HOGESCHOOL VOOR WETENSCHAP & KUNST DE NAYER INSTITUUT SINT-KATELIJNE-WAVER

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 Software Tutorial

Linux

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 build a working GNU Toolchain and if you want, to run a simple Hello World program on an OpenRISC processor.

The only thing you need to build a working GNU toolchain is a **PC** with a **Linux** operating system. If you also want to compile and run a simple “Hello World” program, you need of course a working OpenRISC processor with a debug unit and preferably a UART to have some output. This tutorial is designed to work with the Basic Custom OpenRISC embedded system, but it can of course be used with any OpenRISC based embedded system.

Finally, I would like to mention that this tutorial is partially based on the excellent “OR1K uClinux and simulator installation guide” of Robert Cragie. The original page can be found at www.asisi.co.uk/or1k.html.

The flow of setting up software for the OpenRISC is:

1. check gcc. if necessary, get gcc source-code and build correct gcc
2. retreive and build toolchain source-code

You should then be ready to retreive and build software to run on the OpenRISC

HOGESCHOOL VOOR WETENSCHAP & KUNST DE NAYER INSTITUUT SINT-KATELIJNE-WAVER

II GCC installation

first check which version of gcc you have:
1) open a terminal window and type “locate gcc | grep /bin/”
2) check every gcc by running it with the “–version option”: i.e. ”/usr/bin/gcc –version”

For building a correct OpenRISC GNU toolchain you need gcc 2.95.x or 2.96. If you find these, you can skip the remainder of this chapter and proceed to chapter III. If not, you have to install the right gcc.

INSTALL GCC

A. go to “gcc.gnu.org/mirrors.html”, pick a nearby server and download “gcc-2.95.3.tar.gz”
B. in a terminal, go to the location where you saved the gcc-tar file and untar it by typing “tar –xzf gcc-2.95.3.tar.gz”
C. mount into the directory, by typing “cd gcc-2.95.3”
D. to install, you need supervisor priviledges. If you’re logged in as a normal user, type “su root” and the supervisor password.
E. Now configure the install script. To avoid confusion the gcc is renamed gcc-295. You can accomplish this by typing ‘./configure --program-transform-name=”s/\\\\(.*\\\\)/\\\\1-295/” ’
F. To finish, build the gcc by typing “make bootstrap && make install”

III GNU Toolchain installation

In this chapter, you will learn how to build a basic GNU Toolchain. This simple toolchain will suffice to compile and run “Hello World” on the OpenRISC processor.

If you would like to build the complete GNU Toolchain, you can follow the additional instructions in Robert Cragie’s “OR1K uClinux and simulator installation” guide at www.asisi.co.uk/or1k.html.

Notice: the directories where the binaries will be installed in “/opt/or32-uclinux/”, but you can replace it by any other directory of your choice.

1) first set the CVSROOT environment variable: “export CVSROOT=:pserver:cvs@cvs.opencores.org:/home/oc/cvs”
2) then login to the CVS server: “cvs login” and when prompted for a password, enter anything.
3) Before retreiving the OR1K directory (with the source code), go to the directory where you want to place it.
4) Now you can checkout the source-directories by typing:
   - “cvs –z9 co or1k/binutils”
   - “cvs –z9 co or1k/gcc-3.2.3”
   - “cvs –z9 co or1k/gdb-5.0”
   - “cvs –z9 co or1k/jtag”
   - “cvs –z9 co or1k/hello-uart”

   cvs will retreive all these directories and place them in the “current directory/or1k/...”
5) change to this or1k directory: “cd or1k”
6) and login as superuser: “su” and type in the superuser password
7) setup the right gcc: “CC=gcc-295” and setup the right language if you don’t have an English linux: “export LANG=eng”

BINUTILS:

8) make a build directory for the binutils: “mkdir b-b” and change to it: “cd b-b”
9) first configure the makefile : “../binutils/configure −−target=or32-uclinux −−prefix=/opt/or32-uclinux”
10) then build the binutils by typing : “make all install”

PATH SETTINGS

11) now set the path for the compiled binaries: “export PATH=$PATH:/opt/or32-uclinux/bin”

CROSSCOMPILER

12) make a build directory for the gcc-crosscompiler: “mkdir b-gcc” and change to it “cd b-gcc”
13) now configure the cross-compiler: “../gcc-3.2.3/configure −−target=or32-uclinux −−prefix=/opt/or32-uclinux −−local-prefix=/opt/or32-uclinux/or32-uclinux −−with-gnu-as −−with-gnu-ld −−verbose −−enable-languages=c”
14) build it: “make all install”
15) now go back up a directory: “cd ..”

GDB 5.0

16) the last build directory to create is one for gdb: “mkdir b-gdb” and change to it “cd b-gdb”
17) configure: “../gdb-5.0/configure −−target=or32-uclinux”
18) build: “make all”

19) copy: “cp gdb/gdb /opt/or32-uclinux/bin/or32-uclinux-gdb”
20) now you can go back to the or1k root directory: “cd ..”

