Dünner Kirchweg 77
32257 Bünde
Germany
www.trenz-electronic.de

Packet-oriented USB Verification Kit

2002-March-26

User's Guide

Overview

Simulation is a vital part of design verification. To ease the task of test vector creation, a powerful VHDL package has been created. The main features of this package are:

- USB Packet-oriented
- Stimuli creation
- Response verification
- written in VHDL

With USB packets being the atomic items of the package, complex testbenches can easily be created, which are isolated from USB protocol and timing details. Devices may be stimulated from USB and their responses to USB may be verified. This allows for simulation of complete systems including microcontroller and firmware.

Setup

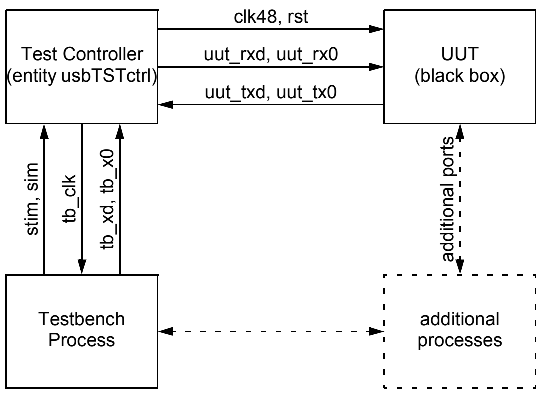

Figure 1: Simulation: Testbench Setup.

Figure 1 illustrates the basic testbench setup. It consists of the following main components:

- *Test Controller.* The Test Controller stimulates the UUT and verifies the UUT's responses. In case of a mismatch, a report is asserted.
- *UUT.* This is the Unit Under Test. It is treated as a black box and exposes its behavior via its outside ports only.
- *Testbench process.* The Testbench Process controls the Test Controller's operation based on USB Packets.

Test Controller

The main component of the USB testbench is the Test Controller. The Test controller consists of three blocks: interface to the UUT, interface to the Testbench Process, internal logic.

The interface to the UUT provides the following functionality:

- provide clock and reset to the UUT
- provide USB stimuli to the UUT
- receive USB responses from the UUT

The interface to the Testbench Process provides the following functionality:

- provide clock information to the testbench process
- receive control data from the testbench process
- receive bit data from the testbench process

The internal logic implements the following functionality:

- clock recovery for responses received from the UUT
- verification of data from the testbench process and the date received from the UUT

Whenever the Test Controller detects a mismatch between the response received from the UUT, and the pattern generated by the Testbench Process, the simulation is aborted with an assertion of severity failure. Please refer to your VHDL simulator's documentation, to see how assertions are handled.

Figure 1 summarizes the Test Controller ports. It is important to understand, that the Test Controller interfaces the Testbench Process with the UUT, instead of the Testbench Process interacting with the UUT directly. The signals are explained below:

- *sim*. This signal should be '1' during simulation, and be set to '0' at the end of simulation. The *clk48* signal will create a 48MHz clock, as long as *sim* is '1'.
- *stim*. This signal is '1', whenever the Test Controller sends stimuli to the UUT. When packets are received from the UUT and verified by the Test Controller, this signal must be set to '0'

- *clk48*. This signal drives a 48MHz clock for use by the UUT
- rst. This signal drives an asynchronous, active high reset for use by the UUT
- *uut_rxd*. This signal drives the USB stimuli to the UUT. It is ‘1’ for USB ‘J’ conditions and ‘0’ for USB ‘K’ conditions.
- *uut_rx0*. This signal drives the USB stimuli to the UUT. It is ‘1’ for USB SE0 conditions, and ‘0’ under all other conditions.
- *uut_txd*. This signal receives the USB responses from the UUT. It is ‘1’ for USB ‘J’ conditions, and ‘0’ for USB ‘K’ conditions.
- *uut_tx0*. This signal receives the USB responses from the UUT. It is ‘1’ for USB SE0 conditions, and ‘0’ under all other conditions.
- *tb_xd*. This signal receives the USB patterns from the Testbench Process. It is ‘1’ for USB ‘J’ conditions, and ‘0’ for USB ‘K’ conditions.
- *tb_x0*. This signal receives the USB patterns from the Testbench process. It is ‘1’ for USB SE0 conditions, and ‘0’ under all other conditions.
- *tb_clk*. This signal provides timing information to the Testbench process.

