Project Hierarchy

The TE-USB core is a layered design, with each layer implementing a clearly defined subset of the USB functionality. The following figure shows the project hierarchy:

Module	Description
usbEP0_XX	endpoint zero layer
usbEP0ctrl_XX	endpoint zero controller
usbRAM_XX	endpoint zero fifo
usbSIE	serial interface engine
usbEPdec	endpoint decoder
usbTRAC	transaction layer
usbTRACrt	transaction FSM
usbPAK	packet layer
usbPAKr	packet layer: receiver
usbPAKt	packet layer: transmitter
usbSP	parallel layer
usbSPr	S/P converter: receiver
usbSPt	S/P converter: transmitter
usbCRC16	CRC16 generation
usbNRZI	NRZI layer
usbNRZIr	NRZI layer: receiver
usbNRZIt	NRZI layer: transmitter
usbFRM	framing layer
usbFRMr	framing: receiver
usbFRMt	framing transmitter
usbINP	input signal synchronizer
usbDPLL	USB clock PLL
usbRST	USB reset detection

Table 1: Project Hierarchy

Framing Layer

The Framing Layer performs the detection and generation of USB frames, as well as the dection of USB reset conditions. To perform this functionality, support modules to recover the USB clock, and to synchronize the input signals with the chip internal clocks are required.

The framing layer provides the following interface to upper hierarchy levels:

- clk48: 48MHz clock from a quarz oscillator
- clk12: the buffered 12MHz clock from the DPLL
- rst: active-high asynchronous reset
- clk12o: the 12MHz clock from the DPLL
- usb_reset: active-high during USB reset condition
- frcv: active-high to denote receive mode
- ftxd: raw bits to be transmitted
- ftxen: active-high, while ftxd accepts data
- frxd: raw bits received
- frxen: active-high, while frxd carries valid data
- urxd: differential data received from USB transceiver
- urx0: active-high while single-ended zero condition is received
- utxd: differential data to be sent by USB transceiver
- utx0: active-high to send single-ended zero
- utxoe: active-high output endable to USB transceiver

The following figure shows the framing layer schematic:

