DE NAYER Instituut
J. De Nayerlaan 5
B-2860 Sint-Katelijne-Waver
Tel. (015) 31 69 44
Fax. (015) 31 74 53
e-mail: ppe@denayer.wenk.be
ddr@denayer.wenk.be
tti@denayer.wenk.be
website: emsys.denayer.wenk.be

Basic Custom OpenRISC System Hardware Tutorial

Xilinx

Version 1.00

HOBU-Fund
Project IWT 020079

Title	:	Embedded systemdesign based upon Soft- and Hardcore FPGA's
Projectleader	:	Ing. Patrick Pelgrims
Projectassistants	:	Ing. Dries Driessens Ing. Tom Tierens

I Introduction

Purpose of this tutorial is to help you compose and implement a custom, OpenRISC based, embedded system in the easiest way possible. Unexperienced users should be warned that the OpenRISC processor is quite a difficult processor : the lack of a self-configuring embedded system-package and the more 'open' nature of the OpenRISC compared to other open-source processors like the Leon SPARC is one of the reasons for this.

Before proceeding, check if you have the following software and hardware:

Hardware:
- Linux-PC or Windows-PC
- Development board with Xilinx FPGA (minimum 3100 Slices), UART and 6 available pins.

Software:
- OpenRISC-GNU Toolchain
- Xilinx ISE Webpack or ISE for Windows or Linux
- For Windows : WinCVS 1.2 (http://prdownloads.sourceforge.net/cvsgui/WinCvs120.zip)

If you experience problems building the OpenRISC-GNU Toolchain, there is also an OpenRISC Software tutorial available from our website (http://emsys.denayer.wenk.be).

The flow of implementing a custom, OpenRISC based, embedded system is:
A. retreive OpenRISC Platform HDL source-code
B. remove unnecessary components from source-code
C. adjust RAM module
D. synthesize and place & route OpenRISC based system
E. download
F. test the OpenRISC system

II Retreive Source Code

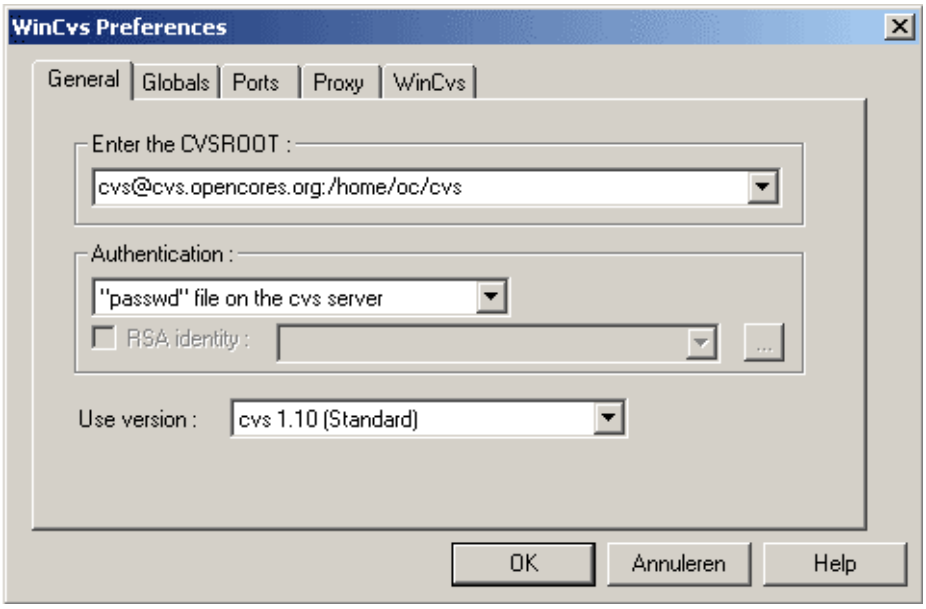
The OpenRISC Platform source code is available through a CVS server. To make sure that this tutorial doesn't get obsolete, the source code that will be downloaded is the same version that was used writing this tutorial.

IN WINDOWS:

Windows doesn't come with a cvs-client, so if you haven't installed wincvs yet, you need to download the CVS client WinCVS 1.2 (http://prdownloads.sourceforge.net/cvsgui/WinCvs120.zip). After downloading, install and open it.

First, configure WinCVS:
- goto menu item 'Admin' and click on 'Preferences'
- 'CVSROOT' should be cvs@cvs.opencores.org:/home/oc/cvs
- 'Authentication" needs to be "passwd file on the cvs server"
- disable "Checkout read-only" at the 'Globals' tab.