Depending on the UUT’s USB Transceiver, the format of the USB signals may deviate from the xd/x0 paradigm used by the Test Controller. To overcome this, some simple combinational logic is required.

Table 1: Test Controller Ports.

signal	direction	purpose
sim	in	‘1’ during simulation
stim	in	‘1’ to stimulate UUT
clk48	out	48MHz clock output
rst	out	asynchronous reset
uut_rxd	out	connected to UUT’s urxd
uut_rx0	out	connected to UUT’s urx0
uut_txd	in	connected to UUT’s utxd
uut_tx0	in	connected to UUT’s utx0
tb_xd	in	testbench xd signal
tb_x0	in	testbench x0 signal
tb_clk	out	testbench 12MHz clock

Unit Under Test

The UUT or Unit Under Test is treated as a black box, which is exercised primarily through the USB ports. Although additional exercising of the UUT’s additional outside ports is possible, this is not covered by this testbench.

The UUT model should not include the USB Transceiver, to allow easy interfacing with the Test Controller.

Testbench Process

The Testbench process implements the specific sequence of USB packets, which is used to exercise the UUT. Testbench creation is relatively simple, as the testbenches are widely invariant to timing details: On bit level, the testbenches are completely invariant to timing changes, while on packet level the exact sequence of packets depends on the speed of the microcontroller.

Figure 2 illustrates the Testbench process. The testbench package is implemented in *usbTSTPAK*. The signal *sim* is used to limit the total simulation time by disabling the clock oscillators inside *usbTSTctrl*. The signal *stim* is used to distinguish stimuli from responses. The example transmits the Setup Stage of a GetDescriptor (Device) Request and verifies that the Device answers with a correct ACK handshake.

```
use work.usbTSTPAK.all;
...
sim<= '1', '0' after 15us;
...
process
  constant get_dscr: packetBUF:=
    ( x"80", x"06", x"00", x"01",
      x"00", x"00", x"40", x"00");
begin
  stim<= '1';
  packetSETUP(tb_clk, tb_xd, tb_x0,
      "0000000", "0000");
  packetDATA1(tb_clk, tb_xd, tb_x0,
      get_dscr);
  stim<= '0';
  packetACK(tb_clk, tb_xd, tb_x0);

  wait;
end process;
```

Figure 2: Testbench Process

Packet procedures

The following paragraphs summarize the procedures contained in *usbTSTPAK*. All these routines create the pattern of a single USB packet, which the Test Controller either applies to the UUT as a stimulus, or which is verified with the response received from the UUT.

packetIN, packetOUT, packetSETUP

```
(signal clk:  in  STD_LOGIC;
 signal xd:   out STD_LOGIC;
 signal x0:   out STD_LOGIC;
 addr:        in  STD_LOGIC_VECTOR
                  (6 downto 0);
 ep:          in  STD_LOGIC_VECTOR
                  (3 downto 0));
```

packetIN, *packetOUT* and *packetSETUP* initiate an IN, OUT or SETUP transaction to the address/endpoint specified by *addr* and *ep*.

packetSOF

```
(signal clk:  in  STD_LOGIC;
 signal xd:   out STD_LOGIC;
 signal x0:   out STD_LOGIC;
 frame:       in  STD_LOGIC_VECTOR
                  (10 downto 0));
```

packetSOF sends a Start-Of-Frame token with the frame number specified by *frame*.

packetDATA0, packetDATA1

```
(signal clk:  in  STD_LOGIC;
 signal xd:   out STD_LOGIC;
 signal x0:   out STD_LOGIC;
 data:        in  packetBUF);
```

packetDATA0 and *packetDATA1* send/receive a DATA package with the contents specified by *data*. In case *data* is omitted, a zero-length packet is sent/received. The 16-bit CRC is calculated internally and not included in the data vector.

packetACK, packetNAK, packetSTALL

```
(signal clk:  in  STD_LOGIC;
 signal xd:   out STD_LOGIC;
 signal x0:   out STD_LOGIC);
```

packetACK, *packetNAK* and *packetSTALL* send/receive an ACK, NAK or STALL handshake.