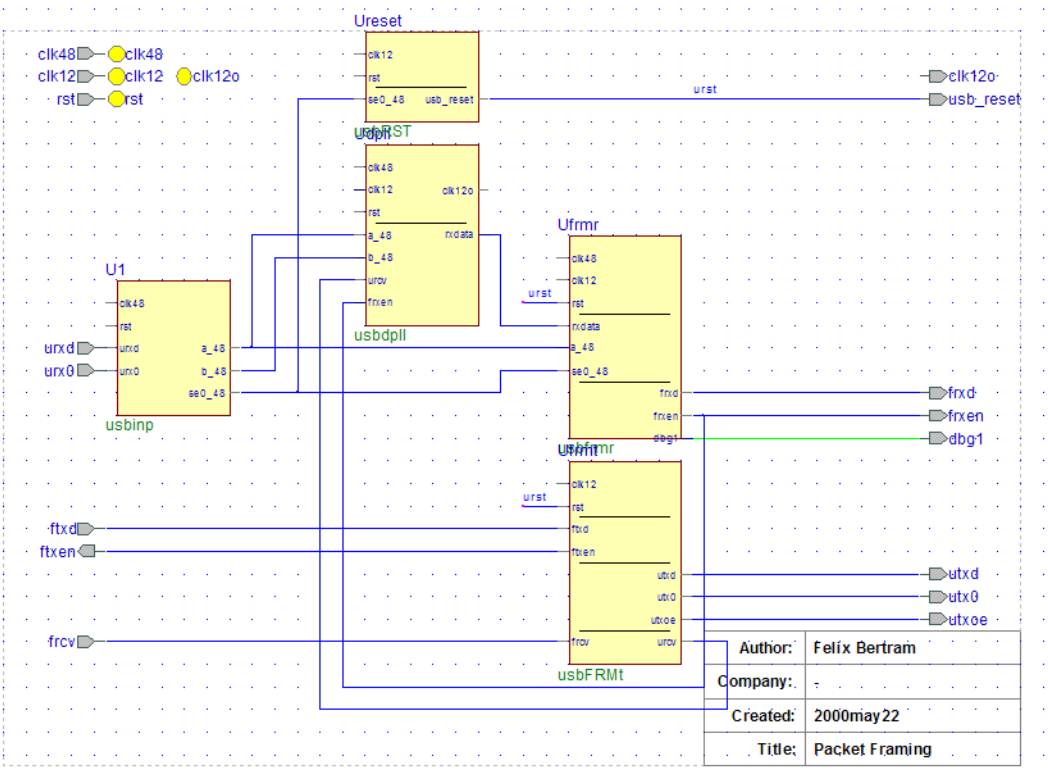

Figure 1: framing layer schematic

The module usbINP synchronizes the incoming USB signals to the 48MHz clock domain. The signals are synchronized to both clock edges, so that the DPLL may use this phase information to recover the 12MHz data clock from these signals.

The module usbDPLL runs off the 48MHz clock and recovers the 12MHz data clock from the incoming synchronized data. The clock is sent out to the upper hierarchy layers, buffered there, and fed back into the framing layer. The USB data are recovered directly from the DPLL and synchronized to the 12MHz clock. In case the USB data are sent, the 12MHz are derived from a simple clock divider. A state machine makes shure, that transitions from the send clock to the receive clock and vice versa are performed without glitches.

The module usbFRMr detects USB frames. It searches the KJKJKJKK preamble and starts receiving. Then it looks for the end-of-packet marker and closes the frame. This module is reset by the USB reset condition and not by the asynchronous reset.

The module usbFRMt generates USB frames. After entering transmit mode, the KJKJKJKK preamble is created. Then raw bits are sent to the bus, until receive mode is entered. The frame is closed by sending the end-of-packet marker. This module is reset by the USB reset condition and not by the asynchronous reset.

NRZI layer

The NRZI layer performs the NRZI encoding and decoding, as well as the bitstuffing and de-stuffing. Upper hierarchy levels communicate with the NRZI layer in term of data bits instead of raw (encoded) bits.

In addition to signals from lower hierarchy levels, the NRZI layer provides the following interface to upper hierarchy levels:

- stxd: data bits to USB
- stxen: active-high, while stxd accepts data
- srxd: data bits from USB
- srxen: active-high, while srxd carries valid data
- srcv: active-high, to denote receive mode

The following figure shows the NRZI layer schematic:

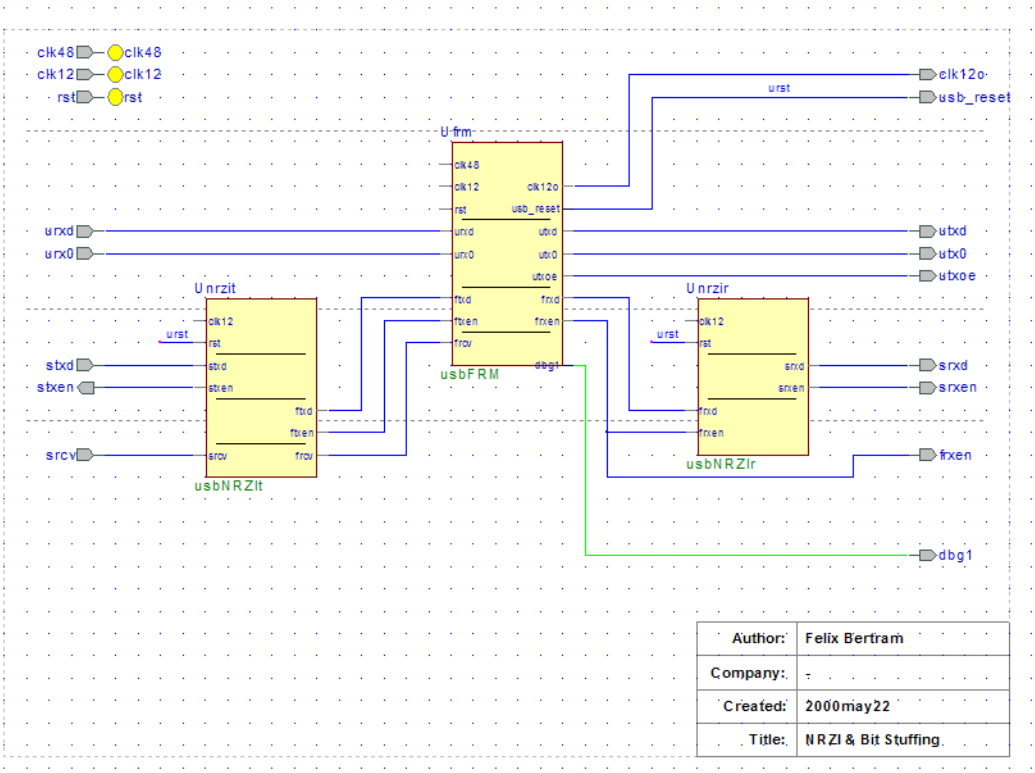

Figure 2: NRZI layer

The usbNRZIt module performs the NRZI encoding and bit stuffing. This module is reset from the USB reset condition and not by the asynchronous reset.

The usbNRZIr module performs the NRZI decoding and bit de-stuffing. This module is reset from the USB reset condition and not by the asynchronous reset.

Parallel Layer

The parallel layer performs the serial/parallel conversion. Upper hierarchy levels communicate with the Parallel layer in terms of 8-bit parallel data. Furthermore, the 16-bit CRC for outgoing USB data packets is generated here.