JP1-PROGRAM
This program isn’t a standard part of a GNU Toolchain, but to setup connection with an OpenRISC processor, it is indispensable.
21) step into the jtag directory: “cd jtag”
22) build the OpenRISC jtag connection software: “make all”

IV Hello World

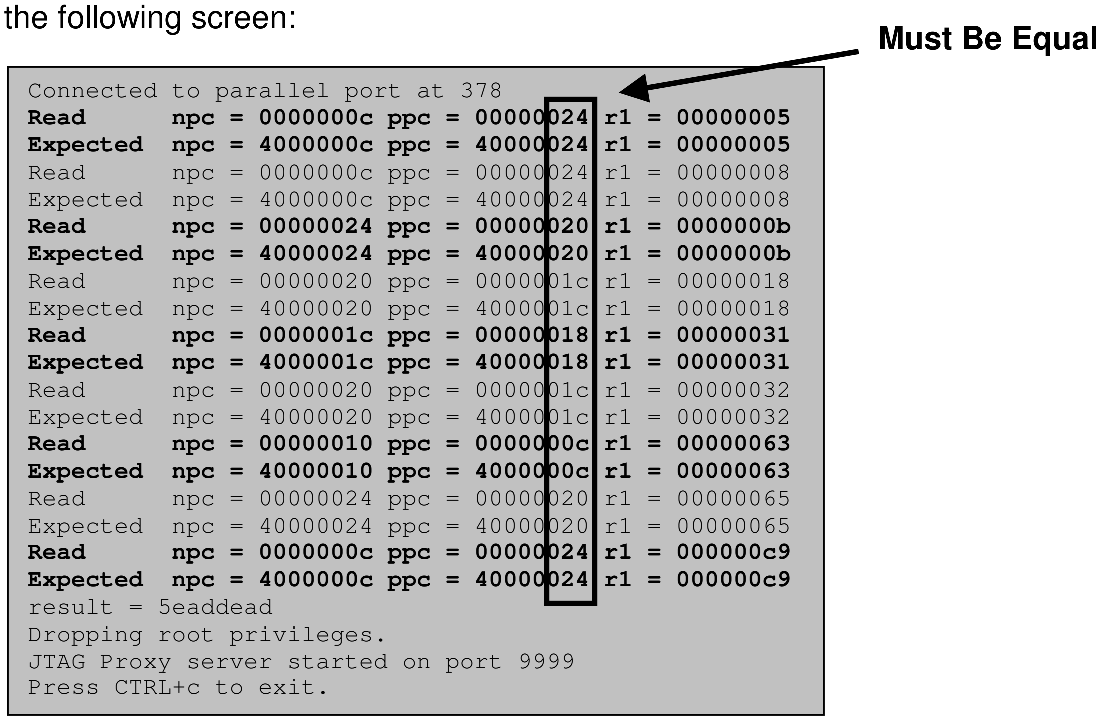
In this chapter you will learn how to build a sample “Hello World” program for the OpenRISC processor.

1) There you have to be made some adjustments to two files.
    a. To “board.h”: the only lines that have to remain are:
        - “# define IN_CLK   10000000”
        - “# define STACK_SIZE   0x1000”
        - “# define UART_BAUD_RATE   19200” *(higher rates might not work)*
        - “# define UART_BASE   0x90000000”
        - the lines with “# define REG8”, “REG16” and “REG32”
    b. The “ram.ld” file has to be adjusted:
        - “ram : ORIGIN = 0x00002000, LENGTH = 0x00002000”
2) In the “or1k/hello-uart” directory, compile the hello world program by typing “make clean all”
3) go back to the or1k root directory: “cd ..”

IV JTAG connection

To make a connetion with the OpenRISC processor, you should first get back into super-user mode ('su') and then go to the jtag directory: "cd jtag".
To run the program type "./jp1-xilinx 9999" (if you make connection via parallel cable III or IV) or "./jp1-xess 9999" (if you make connection to the xess board via parallel port)

You should now get the following screen:



```
Connected to parallel port at 378
Read      npc = 0000000c ppc = 00000024 r1 = 00000005
Expected  npc = 4000000c ppc = 40000024 r1 = 00000005
Read      npc = 0000000c ppc = 00000024 r1 = 00000008
Expected  npc = 4000000c ppc = 40000024 r1 = 00000008
Read      npc = 00000024 ppc = 00000020 r1 = 0000000b
Expected  npc = 40000024 ppc = 40000020 r1 = 0000000b
Read      npc = 00000020 ppc = 0000001c r1 = 00000018
Expected  npc = 40000020 ppc = 4000001c r1 = 00000018
Read      npc = 0000001c ppc = 00000018 r1 = 00000031
Expected  npc = 4000001c ppc = 40000018 r1 = 00000031
Read      npc = 00000020 ppc = 0000001c r1 = 00000032
Expected  npc = 40000020 ppc = 4000001c r1 = 00000032
Read      npc = 00000010 ppc = 0000000c r1 = 00000063
Expected  npc = 40000010 ppc = 4000000c r1 = 00000063
Read      npc = 00000024 ppc = 00000020 r1 = 00000065
Expected  npc = 40000024 ppc = 40000020 r1 = 00000065
Read      npc = 0000000c ppc = 00000024 r1 = 000000c9
Expected  npc = 4000000c ppc = 40000024 r1 = 000000c9
result = 5eaddead
Dropping root privileges.
JTAG Proxy server started on port 9999
Press CTRL+c to exit.
```

If you don't get this screen, something is wrong.

TROUBLESHOOTING

- *problem:* you only get "Connected to parallel port at 378"
  advice: check the connection with the debug unit.

- *problem:* 'Read' and 'Expected' values are always zero
  advice: check the OpenRISC reset state

- *problem:* 'Read' and 'Expected' values always differ
  advice: check the onchip-RAM module

- *problem:* 'Read' and 'Expected' values only sometimes differ
  advice: check the OR1200 core itself, more specifically check the timing of the ALU registers.

V GDB connection

Finally you're ready to upload the Hello World program and run it.

1) Connect the serial port of your board to a PC and open a UART-terminal window
2) go to the directory where Hello World has been untarred.
3) Enter "or32-uclinux-gdb hello.or32" to run gdb
4) In gdb type in the following commands:
   - "target jtag jtag://localhost:9999"
   - "load"
   - "set $pc=0x100"
   - "c"

You should now see a "Hello World!" appear on the UART terminal window. You can even send characters. The OpenRISC will return a character one higher. So if you send 'a', you get 'b' on the terminal.