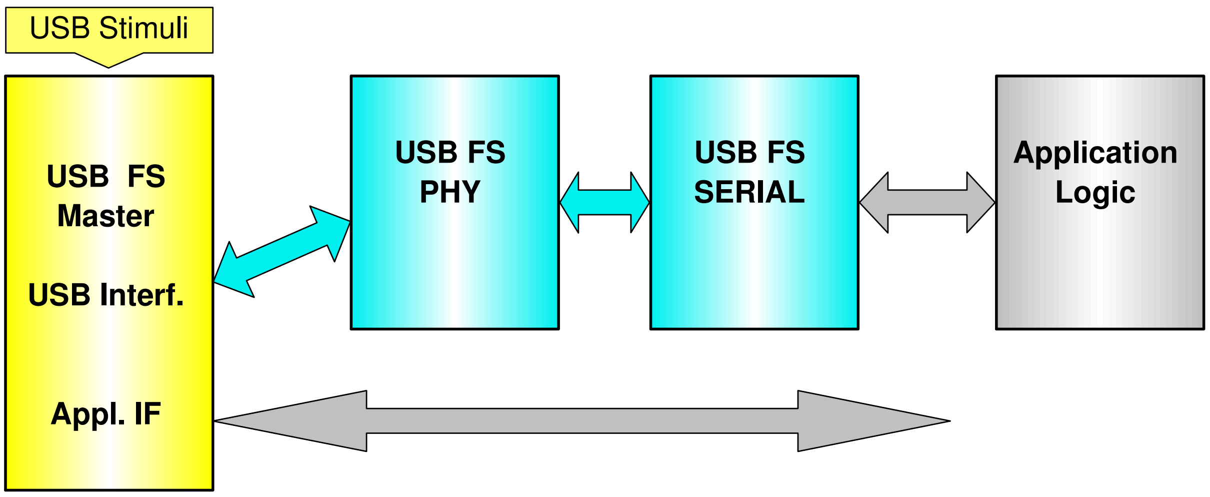
USB FS (Full Speed) Test Bench - Documentation

05 Mar 2011 Martin Neumann

This USB FS test bench has been used with the Model Sim VHDL Simulator, however any other 'event driven' Simulator should work as well.

When developing an USB application device, a test scenario should be able to simulate the USB host and of course the private developed application as well. The concept of the USB test environment is shown in the following figure:



The above block diagram symbolizes a test bench usb_tb.vhd consisting out of the following parts:

USB FS Master

• usb_commands.vhd	- the USB command package
• usb_stimuli.vhd	- a renamed test case file
• usb_fs_monitor.vhd	- logs all usb activities in files Transcript and Result.out
• usb_fs_master.vhd	- the top entity with usb and application interface

USB FS PHY

Open Cores USB Phy, designed by Rudolf Usselmanns according to the USB 2.0 UTMI interface specification.
This Verilog design has been translated to VHDL. From the RX portion are two versions derived, the original version that operates with a 48 MHz clock and a modified version for a 60 MHz clock.

- usb_rx_phy_60MHz.vhdl
- usb_tx_phy.vhdl
- usb_phy.vhdl

USB Serial

Open Cores USB Serial, designed by Joris van Rantwijk, an USB to RS232 converter. This VHDL design operates either at FS or HS, here tied to FS.

- usb_pkg.vhdl
- usb_init.vhdl
- usb_control.vhdl
- usb_transact.vhdl
- usb_packet.vhdl
- usb_serial.vhdl

Others

The USB top entities (usb_phy.vhdl and usb_serial.vhdl) are warped together in file usb_slave.vhdl while any application files have been omitted since those are truly design specific. The USB Masters application interface deals directly with the application inputs and outputs of the usb_serial.vhdl design.

In the following we concentrate on the yellow portion of the above figure. To have a stable test bench environment even with various test cases, each test case file (in the examples just usb_tc_01.vhd and usb_tc_02.vhd) must have an identical Entity structure. For this reason it is recommended to copy and rename the actual test case to usb_stimuli.vhd prior to a new simulation run. In a Windows environment this could be simplified with the TC_Copy.bat (after adopting it's directory structure to your environment). Place a link to this file on the desktop and then just drop a new tc_xx.vhd on this link - and viola, a new usb_stimuli.vhd is created.