In addition to signals from lower hierarchy levels, the NRZI layer provides the following interface to upper hierarchy levels:

- ptxd(7:0): data to be sent to USB
- ptxen: active-high, while ptxd accepts data
- prxd(7:0): data received from USB
- prxen: active-high strobe to validate data on prxd
- prcv: active-high, to denote receive mode

The following figure shows the parallel layer schematic:

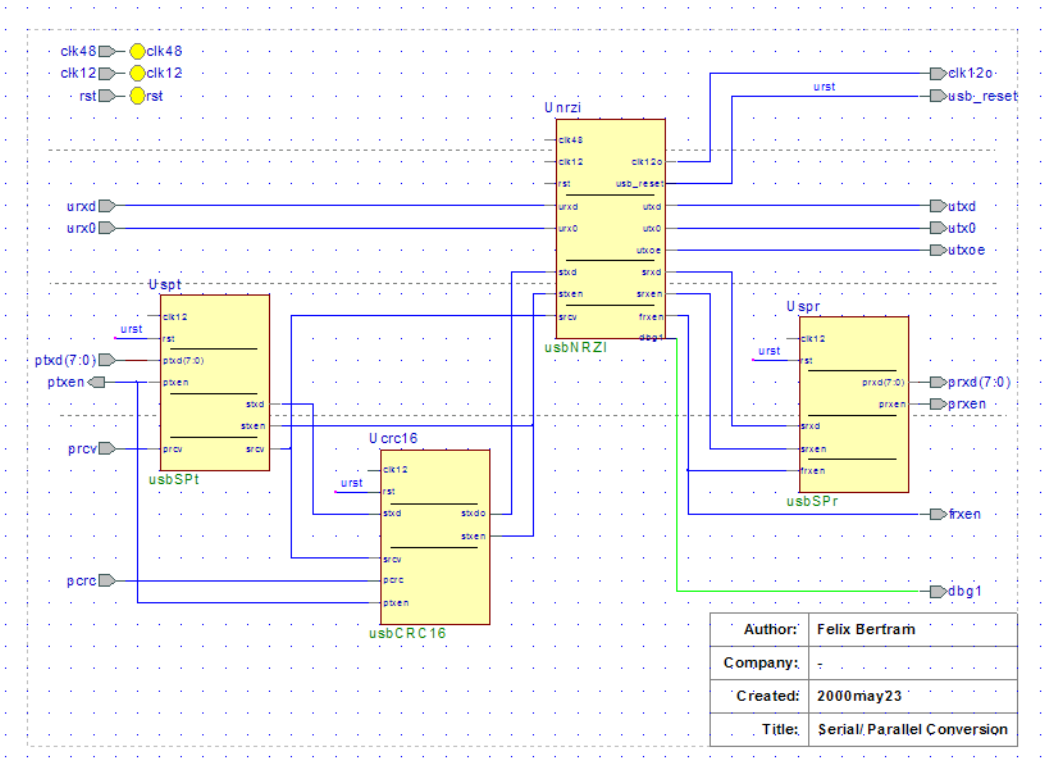

Figure 3:

The module usbSPt performs the parallel-to-serial conversion for outgoing data. This module is reset from the USB reset condition and not by the asynchronous reset.

The module usbCRC16 calculates and transmits the 16-bit CRC checksum protecting outgoing data packets. This module is reset from the USB reset condition and not by the asynchronous reset.

The module usbSPr performs the serial-to-parallel conversion for incoming data. This module is reset from the USB reset condition and not by the asynchronous reset.

Packet Layer

The packet layer decodes incoming packets and generates outgoing packets. Upper hierarchy levels communicate with the packet layer in terms of 8-bit parallel data, packet IDs, addresses, endpoints and frame numbers.

In addition to signals from lower hierarchy levels, the packet layer provides the following interface to upper hierarchy levels:

- txd(7:0): data to be sent to USB
- txen: active-high, while txd accepts data
- txpid: packet identifier of outgoing packet
- txdone: active-high strobe to denote end of packet transmission
- rxd(7:0): data received from USB
- rxen: active-high strobe to validate data on rxd
- rxpid: packet identifier of incoming packet
- rxvalid: active-high pulse denoting receival of a valid packet
- rxadep(10:0): address and endpoint information of incoming packet
- rxfrm(10:0): frame number of incoming SOF token

The following figure shows the packet layer schematic:

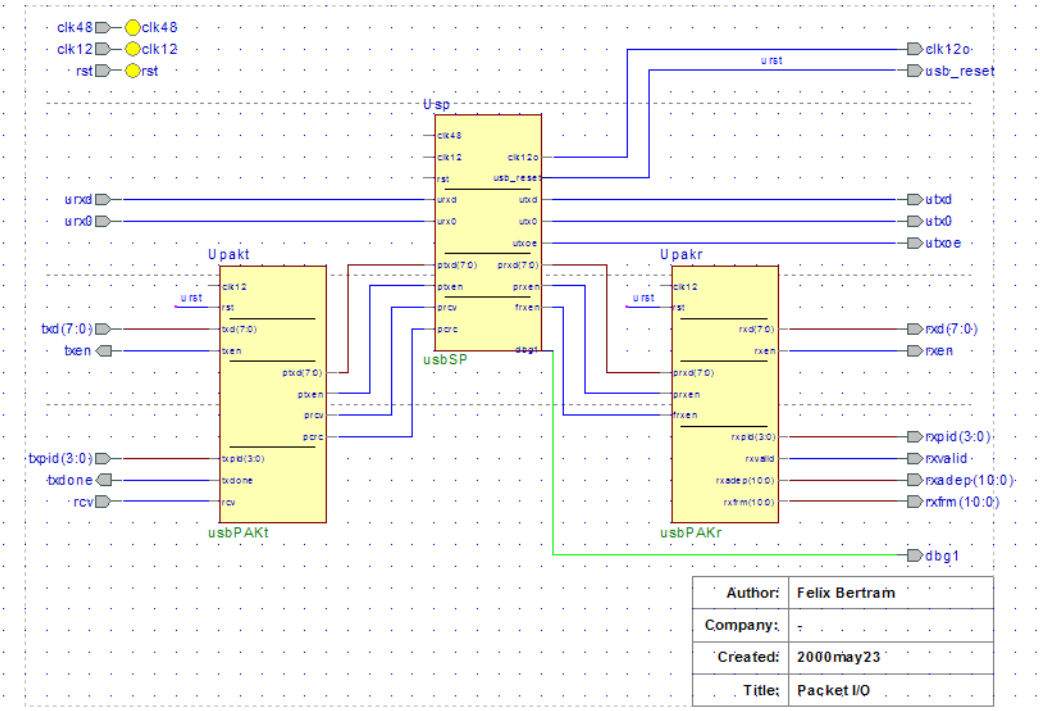

Figure 4: packet layer schematic

The module usbPAKt creates an outgoing packet with the PID specified by txpid. A pulse is generated, once the packet is transmitted. Additional data for DATA0 or DATA1 packets are fed into txd, and strobed with txen. This module is reset from the USB reset condition.

