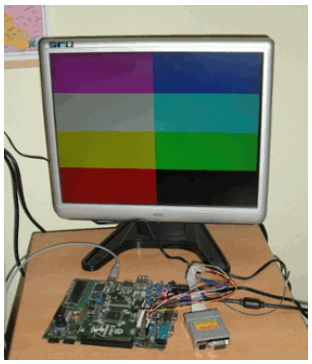
Simulieren der Unisim Library mit Hilfe des Simulator GHDL

René Doß
http://www.dossmatik.de

22. Januar 2010

GHDL ist ein frei verfügbarer Simulator, der in der Entwicklung einsatzfähig ist. Das Tool ist mitlerweile sehr mächtig geworden, und eignet sich, um Fehler im Design rechtzeitig zu erkennen. Er kann die Xilinx spezifische Library unisim auch simulieren. Wie die Einbindung der Library erfolgt, soll in diesem Dokument näher erklärt werden. Anhand des Beispiel eines VGA Monitors wird die Taktvervielfachung DCM simuliert. Der VGA Port ist an den meisten Entwicklungsboard enthalten, so kann das Beispiel an existierender Hardware nachvollzogen werden. Zum Kennenlernen von GHDL wird zuerst ein Beispiel ohne herstellerspezifischen Komponenten simuliert. Anschließend ein Beispiel mit einer DCM zur Taktvervielfachung.



Drei Sachen sollten funktionsfähig auf dem PC sein, um die Simulation nach zu vollziehen.

GHDL http://ghdl.free.fr/

ISE 11.4 http://www.xilinx.com/

Gtkwave http://gtkwave.sourceforge.net/

1 Der VGA-Port mit einer Auflösung von 640x480

Der zur Verfügung stehende Eingangstakt von 50MHz wird durch ein Flip-Flop auf 25MHz halbiert. Das ist der Pixelclock. Im Takt wird der Spaltenzähler von 0 bis 799 erhöht. Bei einem Übertrag wird der Zeilenzähler um eins erhöht. Von den beiden Zählerständen werden die Synchronisationssignale erzeugt und auf den Farbkanälen ein Testbild erzeugt.

Der VHDL-Code

Der folgende Code erzeugt auf einem VGA Monitor ein Streifenmuster und sollte mit dem Dateinamen vga640_480.vhd gespeichert werden.

```
----------------------------------
-- Dossmatik GmbH
-- http://www.dossmatik.de
--simple VGA example

-- Additional Comments:
--
--Spartan3AN board
-- Pin Assignment:
-- NET clk50_in loc = T9
-- NET red_out LOC=R12;
-- NET green_out LOC=T12;
-- NET blue_out LOC=R11;
-- NET hs_out LOC=R9;
-- NET vs_out LOC=T10;
----------------------------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use ieee.numeric_std.ALL;

entity vga_timing is
port(
clk50_in : in std_logic;

red_out   : out std_logic;
green_out : out std_logic;
blue_out  : out std_logic;
hs_out : out std_logic;
vs_out : out std_logic);
end vga_timing;

architecture behavioral of vga_timing is
```

```
signal clk25 : std_logic:='0';
signal hcounter : unsigned (9 downto 0):=to_unsigned(0,10);
signal vcounter : unsigned (9 downto 0):=to_unsigned(0,10);

begin

-- generate a 25Mhz clock
process (clk50_in)
begin
if clk50_in'event and clk50_in='1' then
    clk25 <= not clk25;
end if;
end process;

process (clk25)
begin
if clk25'event and clk25 = '1' then
        if vcounter<480 then
            if hcounter < 320 then
                  red_out <='1';
             else
                red_out <='0';
            end if;
        else
            red_out <='0';
        end if;
end if;
end process;


process (clk25)
begin
if clk25'event and clk25 = '1' then
        if hcounter<640 then
            if vcounter < 240 then
                  blue_out <='1';
            else
                  blue_out <='0';
            end if;
            if vcounter > 120 and vcounter < 360 then
                  green_out <='1';
            else
                  green_out <='0';
            end if;
```

```
        else
            blue_out <='0';
            green_out <='0';
        end if;
end if;
end process;


--sync signal generation
process (clk25)
begin

if clk25'event and clk25 = '1' then

        if hcounter >= (639+16) and hcounter <= (639+16+96) then
                hs_out <= '0';
        else
                hs_out <= '1';
        end if;
        if vcounter >= (479+10) and vcounter <= (479+10+2) then
                vs_out <= '0';
        else
                vs_out <= '1';
        end if;
-- horizontal counts from 0 to 799
        hcounter <= hcounter+1;
        if hcounter = 799 then
                vcounter <= vcounter+1;
                hcounter <= to_unsigned(0,10);
        end if;
-- vertical counts from 0 to 524
        if vcounter = 524 then
                vcounter <= to_unsigned(0,10);
        end if;
end if;
end process;

end behavioral;
```