Figure 1: WinCVS settings



Next, goto the right location where you want to put the data downloaded from CVS: goto menu item 'View', then "browse location", click 'change' and choose your location.

Then you should login: goto menu item 'Admin' and click on 'Login'. When you're promted for a password, you can type anything.

Finally you can download the OpenRISC source code. Goto menu item 'Admin' and click on "Command Line...". Now type "cvs –z9 co –D 1/1/04 or1k/orp/orp_soc/rtl" in the command box.

To finish you just have to logout: goto menu item 'Admin' and click on 'Logout'

IN LINUX:

To begin, open a terminal so you have a prompt.

First, set the correct CVSROOT: type "export CVSROOT=:pserver:cvs@cvs.opencores.org:/home/oc/cvs"

Then go to the directory where you want to place the downloaded OR1K source code.

Next, login by typing "cvs login". When promted for a password, press enter.

Finally you can download the source code: just type "cvs –z9 co –D 1/1/04 or1k/orp/orp_soc/rtl"

To finish, just log out, by typing "cvs logout".

III Adjust Source Code

After retrieving the source code, comes the more complex part: adjusting the source code.
Keep in mind that we want to build a "basic custom OpenRISC system":

- 'Basic' because we only use the very minimum of peripherals: a debug-unit (for input), onchip-RAM (for processing) and a UART (for output).
- 'Custom' because this tutorial isn't targeted for one or a few boards specifically. Everything is kept general so that anybody can implement the OpenRISC on his "Xilinx FPGA"-board.
- 'OpenRISC' because of the processor used.
- System': because the system we're building contains all the necessary components of an embedded hardware system (input, processing and output components).

1) Delete unnecessary files

In the 'Verilog' directory remove:

- the following directories: 'audio', 'ethernet', 'ethernet.old', 'or1200.old', 'ps2', 'ps2.old', 'svga' and 'uart16550.old'
- the file 'tdm_slave_if.v'.

2) Adjust 'xsv_fpga_top.v'

a. 'module xsv_fpga_top' part:

There are 4 groups of signals that are necessary (So if they're not present add them):

- 2 global signals (clk, rstn),
- 2 uart signals (uart_stx, uart_srx),
- 7 jtag debug signals (jtag_tvref, jtag_tgnd, jtag_tck, jtag_tms, jtag_trst, jtag_tdi, jtag_tdo),
- and chip enable signals to shut down any unused electronic ICs of your custom board.

All other signals can be removed.

b. 'input and output' list:

For every signal that was used in the module part, you should define if it is either an input or an output:

- **7 inputs** : clk, rstn, jtag_tck, jtag_tms, jtag_tdi, jtag_trst, uart_srx
- **4 outputs** : jtag_tvref, jtag_tgnd, jtag_tdo, uart_stx

c. 'internal wires' list:

The following wires can remain: debug core wires, debug<->risc wires, RISC instruction & data master wires, "SRAM controller slave i/f" wires (for the onchip RAM), UART16550 wires, UART external wires, JTAG wires and your custom chip enable wires (so that you can assign vcc or gnd). When using a clock-DLL to adjust the clock, you'll need 3 additional wires: 2 in and out wires for the clock-feedback signal and 1 for the DLL output.

d. 'assign' part:

In the following part of the 'xsv_fpga_top.v' file, insert an interface for a DLL-module if the clock speed is too high or too low. The following example is for a 10 x clock-divider.

Replace : “ibufg ibug1 (
.o (wb_clk),
.i (clk));”
With:

```
CLKDLL clkdiv (.CLKIN(clk),
        .CLKFB(clock_feedback_input),
        .RST(1'b0),
        .CLK0(clock_feedback_output),
        .CLK90(),
        .CLK180(),
        .CLK270(),
        .CLK2X(),
        .CLKDV(dll_output),
        .LOCKED());

// synthesis attribute CLKDV_DIVIDE of clkdiv is "10.0"

BUFG   clkg1   (.I(clock_feedback_output), .O(clock_feedback_input));
BUFG   clkg2   (.I(dll_output), .O(wb_clk));
```

ⓘ As mentioned before, do not forget to add the necessary ‘wires’ that you used in your clkdiv!

Remove “SRAM tri-state data”, “Ethernet tri-state”, “PS/2 Keyboard Tristate”, “Unused interrupts” and “Unused WISHBONE signals” (except for “assign wb_us_err_o = 1’b0;”)

ⓘ Also do not forget to remove the “RISC Instruction address for Flash” part.

