```
--*************************************************************************************************
-- Product: EZ2SUSB
-- Filename: sample2.vhd
-- Date:     08-01-2003

-- EasyFPGA
-- 905 Shell Blvd.,#202N
-- Foster City, CA 94404
-- USA
-- support@easyfpga.com
-- If this code works then it was written by Peter Bolech. If not, I don't know who wrote it.
--*************************************************************************************************
library ieee;
use ieee.std_logic_1164.all;
use ieee.std_logic_unsigned.all;

entity sample2 is port (

  -- Clock signal and reset --
  LCLK:                 in std_logic;                       -- clock 10 MHz, on-board clock
  nRESET:               in std_logic;                       -- FPGA reset, on-board reset

  -- FT245BM USB FIFO interface --
  USB_D:                inout std_logic_vector(7 downto 0);  -- FT245BM data bus
  nUSB_RD:              inout std_logic;                    -- FT245BM read enable
  USB_WR:               inout std_logic;                    -- FT245BM write enable
  nUSB_TXE:             in std_logic;                       -- FT245BM transmit FIFO empty flag
  nUSB_RXF:             in std_logic;                       -- FT245BM receive FIFO full flag

  -- LEDs
  LED:                  out std_logic_vector(7 downto 0);    -- on-board LEDs

  -- I/O headers
  IO:                   inout std_logic_vector(31 downto 0); -- I/O pins

  -- EXP header
  EXT_CLOCK_IN0:        in std_logic;                       -- external clock/ user input pin
  EXT_CLOCK_IN1:        in std_logic;                       -- external clock/ user input pin
  EXT_CLOCK_IN2:        in std_logic;                       -- external clock/ user input pin
  EXP_IO:               inout std_logic_vector(24 downto 0)  -- I/O pins

   );
  end sample2;
```

```
architecture sample2_arch of sample2 is

type STATE_MACHINE_TYPE is (wait_for_rxf_low_0, usb_read_0, wait_one_cycle_0,
                            wait_for_rxf_low_1, usb_read_1, wait_one_cycle_1,
                            wait_for_rxf_low_2, usb_read_2, wait_one_cycle_2,
                            wait_for_rxf_low_3, usb_read_3, wait_one_cycle_3,
                            local_lad, local_data, wait_for_txe_low, usb_write,
                            wait_one_cycle_4);

signal STATE_MACHINE:    STATE_MACHINE_TYPE;
signal nUSB_TXE_LCH:     std_logic;
signal nUSB_RXF_LCH:     std_logic;
signal USB_D_LCH0:       std_logic_vector(7 downto 0);
signal USB_D_LCH1:       std_logic_vector(7 downto 0);
signal USB_D_LCH2:       std_logic_vector(7 downto 0);
signal USB_D_LCH3:       std_logic_vector(7 downto 0);

signal LAD_IN:           std_logic_vector(7 downto 0);
signal LAD_OUT:          std_logic_vector(7 downto 0);
signal LAD_INT:          std_logic_vector(7 downto 0);

signal nCSINT:           std_logic;
signal nLADS:            std_logic;
signal nLRW:          std_logic;
signal WAIT_COUNTER:     std_logic_vector(2 downto 0);
signal PROC_ADD:         std_logic_vector(7 downto 0);
signal nLRDY:            std_logic;

signal FPGA_REV:         std_logic_vector(7 downto 0);
signal LED_REG:          std_logic_vector(7 downto 0);

constant FPGA_REVISION:  std_logic_vector(7 downto 0) := "00000001";

constant READ_COMMAND:   std_logic_vector(7 downto 0) := "10101010";
constant CS_INT_ADD:     std_logic_vector(7 downto 0) := "00000000";

constant FPGA_REV_ADD:   std_logic_vector(7 downto 0) := "00000000";
constant LED_REG_ADD:    std_logic_vector(7 downto 0) := "00000001";


begin

IO <= (others => 'Z');
EXP_IO <= (others => 'Z');