Testbench

Die dazugehörige Testbench liefert eine Stimulation des Signals clk50_in. Mit diesem Takt werden die Counter zur Steuerung des Bildsignales angeregt.

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;

ENTITY tb_vga IS
END tb_vga;

ARCHITECTURE behavior OF tb_vga IS

    -- Component Declaration for the Unit Under Test (UUT)

    COMPONENT vga_timing
    PORT(
         clk50_in  : IN  std_logic;
         red_out   : OUT  std_logic;
         green_out : OUT  std_logic;
         blue_out  : OUT  std_logic;
         hs_out    : OUT  std_logic;
         vs_out    : OUT  std_logic
        );
    END COMPONENT;


   --Inputs
   signal clk50_in : std_logic := '0';

 --Outputs

   signal red_out : std_logic;
   signal green_out : std_logic;
   signal blue_out : std_logic;
   signal hs_out : std_logic;
   signal vs_out : std_logic;

   -- Clock period definitions
   constant clk50_in_period : time := 10 ns;

BEGIN

        -- Instantiate the Unit Under Test (UUT)
   uut: vga_timing PORT MAP (
          clk50_in => clk50_in,
          red_out => red_out,
          green_out => green_out,
```

```
            blue_out => blue_out,
            hs_out => hs_out,
            vs_out => vs_out
        );

    -- Clock process definitions
    clk50_in_process :process
    begin
                clk50_in <= '0';
                wait for clk50_in_period/2;
                clk50_in <= '1';
                wait for clk50_in_period/2;
    end process;


END;
```

GHDL ist ein Kommandozeilentool und die Simulation wird durch Optionen gesteuert. Jede Datei muss zuerst analysiert werden. Falls ein Fehler im VHDL Code erkannt wird, gibt GHDL eine Fehlermeldung mit einer Lokalisierung aus. Aufruf mit der Option -a für analysing.

```
ghdl -a vga640_480.vhd
```

```
ghdl -a tb_vga.vhd
```

Danach wird eine ausführbare Datei erstellt. Option -e für executable mit dem top entity port.

