The Brainf*ck CPU Project

An Introduction to FPGA Development using VHDL

http://www.clifford.at/bfcpu/

Clifford Wolf

ROCK Linux - http://www.rocklinux.org

CNGW - http://www.cngw.orig

LINBIT - http://www.linbit.com
Overview

- Building custom hardware
- The Brainf*ck CPU

Programmable Hardware

Introduction to VHDL
Building custom hardware

With programmable hardware (such as PLDs and FPGAs) it is possible to

- Instantly test hardware designs with almost no prototyping costs
- Simply "upload" hardware designs (bitstream files) to a chip like it would be software
- Have much fun with building hardware without touching a soldering gun

With HDLs (such as Verilog and VHDL) it is possible to

- Describe a hardware design like program source describes the behavior of a program
- Automatically create bitstream files for different chips from the same (portable) source
- Simulate the behavior of your hardware on various levels
The Brainf*ck CPU

A minimalistic CPU with 8 bit data-bus and 16 bit address-bus which can execute brainf*ck code. The VHDL design file for it has 260 lines of code. The optimised variation of the CPU with a 1 byte internal data cache has 340 lines of code.
Programmable Hardware

Overview

Programmable Hardware
- PALs
- PLDs and CPLDs
- FPGAs
- Recommended reading

Introduction to VHDL
PALs

A PAL is a very simple and limited programmable hardware device:

It’s programm only consists of a single truth table. Which is implemented as one big unit with d-flipflops (connected to one global clock) on the output lines.
A PLD consists of macrocells which are something like better PALs and one central interconnection logic. PLDs are usually used to connect a high number of pins with a more or less simple logic at very low cost. Typical PLD applications are "glue logic" for connecting other ASICs.
FPGAs

FPGAs consist of even more complex logic blocks connected by a very complex interconnection matrix. FPGAs can implement a very complex logic on one chip. Typical applications are CPUs and DSPs up to very complex SoC setups.
Recommended reading

The Xilinx "Detailed Functional Description" Datasheets

E.g.: "Spartan-IIE 1.8V FPGA Detailed Functional Description"
Introduction to VHDL
What is VHDL

- VHDL is the "VHSIC Hardware Description Language"

- VHSIC is a "Very High Speed Integrated Circuit"

- VHDL was originally design to document circuits

- Later on, programs have been develop to generate ciruct designs from VHDL code

- Only a small subset of correct VHDL code can be used to synthesize designs
A 2 bit counter (1)

A simple clocked 2 bit counter (00, 01, 10, 11, 00, ..):

- 2 Output signals: D1 (lower bit) and D2 (higher bit)

- The following truth table shows how to calculate new D1 and D2 from the old values:

<table>
<thead>
<tr>
<th>D2</th>
<th>D1</th>
<th>D2, D1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 1</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>D2</th>
<th>D1</th>
<th>D2, D1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0</td>
<td>0 0</td>
<td></td>
</tr>
<tr>
<td>0 1</td>
<td>1 1</td>
<td></td>
</tr>
<tr>
<td>1 0</td>
<td>1 0</td>
<td></td>
</tr>
<tr>
<td>1 1</td>
<td>0 0</td>
<td></td>
</tr>
</tbody>
</table>

So it turns out: D2 xor D1 not D1
A 2 bit counter (2)

The 2 bit counter as circuit:

And this is the VHDL code for the same thing:

```vhdl
process (clock) begin
    if rising_edge(clock) then
        output1 <= not output1;
        output2 <= output1 xor output2;
    end if;
end process;
```
Signals in VHDL

- Signals are single wires or buses in the circuit.

- Buses (multiple wires with one name) are usually called "signal vectors".

- There are multiple signal types (vitaly important: bit and std_logic).

- Usually a signal my only have one single source (exception: std_ulogic).

- Many signal types have predefined operations (like "xor", "not", "+" or "*").

- Signals can be assigned a value using the "<=" operator.
Prozesses in VHDL (1)

- Processes are one way to group VHDL statements to a logical unit.

- A dependency list contains all signals the statements in the process depend on.

```vhdl
process (mysignal2, mysignal3) begin
  mysignal1 <= mysignal2 xor mysignal3;
end process;
```

- Clocked processes are the preferred way to use d-flip-flops in the design:

```vhdl
process (clk) begin
  if rising_edge(clk) then
    mysignal1 <= mysignal2 xor mysignal3;
  end if;
end process;
```
Processes in VHDL (2)

- Clocked processes may also contain code for reset signals:

```
process (clk, rst) begin
  if rst = '1' then
    mysignal1 <= x"00";
  elsif rising_edge(clk) then
    mysignal1 <= mysignal1 + 1;
  end if;
end process;
```

- Remember: Only a small subset of syntactically and grammatically correct code can actually used to synthesize a real hardware design.
- No more than one "if rising_edge(clk)" per process.
- That "if" must surround all other instructions in the process.
- The only allowed variation is the check for a reset signal.
Processes in VHDL (3)

Processes may always contain code which implies a feedback of outputs. The first code fragment implies the else-tree of the 2nd one:

```vhdl
process (clk) begin
  if rising_edge(clk) then
    if enable = 1 then
      mysignal1 <= mysignal1 + 1;
      end if;
    end if;
  end process;

process (clk) begin
  if rising_edge(clk) then
    if enable = 1 then
      mysignal1 <= mysignal1 + 1;
      end if;
    else
      mysignal1 <= mysignal1;
      end if;
  end process;
```
Variables in VHDL

- Variables may be used just as variables in a traditional program.
- Variables are assigned values using the ":=" operator.
- Variables are always local to a process.

```vhdl
process (clk)
begin
    if rising_edge(clk) then
        temp := mysignal1;
        temp := temp + 25;
        temp := temp * mysignal2;
        if mysignal3 = '0' then
            temp := mysignal2;
        end if;
        mysignal1 <= temp;
        temp := temp + 844;
        mysignal2 <= temp;
    end if;
end process;
```
Entities, archs and components

- A VHDL project is structured in so-called entities.

- An "architecture" is the concrete implementation of an entity.

- An entity may have multiple architectures (e.g. optimized for size or speed).

- When an entity is used in another entity, it is called a component.

- Components and signal types can be imported from libraries.
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity demo is
  port( 
      clk : in std_logic;
      rst : in std_logic;
      dat : out std_logic_vector (15 downto 0)
  );
end demo;
architecture demo_arch of demo is
  signal val : std_logic_vector (15 downto 0);
begin
  process (clk, rst) begin
    if rst = '1' then
      val <= (others => '0');
    elsif rising_edge(clk) then
      val <= val + 1;
    end if;
  end process;
  dat <= val;
end demo_arch;
The Brainf*ck CPU

[ Discussion the Brainf*ck CPU VHDL code ]
Credits

- LINBIT Information Technologies GmbH: http://www.linbit.com/

- The ROCK Linux Project: http://www.rocklinux.org/

- Clifford Wolf: http://www.clifford.at/

http://www.clifford.at/bfcpu/