```

```
--*************************************************************************************************************
-- Byte 0: command (read/write)
-- Byte 1: chip select (CS_INT = "00000000")
-- Byte 2: address
-- Byte 3: if write - data, if read - don't care

STATE_MACHINE_PR: process (LCLK, nRESET)
begin

if(LCLK'event and LCLK = '1') then
 if(nRESET = '0') then
  STATE_MACHINE <= wait_for_rxf_low_0;
 else
 case STATE_MACHINE is

-- USB read byte 0
   when wait_for_rxf_low_0 =>
    if(nUSB_RXF_LCH = '0') then
     STATE_MACHINE <= usb_read_0;
    end if;

   when usb_read_0 =>
    STATE_MACHINE <= wait_one_cycle_0;

   when wait_one_cycle_0 =>
    STATE_MACHINE <= wait_for_rxf_low_1;

-- USB read byte 1
   when wait_for_rxf_low_1 =>
    if(nUSB_RXF_LCH = '0') then
     STATE_MACHINE <= usb_read_1;
    end if;

   when usb_read_1 =>
    STATE_MACHINE <= wait_one_cycle_1;

   when wait_one_cycle_1 =>
    STATE_MACHINE <= wait_for_rxf_low_2;

-- USB read byte 2
   when wait_for_rxf_low_2 =>
    if(nUSB_RXF_LCH = '0') then
     STATE_MACHINE <= usb_read_2;
    end if;

   when usb_read_2 =>
    STATE_MACHINE <= wait_one_cycle_2;

   when wait_one_cycle_2 =>
    STATE_MACHINE <= wait_for_rxf_low_3;
```

```
-- USB read byte 3
   when wait_for_rxf_low_3 =>
    if(nUSB_RXF_LCH = '0') then
     STATE_MACHINE <= usb_read_3;
    end if;

   when usb_read_3 =>
    STATE_MACHINE <= wait_one_cycle_3;

   when wait_one_cycle_3 =>
    STATE_MACHINE <= local_lad;

-- local bus access
   when local_lad =>
    STATE_MACHINE <= local_data;

   when local_data =>
    if(nLRDY = '0') then
     if(nLRW = '0') then                      -- if local bus READ goto USB write
      STATE_MACHINE <= wait_for_txe_low;
     else
      STATE_MACHINE <= wait_for_rxf_low_0;    -- if loacal bus WRITE goto "idle"
     end if;
    end if;

-- USB write
   when wait_for_txe_low =>
    if(nUSB_TXE_LCH = '0') then
     STATE_MACHINE <= usb_write;
    end if;

   when usb_write =>
    STATE_MACHINE <= wait_one_cycle_4;

   when wait_one_cycle_4 =>
    STATE_MACHINE <= wait_for_rxf_low_0;     -- data from local bus written into the ftdi chip, goto "idle"

   when others =>
    STATE_MACHINE <= wait_for_rxf_low_0;

   end case;

  end if;
 end if;

end process STATE_MACHINE_PR;
```

```
--*************************************************************************************************************
-- nUSB_RD
USB_RD_PR: process(LCLK)
begin

if(LCLK'event and LCLK = '1') then
 if(STATE_MACHINE = wait_for_rxf_low_0 or STATE_MACHINE = wait_for_rxf_low_1 or
    STATE_MACHINE = wait_for_rxf_low_2 or STATE_MACHINE = wait_for_rxf_low_3) then
    if(nUSB_RXF_LCH = '0') then
      nUSB_RD <= '0';
     else
      nUSB_RD <= '1';
     end if;
  else
   nUSB_RD <= '1';
  end if;
end if;

end process USB_RD_PR;

--*************************************************************************************************************
-- USB_WR
USB_WR_PR: process(LCLK)
begin

if(LCLK'event and LCLK = '1') then
 if(STATE_MACHINE = wait_for_txe_low) then
  if(nUSB_TXE_LCH = '0') then
   USB_WR <= '1';
  else
   USB_WR <= '0';
  end if;
 else
  USB_WR <= '0';
 end if;
end if;

end process USB_WR_PR;

--*************************************************************************************************************
-- sync. USB flags with local clock
USB_FLAGS_PR: process(LCLK, nUSB_TXE, nUSB_RXF)
begin

if(LCLK'event and LCLK = '1') then
 nUSB_TXE_LCH <= nUSB_TXE;
end if;

if(LCLK'event and LCLK = '1') then
 nUSB_RXF_LCH <= nUSB_RXF;
end if;
```

```
end process USB_FLAGS_PR;