```
ghdl -e tb_vga
```

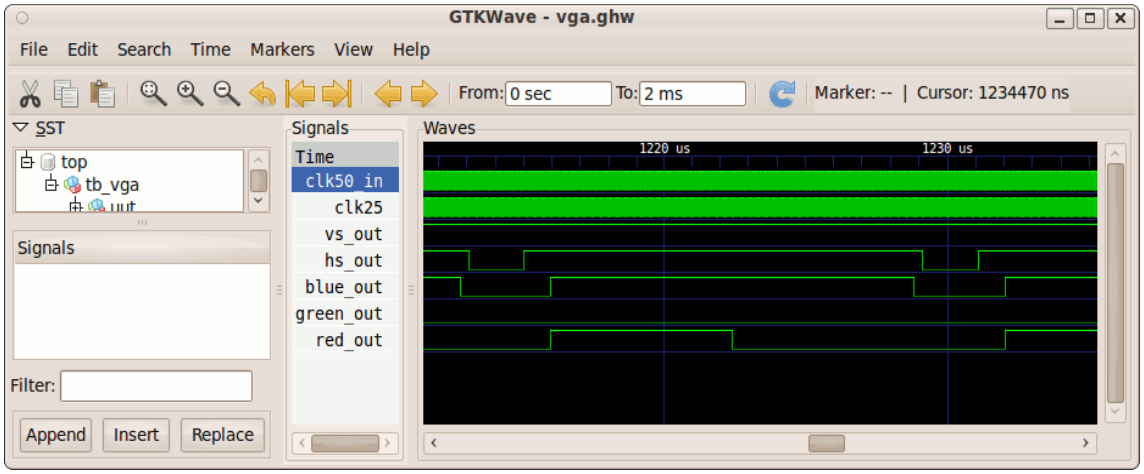
Diese ausführbare Datei beinhaltet das Verhalten der Verdrahung der Elemente, wie sie im VHDL-Code geschrieben ist. Für eine bestimmte Zeit wird simuliert und in die Datei vga.ghw als Ausgabe geschrieben. Hier steht die Option -r für run.

```
ghdl -r tb_vga --stop-time=2000000ns -wave=vga.ghw
```

Die simulierten Signalverläufe sind nun in der Datei vga.ghw gespeichert und können mit dem Programm gtkwave betrachtet werden.

```
gtkwave vga.ghw
```

Die zu begutachtenden Signale sind aus dem Design auszuwählen und zu aktivieren. Danach sind die Signalverläufe im Taktdigagramm sichtbar.



2 Der VGA-Port mit einer Auflösung von 1024x768

Bei dieser Auflösung ist es nicht so einfach, einen Pixeltakt von 75MHz mit einem 50MHz Eingangstakt zu erzeugen. Der folgende Code erzeugt einen Farbwechsel auf VGA Monitor und sollte mit dem Dateinamen vga1024_768.vhd gespeichert werden. Für die Simulation wird die gleiche Testbench genutzt.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

--use ieee.numeric_std.ALL;
library UNISIM;
use UNISIM.Vcomponents.ALL;


entity vga_timing is
port(
    clk50_in     : in std_logic;
    red_out      : out std_logic;
    green_out    : out std_logic;
    blue_out     : out std_logic;
    hs_out       : out std_logic;
    vs_out       : out std_logic);
```

```
end vga_timing;

architecture behavioral of vga_timing is

signal clk75            : std_logic;
signal CLK0_BUF         : std_logic;
signal CLKFB_IN         : std_logic;
signal hcounter         : integer range 0 to 1328;
signal vcounter         : integer range 0 to 806;
signal color            : std_logic_vector(2 downto 0):="111";
signal CLKIN_IBUFG      : std_logic;
signal CLKFX_BUF        : std_logic;
signal GND_BIT          : std_logic;


begin

   GND_BIT <= '0';

   CLKIN_IBUFG_INST : IBUFG
      port map (I=>CLK50_in,
                O=>CLKIN_IBUFG);

   CLK0_BUFG_INST : BUFG
      port map (I=>CLK0_BUF,
                O=>CLKFB_IN);



   CLKFX_BUFG_INST : BUFG
      port map (I=>CLKFX_BUF,
                O=>CLK75);


  DCM_SP_INST : DCM_SP
   generic map( CLK_FEEDBACK => "1X",
            CLKDV_DIVIDE => 2.0,
            CLKFX_DIVIDE => 2,
            CLKFX_MULTIPLY => 3,
            CLKIN_DIVIDE_BY_2 => FALSE,
            CLKIN_PERIOD => 20.000,
            CLKOUT_PHASE_SHIFT => "NONE",
            DESKEW_ADJUST => "SYSTEM_SYNCHRONOUS",
            DFS_FREQUENCY_MODE => "LOW",
            DLL_FREQUENCY_MODE => "LOW",
```

```
            DUTY_CYCLE_CORRECTION => TRUE,
            FACTORY_JF => x"C080",
            PHASE_SHIFT => 0,
            STARTUP_WAIT => FALSE)
        port map (CLKFB=>CLKFB_IN,
                  CLKIN=>CLKIN_IBUFG,
                  DSSEN=>GND_BIT,
                  PSCLK=>GND_BIT,
                  PSEN=>GND_BIT,
                  PSINCDEC=>GND_BIT,
                  RST=>'0',
                  CLKDV=>open,
                  CLKFX=>CLKFX_BUF,
                  CLKFX180=>open,
                  CLK0=>CLK0_BUF,
                  CLK2X=>open,
                  CLK2X180=>open,
                  CLK90=>open,
                  CLK180=>open,
                  CLK270=>open,
                  LOCKED=>open,
                  PSDONE=>open,
                  STATUS=>open);
```

```
-- change color
p1: process (clk75)
variable cnt: integer;
begin
if clk75'event and clk75='1' then
cnt := cnt + 1;
if cnt = 25000000 then
color <= color + "001";
cnt := 0;
end if;
end if;
end process;

p2: process (clk75, hcounter, vcounter)
variable x: integer range 0 to 2000:=0;
variable y: integer range 0 to 2000:=0;
begin
x := hcounter ;
```

```
y := vcounter ;
if clk75'event and clk75 = '1' then
    if x < 1023 and y < 767 then
      red_out <= color(0);
      green_out <= color(1);
      blue_out <= color(2);
    else
    -- if not traced, set it to "black" color
    red_out <= '0';
    green_out <= '0';
    blue_out <= '0';
    end if;

    if hcounter > 1047 and hcounter < 1185 then
        hs_out <= '0';
    else
        hs_out <= '1';
    end if;

    if vcounter > 770 and vcounter < 778 then
        vs_out <= '0';
    else
        vs_out <= '1';
    end if;
        hcounter <= hcounter+1;
    if hcounter = 1238 then
        vcounter <= vcounter+1;
        hcounter <= 0;
    end if;
--
    if vcounter = 806 then
        vcounter <= 0;
    end if;
end if;
end process;

end behavioral;
```