When looking at a sample usb_tc file, we see that the USB access is done as procedure calls from within a normal process without sensitivity list. This allows us to insert WAIT statements at any time, we may insert LOOP constructs and any other legal sequential statements.

Before we go into more details, all usb and the two list commands log their output at two places :

- First in the Model Sim specific Transcript file (with other simulators - wherever they place their comments and warnings) and
- Second in a simulator independent file named Result.out, created in the simulator home directory (I force this to be the top level of an actual test structure with the Model Sim 'change directory' command).

After a successful simulation run it is a good idee to rename the test log file Result.out so that this file name matches that of the original test case file, e.g. **tc_02.out** for later reference.

Now to the first procedure calls provided:

- **list(T_No, positive);**
- **list("String");**

List() procedure calls are just for better orientation within a larger test case, the first statement loads an otherwise not used test case signal **T_No** to the specified value and logs this in the Transcript and Result.out file , the

list (”Reset completed”);

procedure writes just a comment, in this case „Reset completed”, at this two places. It is recommended to add the signal T_No on top of the the waveform watch list of all design units, in Model Sim this has to be added in a 'xx.do' file like

```
add wave -noupdate -divider {USB_PHY}
add wave -noupdate -format Literal -radix decimal   /usb_tb/usb_fs_master/test_case/t_no
add wave -noupdate -format Logic -radix hexadecimal  /usb_tb/usb_fs_slave_1/usb_phy_1/*
```

The next following procedure calls carry all usb as first parameter. This out-parameter 'usb' is only internally used and controls the proper timing sequence. The USB Token Commands OUT, IN, SOF and SETUP are realized with the following very similar are procedure calls

- **out_token(usb, device_addr, endp_addr);**
- **in_token(usb, device_addr, endp_addr);**
- **sof_token(usb, frame_no);**
- **setup(usb, device_addr, endp_addr);**

To use the handy hexadecimal notation, the device_addr must be specified with 8 instead 7 bits, and similarly frame_no with 12 instead 11 bits. This MSB bit is dropped in all the four cases.

    setup(usb, X"00", X"0");

is then a setup command , in this case to usb-device address 0, endpoint 0;
The procedures adds internally the synchronization preamble, PIP, its complement, correct bit-stuffing and CRC-5 in all this cases.

The data procedure calls

- **send_D0(usb, wr_data);**
- **send_D1(usb, wr_data);**
- **send_D2(usb, wr_data);**
- **send_Dm(usb, wr_data);**

require as wr_data parameter a byte array in the following notation :

    send_D0(usb,(X"80",X"06",X"00",X"01",X"00",X"00",X"12",X"00"));

After transfer of the last byte the procedure will add automatically the internally computed CRC-16 value. The length of the array is freely selectable (also no data is allowed), however in case of a single byte the notation gets clumsy, we have to specyfy wr_data with index 0 as:

    Send_D0(usb,wr_data(0) => X"12");

The handshake procedures are very simple and self explaining:

- **send_ACK(usb);**
- **send_NAK(usb);**
- **send_STALL(usb);**
- **send_NYET(usb);**

We covered now the Token, Data and Handshake Commands. A specialty is the USB reset - the USB reset condition is met if the USB Master pulls down both data lines to low levels (SE0) for at least 10 ms, the USB Slave may recognize the reset condition already after 2.5 µs. To keep simulation time to a minimum, the

- **send_RES(usb);**

command forces the SE0 condition for only 5 µs, both design units, the USB PHY and USB Serial will detect the reset condition within this time frame.

Whenever we expect a response from the USB slave we must issue

- **wait_slv(usb);**

This command waits for any slave response, is it either a handshake command or a stream of data.