You can now add the necessary assignments for the chip enable signals and **jtag_tvref**/**jtag_tgnd**. Every assignment looks like “assign signal_to_be_assigned = 1’b0;” (or “1’b1;”)

e. ‘instantiations’ part:

Remove the following instantiations: “VGA CRT controller”, “Audio controller”, “Flash controller”, “Ethernet 10/100 MAC”, “PS/2 Keyboard Controller”, “CPLD TDM”.

The following adjustments have to be made into the remaining instantiations:
- change the ‘clk_i’ of ‘or1200_top’ into the divided clock ‘wb_clk’.
- the last connection ‘pic_ints’ of the ‘or1200_top’ instantiation should be replaced by “20’b0”
- “sram_top sram_top” should be replaced by “onchip_ram_top onchip_ram_top”
- all the external SRAM connections should be removed (‘// SRAM external’ part of ‘sram_top’)
- in the ‘uart_top’ instantiation the “.int_o (pic_ints[`APP_INT_UART])” must be replaced by “.int_o ()”
- in the Traffic cop instantiation, the following changes should be made:
  - MASTERS: Wishbone Initiators 0, 1 and 2 must be replaced by stubs (like Initiators 6 and 7)
  - SLAVES: Wishbone Targets 1, 2, 3, 4 and 6 must be replaced by stubs (like Targets 7, 8)
  - “.i4_wb_ack_o (wb_rdm_ack)” must be “.i4_wb_ack_o (wb_rdm_ack**_i**)”
  - and “.i5_wb_adr_i (wb_ri**f**_adr)” must be “.i5_wb_adr_i (wb_ri**m**_adr**_o**)”

3) Adjust ‘or1200_defines.v’

Several defines should be enabled or disabled:
- Enable “`define OR1200_NO_DC”, “`define OR1200_NO_IC”, “`define OR1200_NO_DMMU” and “`define OR1200_NO_IMMU”. These defines are defined double in the file, once for “ifdef OR1200_ASIC” and one for the “else” (ie FPGA). Of course the latter is the one of importance.
- Disable “`define OR1200_CLKDIV_2_SUPPORTED”

4) Adjust ‘or1200_cpu.v’

Remove “.genpc_stop_prefetch (genpc_stop_prefetch)” from the ‘or1200_genpc’ component.

5) Adjust ‘or1200_sprs.v’

Remove “ ‘OR1200_SR_EPH_DEF,” (line 367) and on the same codeline change WIDTH-3 into WIDTH-2.

IV Add new components

onchip RAM component

To generate the “onchip_ram” module, there are 2 possibilities:

- either you write your own module composing standard Xilinx memory blocks
- either you generate a module with CORE Generator.

The latter is the easiest way and here comes how you accomplish this:

1) open Xilinx CORE Generator (*Start → programs → Xilinx ISE … → Accessories → CORE Generator*)

2) Press the “Create a New Project” button.

3) Choose a directory where to place the core. (preferably in a sub-directory of your source directory)

4) Select “Flow Vendor”, “Verilog” and “Other”. Also select the correct “Target Architecture”.

5) Now you can select what core to generate: select “Memories & Storage Elements”, then select “RAMs & ROMs”. Now you can double click on “Single Port Block Memory”.

6) In this wizard, type in the component name “onchip_ram”, adjust the width (8) and the Depth (1024 or a multitude). Then press “Next>” (Figure 2)

7) Enable the “Enable Pin” option after which you can click “Generate”. To exit, click “Dismiss” and close “CORE Generator”.

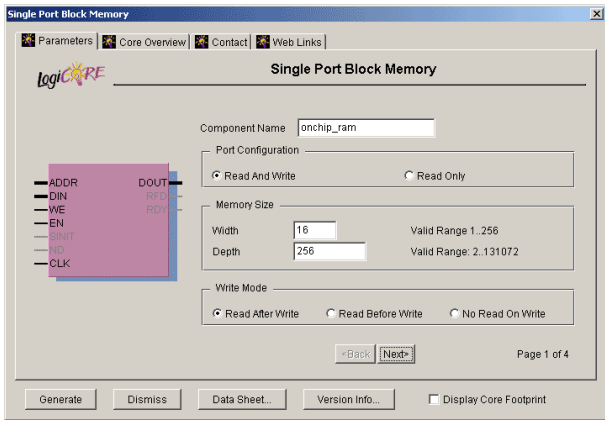


Figure 2: CORE Generator wizard

Now comes an important step: writing the onchip RAM component. A completely new component has to be written. The module consists of:

- a ‘module’ part
- a ‘parameter’ part
- an ‘input & output’ part
- a ‘wire’ part
- an ‘assignment’ part
- ‘read and write acknowledge processes’ part
- ‘altsyncram instantiation’ part