packetIDLE

```
(signal clk:  in  STD_LOGIC;
 signal xd:   out STD_LOGIC;
 signal x0:   out STD_LOGIC;
 cycles:      in  INTEGER);
```

packetIDLE keeps the bus in idle (J) state for the number of USB bit cycles specified by *cycles*.

packetRESET

```
(signal clk:  in  STD_LOGIC;
 signal xd:   out STD_LOGIC;
 signal x0:   out STD_LOGIC);
```

packetRESET keeps the bus in SE0 state for more than 2.5us to generate a USB reset condition.

Example

Get Descriptor (Device)

The following example lists the complete code to verify the GetDescriptor (Device) command. It is assumed, that the UUT implements an 8-byte FIFO. Please note, that this code will need to be slightly adapted, depending on your actual descriptor data, the speed of your microcontroller, and the set of supported USB requests.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use work.usbTSTPAK.all;

entity TBexample is
end TBexample;

--------------------------------------------------------------------------------
architecture test of TBexample is
  ------------------------------------
  -- interface of unit under test
  component myDevice
    port(
      -- clocks & reset
      clk48      : in  STD_LOGIC;
      rst        : in  STD_LOGIC;
      -- usb transceiver
      urxd       : in  STD_LOGIC;
      urx0       : in  STD_LOGIC;
      utxd       : out STD_LOGIC;
      utx0       : out STD_LOGIC;
      utxoe      : out STD_LOGIC
      );
  end component;

  ------------------------------------
  -- UUT signals
  -- clocks & reset
  signal clk48      : STD_LOGIC;
  signal rst        : STD_LOGIC;
  -- usb transceiver
  signal urxd       : STD_LOGIC;
  signal urx0       : STD_LOGIC;
  signal utxd       : STD_LOGIC;
  signal utx0       : STD_LOGIC;
  signal utxoe      : STD_LOGIC;

  ------------------------------------
  -- test controller signals
  signal tb_xd      : STD_LOGIC;
  signal tb_x0      : STD_LOGIC;
  signal tb_clk     : STD_LOGIC;
  signal sim        : STD_LOGIC;
  signal stim       : STD_LOGIC;

begin
  ------------------------------------------------------------------------------
  -- instantiate test controller
  Uctrl: usbTSTctrl port map(
    sim    => sim,
    stim   => stim,
    clk48  => clk48,
    rst    => rst,
    tb_xd  => tb_xd,
    tb_x0  => tb_x0,
    tb_clk => tb_clk,
    uut_txd=> utxd,
    uut_tx0=> utx0,
    uut_rxd=> urxd,
    uut_rx0=> urx0);

  ------------------------------------
  -- instantiate unit under test
  UUT: myDevice port map(
    -- clocks & reset
    clk48      => clk48,
    rst        => rst,
    -- usb transceiver
    urxd       => urxd,
    urx0       => urx0,
    utxd       => utxd,
    utx0       => utx0,
    utxoe      => utxoe);

  ------------------------------------------------------------------------------
  -- testbench process
  process
    -- get descriptor (device) command
    constant gdd: packetBUF:= (
    x"80",  --  0: bmRequestType      0x80   Device-to-host
    x"06",  --  1: bRequest           0x06   GET_DESCRIPTOR
    x"00",  --  2: wValue             0x0100 DEVICE
    x"01",  --  3:
    x"00",  --  4: wIndex             0x0000
    x"00",  --  5:
    x"40",  --  6: wLength            0x0040
    x"00"); --  7:

    -- device descriptor 0:7
    constant dd_0_7: packetBUF:= (
    x"12",  --  0: bLength            0x12   18 bytes
    x"01",  --  1: bDescriptorType    0x01   DEVICE
    x"10",  --  2: bcdUSB             0x0110 Rev 1.10
    x"01",  --  3:
    x"00",  --  4: bDeviceClass       0x00
    x"00",  --  5: bDeviceSubClass    0x00
    x"00",  --  6: bDeviceProtocol    0x00
    x"08"); --  7: bMaxPacketSize0    0x08   08 bytes