The details of this as of all other transfers are logged by the USB_monitor entity - this entity monitors all bus activities and writes the result into the Transcript and Result.out files. The Monitor detects all Token, Data and Handshake commands In case the commands are initialized from the USB Master, the commands are preceded by **‘Send’**, otherwise by **‘Recv’**. Data transfer results will start a new line every 16 bytes. The last two bytes are the CRC-16 bytes witch may be ignored.
In the following attachment we see a simple usb_stimuli.vhd and the corresponding Result.out file.

```
--==================================================================================================--
--  FILENAME    : USB_tc_02.vhd                                                                     --
--  DESCRIPTION : FS-USB test case                                                                  --
--  Designer    : Martin Neumann                                                                    --
--  Description : Copy and rename this file to  usb_stimuli.vhd before running a new simulation!   --
--==================================================================================================--
--                                                                                                  --
-- Change history -----------------------------------------------------------------------------------
-- Version:| Author:| Date:     | Comment:                                                          --
-- --------|--------|-----------|--------------------------------------------------------------------
--   1.0   | MN     |07 Feb 2011| Initial version                                                   --
--==================================================================================================--
LIBRARY work, IEEE;
  USE IEEE.std_logic_1164.ALL;
  USE IEEE.std_logic_arith.ALL;
  USE work.usb_commands.ALL;

ENTITY USB_Stimuli IS PORT(
  -- Test Control Interface --
  USB             : OUT usb_action;
  t_no            : OUT NATURAL;
  -- Application Interface
  clk             : IN  STD_LOGIC;
  rst_neg_ext     : OUT STD_LOGIC;
  RXval           : IN  STD_LOGIC;                    -- RX bytes available
  RXdat           : IN  STD_LOGIC_VECTOR(7 DOWNTO 0); -- Received data bytes
  RXrdy           : OUT STD_LOGIC := '0';             -- Application ready for data
  RXlen           : IN  STD_LOGIC_VECTOR(7 DOWNTO 0); -- Number of bytes available
  TXval           : OUT STD_LOGIC := '0';             -- Application has valid data
  TXdat           : OUT STD_LOGIC_VECTOR(7 DOWNTO 0); -- Data byte to send
  TXrdy           : IN  STD_LOGIC;                    -- Entity is ready for data
  TXroom          : IN  STD_LOGIC_VECTOR(7 DOWNTO 0); -- No of free bytes in TX
  TXcork          : OUT STD_LOGIC := '1');            -- Hold TX transmission
END USB_Stimuli;

ARCHITECTURE sim OF usb_stimuli IS

  SIGNAL   rd_data : byte_array(0 TO 7);
  SIGNAL   TX_load : STD_LOGIC := '0';

BEGIN
  p_stimuli_data : PROCESS
  BEGIN
    list("*****************************");
    list("*   Results of  tc_02.vhd    *");
    list("*****************************");
    list(" ");
    list(T_No, 10);
    rst_neg_ext <= '0';
    WAIT FOR 301 ns;
    rst_neg_ext <= '1';
    WAIT FOR 40 ns;
    list("Reset completed, initializing");
    TX_load <= '1';
    list(T_No, 20);
    setup(usb, X"00", X"0");                                            --Setup to  addr 0, endp 0 ..
    send_D0(usb,(X"80",X"06",X"00",X"01",X"00",X"00",X"40",X"00"));      -- .. 'Get Device descriptor'
    wait_slv(usb);                                                       --Recv ACK
    in_token(usb, X"00", X"0");                                         --Send IN-Token
    wait_slv(usb);                                                       --Recv Data1 Device Discriptor
    send_ack(usb);                                                      --Send  ACK
    out_token(usb, X"00", X"0");
    send_D0(usb);                                                       --Send Zero Data
    Wait_slv(usb);
    list(T_No, 30);
    setup(usb, X"00", X"0");                                      --Setup to  addr 0, endp 0 ..
    send_D0(usb,(X"00",X"05",X"03",X"00",X"00",X"00",X"00",X"00"));  -- .. 'Set Address'
    wait_slv(usb);                                                 --Recv ACK
    in_token(usb, X"00", X"0");                                   --Send IN-Token
    wait_slv(usb);                                                 --Recv Data0 zero Length
    send_ack(usb);                                                --Send  ACK

    --Now we may use the new address :
    list(T_No, 50);
    out_token(usb, X"03", X"1");                                  --Send Out-Token to Endpoint 1
    send_D0(usb, (X"11",X"22",X"33",X"44",X"55",X"66",X"77",X"88"));
    wait_slv(usb);
    list(T_No, 51);
    out_token(usb, X"03", X"1");                                  --Send Out-Token to Endpoint 1
    send_D0(usb, (X"11",X"12",X"13",X"14",X"15",X"16",X"17",X"18"));
    wait_slv(usb);
    TXcork <= '0';                                                --Release TX buffer
    FOR i IN 0 TO 5 LOOP
      list(T_No, 60+i);
      in_token(usb, X"03", X"1");                                 --Send IN-Token to Endpoint 1
      wait_slv(usb);                                               --Recv Data ?
      send_ack(usb);                                              --Send  ACK
    END LOOP;
    IF usb_busy THEN                                              --is a  usb_monitor signal
      WAIT UNTIL NOT usb_busy;
    END IF;
    WAIT FOR 300 ns;
    send_RES(usb);
    WAIT FOR 1 us;
    ASSERT FALSE REPORT"End of Test" SEVERITY FAILURE;
  END PROCESS;

  p_rd_data : PROCESS
    VARIABLE i : NATURAL := 0;
  BEGIN
    WAIT UNTIL rising_edge(clk);
    RXrdy <= '1';
    IF i < 8 THEN
      RXrdy <= '1';
      IF RXval ='1' THEN
        rd_data(i) <= RXdat;
      END IF;
    ELSE
      RXrdy <= '0';
    END IF;
  END PROCESS;

  p_wr_data : PROCESS
    VARIABLE i : NATURAL := 0;
  BEGIN
    WAIT UNTIL rising_edge(clk);
    IF i < 333 AND TXrdy ='1' and TX_load ='1' THEN
      TXval <= '1';
      TXdat <= CONV_STD_LOGIC_VECTOR(i,8);
      i := i +1;
    ELSE
      TXval  <= '0';
    END IF;
  END PROCESS;

END sim;
```