--*************************************************************************************************************
-- read data from the USB chip
USB_BUS_PR: process (LCLK)
begin

if(LCLK'event and LCLK = '1') then
 if(nUSB_RD = '0') then
  USB_D_LCH0 <= USB_D;
  USB_D_LCH1 <= USB_D_LCH0;
  USB_D_LCH2 <= USB_D_LCH1;
  USB_D_LCH3 <= USB_D_LCH2;
 end if;
end if;

if(USB_WR = '1') then
 USB_D <= LAD_IN;
else
 USB_D <= (others => 'Z');
end if;

end process USB_BUS_PR;

--*************************************************************************************************************
-- local bus
LOCAL_BUS_LAD_PR: process (LCLK, nLRW, nLRDY)
begin

if(LCLK'event and LCLK = '1') then
 if(STATE_MACHINE = local_lad) then
  LAD_OUT <= USB_D_LCH1;   -- address
 else
  LAD_OUT <= USB_D_LCH0;   -- data
 end if;
end if;

-- read data from internal registers
if(LCLK'event and LCLK = '1') then
 if(STATE_MACHINE = local_data and nLRW = '0' and nLRDY = '0') then
  LAD_IN <= LAD_INT;
 end if;
end if;

end process LOCAL_BUS_LAD_PR;
```

```
--*************************************************************************************************************
-- local bus address strobe
LOCAL_BUS_ADS_PR: process (LCLK)
begin

if(LCLK'event and LCLK = '1') then
 if(STATE_MACHINE = local_lad) then
  nLADS <= '0';
 else
  nLADS <= '1';
 end if;
end if;

end process LOCAL_BUS_ADS_PR;

--*************************************************************************************************************
-- local bus chip-select
LOCAL_BUS_CS_PR: process (LCLK)
begin

-- chip-select for internal registers
if(LCLK'event and LCLK = '1') then
 if(STATE_MACHINE = local_data and USB_D_LCH2 = CS_INT_ADD) then
  nCSINT <= '0';
 else
  nCSINT <= '1';
 end if;
end if;

end process LOCAL_BUS_CS_PR;

--*************************************************************************************************************
-- AA - read command, 55 - write command
-- local bus nRW signal
LOCAL_BUS_RW_PR: process (LCLK)
begin

if(LCLK'event and LCLK = '1') then
 if(STATE_MACHINE = local_data and USB_D_LCH3 = READ_COMMAND) then
  nLRW <= '0';
 else
  nLRW <= '1';
 end if;
end if;

end process LOCAL_BUS_RW_PR;
```

```
--*************************************************************************************************************
-- internal registers access
--
INTERNAL_REG_PR: process (LCLK)
begin

-- local bus address latch
if(LCLK'event and LCLK = '1') then
 if(nLADS = '0') then
  PROC_ADD <= LAD_OUT;
 end if;
end if;

-- local bus wait states
if(LCLK'event and LCLK = '1') then
 if(nCSINT = '1') then
  WAIT_COUNTER <= "000";
 else
  WAIT_COUNTER <= WAIT_COUNTER + '1';
 end if;
end if;

-- local bus ready
if(LCLK'event and LCLK = '1') then
 if(nCSINT = '0' and PROC_ADD(7 downto 0) >= "00000000" and
    PROC_ADD <= "11111111" and WAIT_COUNTER = "100"
   ) then
   nLRDY <= '0';
 else
   nLRDY <= '1';
 end if;
end if;

-- local registers: write
if(LCLK'event and LCLK = '1') then
 if(nRESET = '0') then
  LED_REG <= (others => '0');
 elsif(nLRW = '1' and nCSINT = '0' and WAIT_COUNTER = "011") then
  if(PROC_ADD = LED_REG_ADD) then
   LED_REG <= LAD_OUT;                                  -- LED register (R/W)
  end if;
 end if;
end if;
```

```
--local registers: read
if(PROC_ADD = FPGA_REV_ADD) then
 LAD_INT <= FPGA_REVISION;                          -- FPGA revision number register (R/O)
elsif(PROC_ADD = LED_REG_ADD) then
 LAD_INT <= LED_REG;                                -- LED register (R/W)
else
 LAD_INT <= (others => '0');
end if;

LED <= LED_REG;

end process INTERNAL_REG_PR;

end sample2_arch;
```