    -- device descriptor 8:15
    constant dd_8_15: packetBUF:= (
    x"D0",  --  8: idVendor           0x0BD0 (3024)
    x"0B",  --  9:
    x"00",  -- 10: idProduct          0x0200
    x"02",  -- 11:
    x"00",  -- 12: bcdDevice          0x0100 Rev 1.0
    x"01",  -- 13:
    x"01",  -- 14: iManufacturer      0x01
    x"02"); -- 15: iProduct           0x02

    -- device descriptor 16:17
    constant dd_16_17: packetBUF:= (
    x"00",  -- 16: iSerialNumber      0x00
    x"01"); -- 17: bNumConfigurations 0x01

    constant addr0: STD_LOGIC_VECTOR(6 downto 0):= "0000000";
    constant ep0:   STD_LOGIC_VECTOR(3 downto 0):= "0000";
  begin
    --------------------------------
    -- init USB and test controller
    sim  <= '1';
    stim <= '1';
    tb_xd<= '1';
    tb_x0<= '0';

    --------------------------------
    -- apply usb reset
    packetRESET(tb_clk, tb_xd, tb_x0);

    --------------------
    -- send get descriptor (device) SETUP stage
    stim<= '1';
    packetSETUP(tb_clk, tb_xd, tb_x0, addr0, ep0);
    packetDATA0(tb_clk, tb_xd, tb_x0, gdd);
    stim<= '0';
    packetACK  (tb_clk, tb_xd, tb_x0);

    --------------------
    -- NAKed data stages may be required here,
    -- while the CPU copies the descriptor data
    -- into the USB fifos. these stages would look like this:
    --    stim<= '1';
    --    packetIN   (tb_clk, tb_xd, tb_x0, addr0, ep0);
    --    stim<= '0';
    --    packetNAK  (tb_clk, tb_xd, tb_x0);
    --    wait for 25 us;

    --------------------
    -- receive 1st data stage (DATA1)
    stim<= '1';
    packetIN   (tb_clk, tb_xd, tb_x0, addr0, ep0);
    stim<= '0';
    packetDATA1(tb_clk, tb_xd, tb_x0, dd_0_7);
    stim<= '1';
    packetACK  (tb_clk, tb_xd, tb_x0);

    --------------------
    -- receive 2nd data stage (DATA0)
    stim<= '1';
    packetIN   (tb_clk, tb_xd, tb_x0, addr0, ep0);
    stim<= '0';
    packetDATA0(tb_clk, tb_xd, tb_x0, dd_8_15);
    stim<= '1';
    packetACK  (tb_clk, tb_xd, tb_x0);

    --------------------
    -- data 3rd stage (DATA1)
    stim<= '1';
    packetIN   (tb_clk, tb_xd, tb_x0, addr0, ep0);
    stim<= '0';
    packetDATA1(tb_clk, tb_xd, tb_x0, dd_16_17);
    stim<= '1';
    packetACK  (tb_clk, tb_xd, tb_x0);

    --------------------
    -- NAKed status stages may be required here,
    -- until the microcontroller completed processing
    -- the USB request. these stages would look like this:
    --    stim<= '1';
    --    packetOUT  (tb_clk, tb_xd, tb_x0, addr0, ep0);
    --    packetDATA1(tb_clk, tb_xd, tb_x0);
    --    stim<= '0';
    --    packetNAK  (tb_clk, tb_xd, tb_x0);
    --    wait for 25 us;

    --------------------
    -- send status stage
    stim<= '1';
    packetOUT  (tb_clk, tb_xd, tb_x0, addr0, ep0);
    packetDATA1(tb_clk, tb_xd, tb_x0);
    stim<= '0';
    packetACK  (tb_clk, tb_xd, tb_x0);

    --------------------
    -- shut down test controller
    stim<= '0';
    sim <= '0';
    wait for 5 us;

    --------------------
    -- simulation ends here automatically,
    -- as no more clocks are created
    wait;
  end process;

end Test;

--------------------------------------------------------------------------------
-- end of file
```

References

- Universal Serial Bus Specification
  USB Implementers Forum
  http://www.usb.org

Revisions History

Table 2: Revisions History.

Version	Date	Who	Description
1.0	02mar25	FB	Initial version