The module usbPAKr receives incoming packets. The PID information is provided on rxpid. Address and endpoint information is provided on rxadep. The current USB frame number if provided on rxfrm. Additional data from DATA0 or DATA1 packets are provided on rxd, and strobed with rxen. rxvalid goes high, whenever a valid packet was received. This module is reset from the USB reset condition.

Transaction Layer

The transaction layer governs complete USB transactions consisting of separate token, data and handshake phases.

In addition to signals from lower hierarchy levels, the transaction layer provides the following interface to upper hierarchy levels:

- epvld: active-high, when rxadep is a valid endpoint for this device
- episo: active-high, when rxadep is an isochronous endpoint on this device
- nak: active-high, when a NAK handshake is sent for this transaction
- stall: active-high, when a STALL handshake is sent for this transaction
- togglein: specify data toggle (DATA0 or DATA1)
- datain: active-high, to denote that the current DATAx packet contains further bytes
- in_trac: active-high, when valid token of IN transaction is received
- out_trac: active-high, when valid token of OUT transaction is received
- setup_trac: active-high, when valid token of SETUP transaction is received
- sof_trac: active-high, when valid SOF token is received
- setup_tok: active-high, while SETUP token is received

The following figure shows the transaction layer schematic:

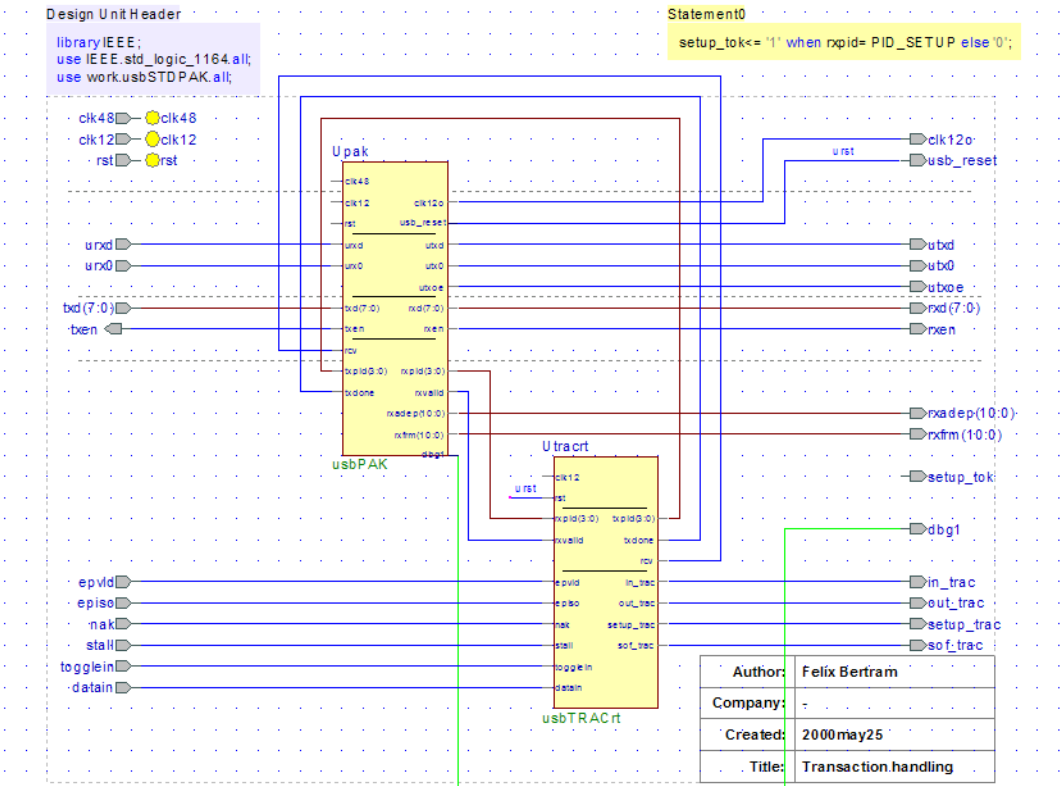

Figure 5: transaction layer schematic

The module usbTRACrt governs the proper execution of three-phase USB transactions. After receiving a token sent by the host, the required transaction is completed, according to the handshake signals, and the status of the datain line. This module is reset from the USB reset condition and not by the asynchronous reset.

Serial Interface Engine

The serial interface engine combines the transaction layer with the endpoint decoder, to decode address and endpoint information. The USB data and handshake signals are (de-) multiplexed, so that up to four endpoints have their separate control and data buses.

In addition to signals from lower hierarchy levels, the serial interface engine layer provides the following interface to upper hierarchy levels:

- txd0, txd1, txd2, txd3: data to be sent to USB for endpoints zero, 1, 2, and 3
- in_trac(3:0): in_trac signal decoded for endpoints zero, 1, 2, and 3
- out_trac(3:0): out_trac signal decoded for endpoints zero, 1, 2, and 3
- setup_trac(3:0): setup_trac signal decoded for endpoints zero, 1, 2, and 3
- toggle_in(3:0): toggle_in signal for endpoints zero, 1, 2, and 3
- data_in(3:0): data_in signal for endpoints zero, 1, 2, and 3
- uc_dadx(6:0): device address from microcontroller
- uc_wradx: strobe to store new device address
- epin_mask(3:0): generic to specify acceptance of IN tokens for endpoints zero, 1, 2, and 3
- epout_mask(3:0): generic to specify acceptance of OUT tokens for endpoints zero, 1, 2, and 3
- epsetup_mask(3:0): generic to specify acceptance of SETUP tokens for endpoints zero, 1, 2, and 3
- episo_mask(3:0): generic to specify isochronous endpoint feature for endpoints zero, 1, 2, and 3

The following figure shows the serial interface engine schematic:

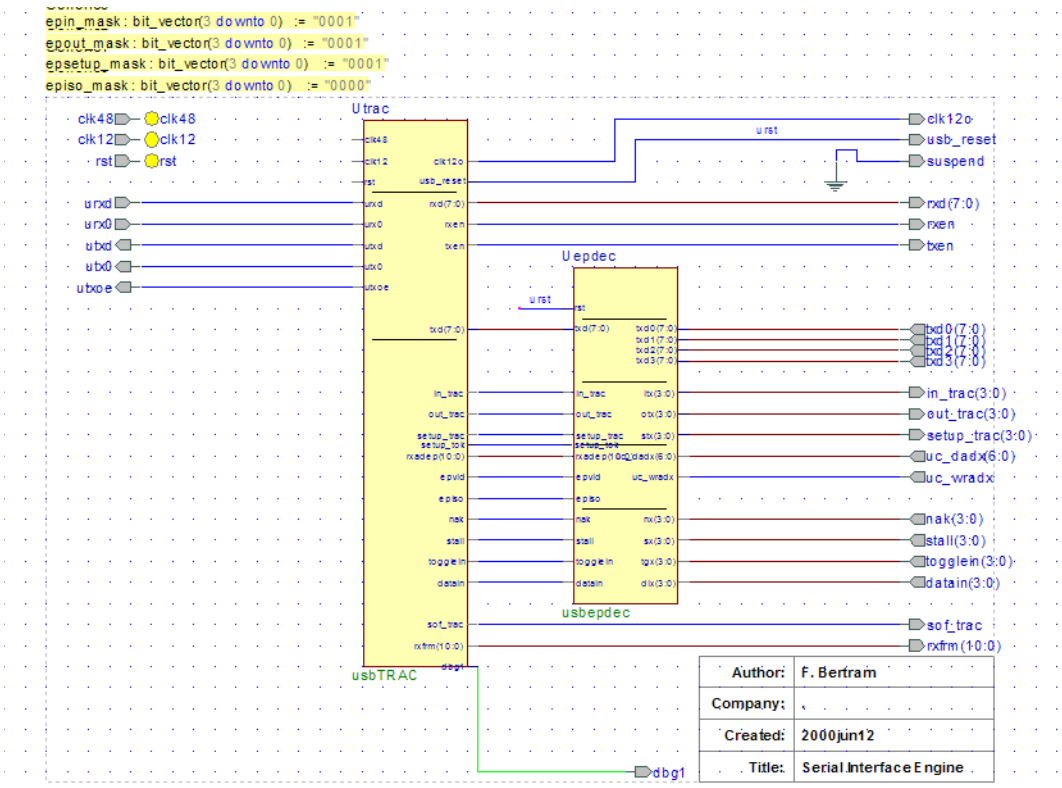

Figure 6: serial interface engine schematic

The module usbEPdec (de-) multiplexes the separate data and control busses and passes the according handshake signals or data to the transaction layer. It contains the device address register and generates the epvld signal for the transaction layer. The device address is cleared automatically, whenever a USB reset condition is encountered.

Endpoint Zero Layer

The endpoint zero layer implements all functionality required for the USB default pipe (endpoint zero). The critical USB timing is decoupled from the microcontroller timing with a fifo. All handshake signals to the Serial Interface Engine are created, and a control/status interface is provided to interface with the microcontroller.

In addition to signals from lower hierarchy levels, the endpoint zero layer provides the following interface to upper hierarchy levels:

- uc_adx(5:0): address bus from microcontroller to fifo
- uc_drd(7:0): data bus from fifo to microcontroller
- uc_dwr(7:0): data bus from microcontroller to fifo
- uc_wren: data strobe from microcontroller to fifo
- uc_status(7:0): status from endpoint controller to microcontroller
- uc_ctrl(7:0): control from microcontroller to endpoint controller
- uc_wrctrl: control strobe from microcontroller to endpoint controller

The following figure shows the endpoint zero layer schematic:

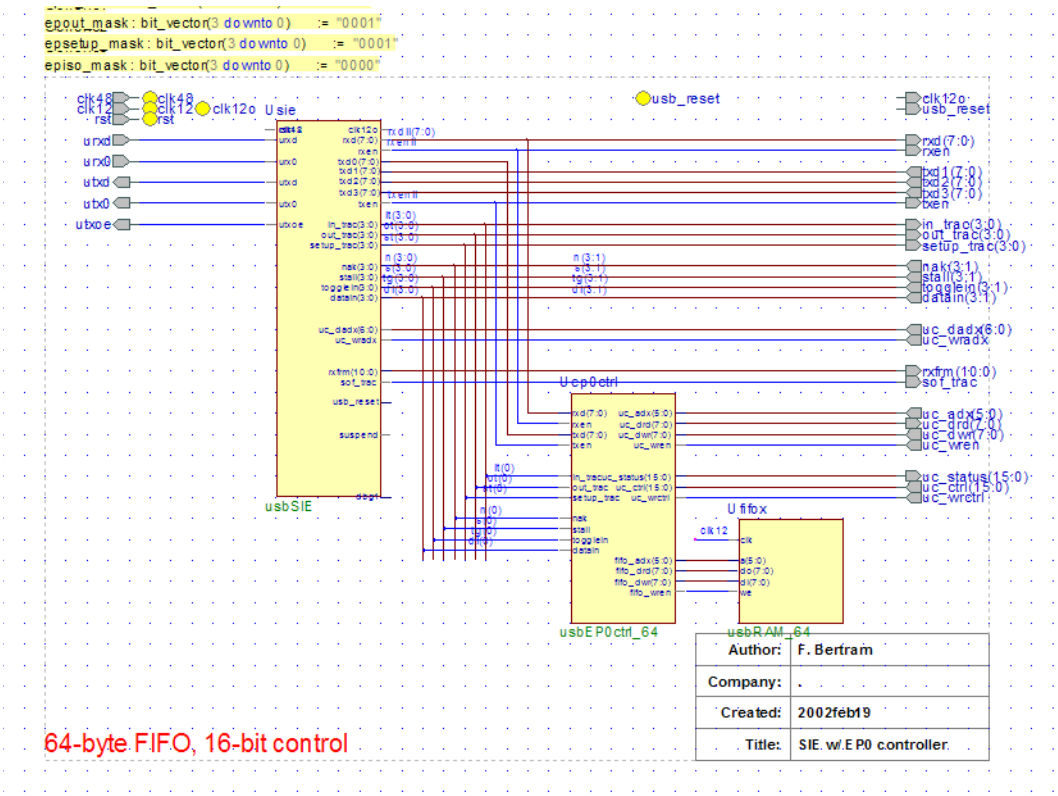

Figure 7: endpoint zero layer

The module usbEP0ctrl_xx creates all handshake signals required by the serial interface engine, according to the commands received from the microcontroller via uc_ctrl. It contains address counters to read/write the fifo from the USB, and arbitrates access to the fifo between USB and the microcontroller.

The module usbRAM_xx implements a simple single-ported memory to perform as fifo for USB data.

Testbenches

The TE-USB core has been verified using a dedicated USB testbench package, which is described in a separate document (Packet-oriented USB Verification Kit). This testbench package sends USB packets to the core, and verifies the responses sent by the UUT with the projected results.

Distinct testbenches exist, to verify the following submodules of the design:

- DPLL: Verification of clock and data extraction of the digital PLL
- FRM: Verification of frame detection and generation
- NRZI: Verification of NRZI en-/decoding and bit (de-) stuffing
- TRAC: Verification of the USB transaction layer
- EP0: Verification of the USB endpoint zero layer
- bugs: Verification of bugs, which have been detected during the development and maintenance

In addition to testing the submodules, the following testbenches have been created to verify the functionality of a complete USB system:

- firmware_8: various testbenches with the firmware written in VHDL. The advantage of these testbenches is, that they are independent from the microprozessor/ firmware. The firmware written in VHDL shows the application of the TE-USB in an easy to understand and easy to simulate way. Furthermore, the firmware written in VHDL simulates extremely fast.
- TE-XC2Se: complete simulation project of the reference application which programs the flash memory of the TE-XC2Se FPGA development board via USB. This is a real-life project, based on a freeware CPU (XR16) written in Verilog. Simulation tends to be lengthy and requires a mixed-language simulator.

Reference Implementations

TE-XC2Se FPGA board

The TE-XC2Se is a sophisticated FPGA development board, based on a Xilinx Spartan-IIe device. The board is configured and powered via USB. After setting the board into its factory configuration, the board acts as a USB Human Interface Device, which accepts a Xilinx bitstream, which is then programmed into the on-board flash memory.

The board has proven to work reliable in daily work scenarios. As Xilinx bitstreams are CRC protected, even single bit errors would be detected- if they occurred. Proper execution of the USB protocol has been verified with a CATC USB protocol analyzer.

The design combines a freeware CPU (XR16) written in Verilog with glue logic written in VHDL- and the TE-USB core.

A specification of the TE-XC2Se board and its USB download mechanism may be found in a separate document (Spartan-IIe Development Platform, User's Manual).

CompactFlash card reader with USB

As a student project, a CompactFlash card reader has been implemented in close cooperation with the Technical University Hamburg-Harburg. The hardware is based upon the TE-XC2Se board, with a simple passive adaptor to attach the CompactFlash connector to it. The design is recognized by Windows XP as a standard removable harddisk.

The design has proven to work reliable under real-life conditions with testfiles of several megabytes in size. Proper execution of the USB protocol has been proven with a CATC USB protocol analyzer.

The design combines a freeware CPU (XR16) written in Verilog with glue logic, and additional endpoint controllers written in VHDL- and the TE-USB core as a precompiled netlist. As a sidenote, a single student implemented this design within three months.

A description of the project (in german language) is available as a separate document (Studienarbeit USB2CF).