Bei größeren Designs lohnt es sich, das Design vom GHDL verwalten zu lassen. Es bestehen Abhängigkeiten in der Struktur und da ist die Reihenfolge der Dateien wichtig. Es lässt sich wie folgt automatisieren. Dafür sind die VHDL Files zu erst zu importieren (Option -i).

```
ghdl -i --work=unisim /opt/Xilinx/11.1/ISE/vhdl/src/unisims/*.vhd
ghdl -i --work=unisim /opt/Xilinx/11.1/ISE/vhdl/src/unisims/primitive/*.vhd
ghdl -i *.vhd
```

Anschließend die ausführbare Datei von GHDL mit der Option -m für make erstellen.

```
ghdl -m -g -Punisim --warn-unused --ieee=synopsys tb_vga
```

Jetzt ist Simulation wie beim ersten Beispiel zu starten und die Ausgabe danach in gtkwave darstellbar.

```
 ghdl -r tb_vga  --disp-tree=inst --stop-time=20000ns --wave=vga.ghw
tb_vga [entity]
‘-behavior [arch]
  ‘-uut [instance]
    ‘-vga_timing [entity]
      ‘-behavioral [arch]
        +-clkin_ibufg_inst [instance]
        | ‘-ibufg [entity]
        |   ‘-ibufg_v [arch]
        +-clk0_bufg_inst [instance]
        | ‘-bufg [entity]
        |   ‘-bufg_v [arch]

......

./tb_vga:info: simulation stopped by --stop-time

 gtkwave vga.ghw
```