When taking in account timing and protocol of Wishbone bus and synchronous onchip RAM module (***onchip_ram_top.v***) eventually must look like beneath. The example is a 16kB or 131072 bits onchip RAM version:

```
module onchip_ram_top (
  wb_clk_i, wb_rst_i,

  wb_dat_i, wb_dat_o, wb_adr_i, wb_sel_i, wb_we_i, wb_cyc_i,
  wb_stb_i, wb_ack_o, wb_err_o
);

//
// Parameters
//
parameter           aw = 12;

//
// I/O Ports
//
input               wb_clk_i;
input               wb_rst_i;

//
// WB slave i/f
//
input  [31:0]       wb_dat_i;
output [31:0]       wb_dat_o;
input  [31:0]       wb_adr_i;
input  [3:0]        wb_sel_i;
input               wb_we_i;
input               wb_cyc_i;
input               wb_stb_i;
output              wb_ack_o;
output              wb_err_o;

//
// Internal regs and wires
//
wire          we;
wire [3:0]    be_i;
wire [aw-1:0]adr;
wire [31:0]   wb_dat_o;
reg           ack_we;
reg           ack_re;
```

```
//
// Aliases and simple assignments
//
assign wb_ack_o = ack_re | ack_we;
assign wb_err_o = wb_cyc_i & wb_stb_i & (|wb_adr_i[23:aw+2]);     // If Access to > (8-bit
leading prefix ignored)
assign we = wb_cyc_i & wb_stb_i & wb_we_i & (|wb_sel_i[3:0]);
assign be_i = (wb_cyc_i & wb_stb_i) * wb_sel_i;

//
// Write acknowledge
//
always @ (negedge wb_clk_i or posedge wb_rst_i)
begin
if (wb_rst_i)
    ack_we <= 1'b0;
  else
  if (wb_cyc_i & wb_stb_i & wb_we_i & ~ack_we)
    ack_we <= #1 1'b1;
  else
    ack_we <= #1 1'b0;
end

//
// read acknowledge
//
always @ (posedge wb_clk_i or posedge wb_rst_i)
begin
  if (wb_rst_i)
    ack_re <= 1'b0;
  else
  if (wb_cyc_i & wb_stb_i & ~wb_err_o & ~wb_we_i & ~ack_re)
    ack_re <= #1 1'b1;
  else
    ack_re <= #1 1'b0;
end

  onchip_ram block_ram_0 (
              .addr(wb_adr_i[aw+1:2]),
              .clk(wb_clk_i),
              .din(wb_dat_i[7:0]),
              .dout(wb_dat_o[7:0]),
              .we(we),
              .en(be_i[0]));


  onchip_ram block_ram_1 (
              .addr(wb_adr_i[aw+1:2]),
              .clk(wb_clk_i),
              .din(wb_dat_i[15:8]),
              .dout(wb_dat_o[15:8]),
              .we(we),
              .en(be_i[1]));

  onchip_ram block_ram_2 (
              .addr(wb_adr_i[aw+1:2]),
              .clk(wb_clk_i),
              .din(wb_dat_i[23:16]),
              .dout(wb_dat_o[23:16]),
              .we(we),
              .en(be_i[2]));

  onchip_ram block_ram_3 (
              .addr(wb_adr_i[aw+1:2]),
              .clk(wb_clk_i),
              .din(wb_dat_i[31:24]),
              .dout(wb_dat_o[31:24]),
              .we(we),
              .en(be_i[3]));

endmodule
```

V Synthesis, place & route, generating the bitstream

The following chapter describes how to build a new project and generate the bitstream in ISE 6.1. Other versions of ISE will have a slightly different interface and flow, but they will be all quite similar.

1) Start ISE (*Start → programs → Xilinx ISE … → Project Navigator*)

2) Create a new project (*File → New Project...*)

3) Choose a project name and the working directory. It's best to put it in a seperate directory than the source code. Now click 'next'.

4) On the second screen of the wizard, select the correct "Device Family", 'Device', 'Package' and "Speed Grade". Also select which synthesis tool you're going to use (XST). Simulation settings aren't important because we won't use any simulator.

5) You can skip the "New Source" and the "Add Existing Sources" windows. Press 'Finish' to generate the new project.

6) The next step is to add the source files to the project. Right click with your mouse in the "Sources in Project" window and click "Add copy of Source". Select all the source files and press 'Open'. Repeat these steps until all source files of all subdirectories have been added. Hint: to add the "onchip_ram", add "onchip_ram.edn", "onchip_ram.v" and "onchip_ram_top.v".

