics (read enable always high, zero-wait DMA path). If not met, adapt usbf_mem_arb or add a memory adapter.