Figure 2) usb_tc_02.vhdl renamed to usb_stimuli.vhd

```
                 *****************************
                 *   Results of  tc_02.vhd    *
                 *****************************

                 Test_No 10
                 Reset completed, initializing
                 Test_No 20
  1862.409 ns    Send Setup: Address 0x00, Endpoint 0x0,  CRC5 0x02
  4862.328 ns    Send Data0 0x80 0x06 0x00 0x01 0x00 0x00 0x40 0x00  0xDD 0x94
 13599.797 ns    Recv ACK
 15452.206 ns    Send IN-Token: Address 0x00, Endpoint 0x0,  CRC5 0x02
 18845.557 ns    Recv Data1 0x12 0x01 0x10 0x01 0x02 0x00 0x00 0x40  0x9A 0xFB 0x9A 0xFB 0x20 0x00 0x00 0x00
 29992.797 ns          ..... 0x00 0x01  0xB4 0x2C
 33861.545 ns    Send ACK
 35533.631 ns    Send OUT-Token: Address 0x00, Endpoint 0x0,  CRC5 0x02
  38533.55 ns    Send Data0 0x00 0x00
 41926.901 ns    Recv ACK
                 Test_No 30
  43779.31 ns    Send Setup: Address 0x00, Endpoint 0x0,  CRC5 0x02
 46779.229 ns    Send Data0 0x00 0x05 0x03 0x00 0x00 0x00 0x00 0x00  0xEA 0xC7
 55598.663 ns    Recv ACK
 57451.072 ns    Send IN-Token: Address 0x00, Endpoint 0x0,  CRC5 0x02
 60844.423 ns    Recv Data1 0x00 0x00
 64024.665 ns    Send ACK
                 Test_No 50
 65696.751 ns    Send OUT-Token: Address 0x03, Endpoint 0x1,  CRC5 0x1C
  68696.67 ns    Send Data0 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88  0x5E 0xBF
 77516.104 ns    Recv ACK
                 Test_No 51
  79352.12 ns    Send OUT-Token: Address 0x03, Endpoint 0x1,  CRC5 0x1C
 82352.039 ns    Send Data0 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x95  0xA7
 91089.508 ns    Recv ACK
                 Test_No 60
 92941.917 ns    Send IN-Token: Address 0x03, Endpoint 0x1,  CRC5 0x1C
 96351.661 ns    Recv Data0 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09  0x0A 0x0B 0x0C 0x0D 0x0E 0x0F
107498.901 ns         ..... 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19  0x1A 0x1B 0x1C 0x1D 0x1E 0x1F
117990.421 ns         ..... 0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29  0x2A 0x2B 0x2C 0x2D 0x2E 0x2F
128481.941 ns         ..... 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39  0x3A 0x3B 0x3C 0x3D 0x3E 0x3F
139055.426 ns         ..... 0x26 0xF7
141612.734 ns    Send ACK
                 Test_No 61
143268.427 ns    Send IN-Token: Address 0x03, Endpoint 0x1,  CRC5 0x1C
146678.171 ns    Recv Data1 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49  0x4A 0x4B 0x4C 0x4D 0x4E 0x4F
157825.411 ns         ..... 0x50 0x51 0x52 0x53 0x54 0x55 0x56 0x57 0x58 0x59  0x5A 0x5B 0x5C 0x5D 0x5E 0x5F
168316.931 ns         ..... 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69  0x6A 0x6B 0x6C 0x6D 0x6E 0x6F
178808.451 ns         ..... 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79  0x7A 0x7B 0x7C 0x7D 0x7E 0x7F