7) Now you have to assign the pins of the FPGA to the top-file ports. There are two ways to do this:
   a) *Graphical:* In the "Sources in Project" window, look for the topfile (xsv_fpga_top) and left-click on it. Now expand the 'User constraints' section. Double click "Assign Package Pins". When ISE asks if you want project navigator to automatically generate an UCF file, press "Yes". Xilinx PACE will now open. In the left window "Design objects List", type in the correction pin location for each signal in the 'Loc' column.

   b) *Text:* Create a new file with extension 'ucf'. Now to assign pins, use the following syntax: "NET jtag_tvref LOC = T8;" After assigning all signals, you save and this file to the project. So right click in the "Sources in Project" window and click "Add Source". Select the 'ucf'-file and press 'Open'. When asked which file it should be associated with, select the topfile (xsv_fpga_top).

8) Before compiling the system, first separately compile the onchip ram library by selecting "onchip_ram_top" in the "Sources in Project" window, then double click "Translate" in the window below. In a perfect world, ISE would recognize and compile the library automatically.

9) Now the bit-file can be generated. Select "xsv_fpga_top" and then double click "generate programming file" in the window below.

10) In the bottom "Console"-window, look for the message indicating the achieved clock speed. It should be higher than the divided clock. Otherwise, adjust the division factor of the DLL component and re-generate the system.

VI Download and test the OpenRISC

1) Download your design with Xilinx iMPACT

   a) Make sure you connected your download cable correctly (e.g. parallel cable IV,…), make sure your board is powered on

   b) Start iMPACT *(Start → programs → Xilinx ISE … → Accessories → iMPACT)*

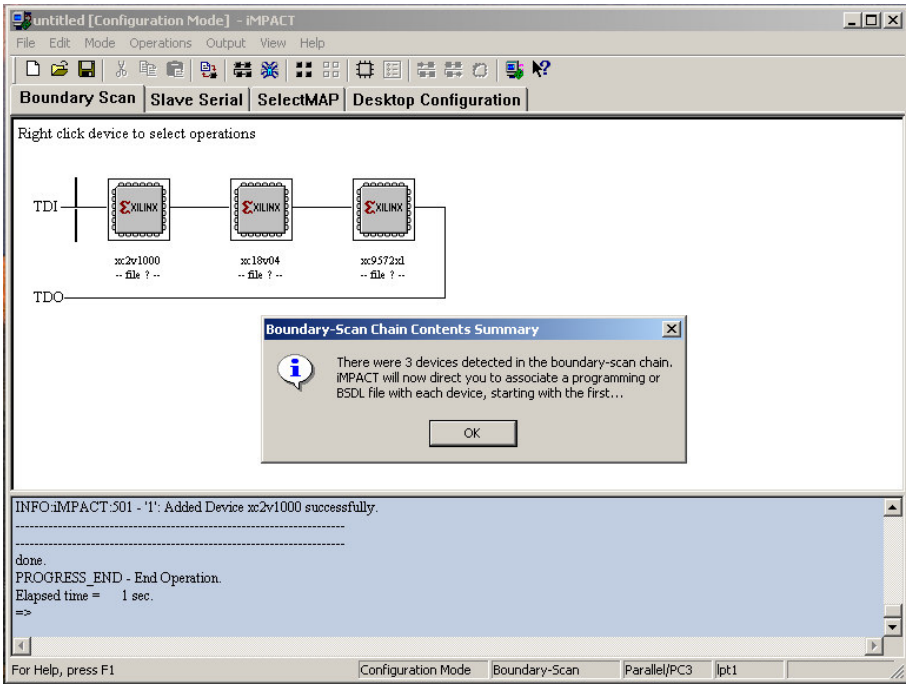
   c) As operation mode select “configure devices”, press on “next”

   d) Configure the device via “Boundary-Scan Mode”, press on “next”

   e) Select “automatically connect to cable and identify Boundary-Scan chain”, press on “next”

   f) Xilinx will report which hardware has been found on the JTAG chain (figure 3)

Figure 3: iMPACT



---

   g) Now Xilinx asks for the configuration files for each device. Select “xsv_fpga_top.”bit for the correct device and click op “open”, press “cancel” for the other devices.

   h) Right click on the target component and select “program…”, in the next screen press on “OK”, and your target device will be programmed.

Congratulations, you have now succesfully generated an OpenRISC based embedded system and downloaded it to an FPGA. To test your system by running software on the processor, follow the instructions of the software tutorial that you can find on our website (http://emsys.denayer.wenk.be).