189463.901 ns         ..... 0x9B 0xBA
192021.209 ns    Send ACK
                 Test_No 62
193693.295 ns    Send IN-Token: Address 0x03, Endpoint 0x1,  CRC5 0x1C
197103.039 ns    Recv Data0 0x80 0x81 0x82 0x83 0x84 0x85 0x86 0x87 0x88 0x89  0x8A 0x8B 0x8C 0x8D 0x8E 0x8F
208250.279 ns         ..... 0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99  0x9A 0x9B 0x9C 0x9D 0x9E 0x9F
218823.764 ns         ..... 0xA0 0xA1 0xA2 0xA3 0xA4 0xA5 0xA6 0xA7 0xA8 0xA9 0xAA 0xAB 0xAC 0xAD 0xAE 0xAF
229315.284 ns         ..... 0xB0 0xB1 0xB2 0xB3 0xB4 0xB5 0xB6 0xB7 0xB8 0xB9 0xBA 0xBB 0xBC 0xBD 0xBE 0xBF
239888.769 ns         ..... 0x5C 0x6C
242446.077 ns    Send ACK
                 Test_No 63
 244101.77 ns    Send IN-Token: Address 0x03, Endpoint 0x1,  CRC5 0x1C
247511.514 ns    Recv Data1 0xC0 0xC1 0xC2 0xC3 0xC4 0xC5 0xC6 0xC7 0xC8 0xC9 0xCA 0xCB 0xCC 0xCD 0xCE 0xCF
258740.719 ns         ..... 0xD0 0xD1 0xD2 0xD3 0xD4 0xD5 0xD6 0xD7 0xD8 0xD9 0xDA 0xDB 0xDC 0xDD 0xDE 0xDF
269314.204 ns         ..... 0xE0 0xE1 0xE2 0xE3 0xE4 0xE5 0xE6 0xE7 0xE8 0xE9 0xEA 0xEB 0xEC 0xED 0xEE 0xEF
279969.654 ns         ..... 0xF0 0xF1 0xF2 0xF3 0xF4 0xF5 0xF6 0xF7 0xF8 0xF9 0xFA 0xFB 0xFC 0xFD 0xFE 0x00
291034.929 ns         ..... 0xA1 0x61
293526.665 ns    Send ACK
                 Test_No 64
295198.751 ns    Send IN-Token: Address 0x03, Endpoint 0x1,  CRC5 0x1C
298592.102 ns    Recv Data0 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09  0x0A 0x0B 0x0C 0x0D 0x0E 0x0F 0x10
309739.342 ns         ..... 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19  0x1A 0x1B 0x1C 0x1D 0x1E 0x1F 0x20
320230.862 ns         ..... 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29  0x2A 0x2B 0x2C 0x2D 0x2E 0x2F 0x30
330722.382 ns         ..... 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39  0x3A 0x3B 0x3C 0x3D 0x3E 0x3F 0x41
341295.867 ns         ..... 0x98  0x8A
343853.175 ns    Send ACK
```

```
                  Test_No 65
345525.261 ns  Send IN-Token: Address 0x03, Endpoint 0x1,  CRC5 0x1C
348935.005 ns  Recv Data1 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49  0x4A 0x4B 0x4C 0x5A 0x87
359360.953 ns  Send ACK
364950.966 ns  USB Reset detected for     5098.223  ns
```

Figure 3) Result.out result file from test case usb_tc_02.vhdl

All lines without a time stamp are user comments, created via a list() command. All other lines are written by the USB Monitor. It permanently monitors the two USB lines and whenever it detects a sync signal, it evaluates the command, regardless if it is caused by the USB Master or the USB Slave. Only the direction signal is obtained from the USB Master. The USB reset detector checks the USB lines for SE0 events over 200 ns, a reset is signaled as soon as SE0 exceeds 2,5 µs.