Openrisc Toolchain Development Factory

Home

This is my Openrisc toolchain page which includes(usually) uClibc and a standard Linux kernel.

In the near future we will be caught up to the official uClibc (0.9.29) and we will have a buildroot port as well.

If you already have a working Openrisc toolchain and you just want my kernel patches and etc just go here.

Building a toolchain can be a forbidding task --make sure to carefully follow instructions. Below are the steps to build a toolchain for the OpenRisc-1xxx core --it is best to have a fast internet connection, loads of patience, and a tolerence for "black magic". :)

NOTE: MOF and ORSoC now provide a fully automated toolchain script please go to: Installation Script

To learn more about this awesome "freecore" go to: Opencores

Before you start, make yourself a nice clean directory. For this exercise I will be using a directory called: /video/new_or32. The instructions at Opencores use /opt which is fine too.

To start building an Openrisc toolchain with uClibc:
Get a fresh binutils-2.16.1 at: Binutils
Get a fresh gcc-3.4.4 compiler at: GCC
Get a fresh uClibc-0.9.28.3 at: uClibc

Ok you have the code from the most excellent sources --now for some patches to stick it all together:
First the Binutils patch, you can pick it up at: OR1K Toolchain or locally here (this has a major bug fix): binutils-2.16.1-fixed-unified.diff.bz2

Now get the GCC compiler patch, (for consistency the local copy is compressed the OR1K version is not):

OR1K Toolchain or locally here: gcc-3.4.4-or32-unified.diff.bz2

Now it is time to pick up a fresh Linux 2.6.19 kernel. You might be wondering why do I have to do this? It is the big toolchain lie --toolchains are deeply coupled to OSes --you'll see...
Linux Kernel Source

Next pick up the following patch for the 2.6.19 kernel here: linux-2.6.19-or32-unified

Last let's pick up the uClibc-0.9.28.3 patch here: uClibc-0.9.28-or32-unified.bz2
And the uClibc-0.9.28.3 supplemental as well:uClibc-0.9.28.3-libc-supplemental

This is the powerful or32 simulator tarball when you're done with the toolchain you can run or32 linux in it.

Here's some other linux patches for Openrisc that have been somwhat tested(booted) on the Opencores simulator:
linux-2.6.18-or32-unified
linux-2.6.17-or32-unified

Had enough yet? No? Ok let's build this beast then. Next

TOP

Openrisc Toolchain Development Factory

Home

BINUTILS

Ok so we have all the code we need to get started. We begin with binutils which is by far the easiest thing to build:

Let's unpack it I just use somthing like:
ralphie:/video/new_or32# bunzip2 binutils-2.16.1.tar.bz2
then:
ralphie:/video/new_or32# tar -xvf binutils-2.16.1.tar
now patch it:
ralphie:/video/new_or32# cd binutils-2.16.1
ralphie:/video/new_or32# bzip2 -dc ../binutils-2.16.1-unified.diff.bz2 |patch -p1

Now we need to configure and build it, but before we do let's do this:
ralphie:/video/new_or32# cd ..
ralphie:/video/new_or32# mkdir b-b (GCC and Binutils build from OUTSIDE of their native directories)
ralphie:/video/new_or32# mkdir tools
now:
ralphie:/video/new_or32# cd b-b

Time to configure:
../binutils-2.16.1/configure --target=or32-uclinux --prefix=/video/new_or32/tools/or32-uclinux

Now build it:

ralphie:/video/new_or32/b-b# make all install
Once done you should be able to do: ralphie:/video/new_or32# ls tools/or32-uclinux/bin
and see some nice shiny new binary utilities for the Openrisc architecture.

Don't forget to let the system know about your new utils!!:
export PATH=/video/new_or32/tools/or32-uclinux/bin:$PATH

More pain for you? Yes? Next No?Previous

TOP

Openrisc Toolchain Development Factory

Home

GCC (Part I)

So now we'll give new meaning to the term orthogonal!!

Let's go back to our base level directory and unpack our kernel --BTW this kernel is just to get us some headers and paths. Don't worry about using it to build your real kernel

ralphie:/video/new_or32# bunzip2 linux-2.6.19.tar.bz2
then:
ralphie:/video/new_or32# tar -xvf linux-2.6.19.tar
now patch it:
ralphie:/video/new_or32# cd linux-2.6.19
ralphie:/video/new_or32# bzip2 -dc ../linux_2.6.19_or32_unified_simtested.bz2 |patch -p1

Ok time to fake a kernel:
ralphie:/video/new_or32# make menuconfig ARCH=or32
Just select or32 as your platform we just need headers and paths.

Now let's make an include directory under our toolchain for the Linux and architecture specific includes:
ralphie:/video/new_or32# mkdir tools/or32-uclinux/include
ralphie:/video/new_or32# mkdir tools/or32-uclinux/include/asm
ralphie:/video/new_or32# mkdir tools/or32-uclinux/include/linux

Now we need to copy the headers:
ralphie:/video/new_or32# cp -f -dR linux-2.6.19/include/linux/* /video/new_or32/tools/or32-uclinux/include/linux
ralphie:/video/new_or32# cp -f -dR linux-2.6.19/include/asm-or32/* /video/new_or32/tools/or32-uclinux/include/asm

Now let the compiler know about the architecture and OS headers:
ralphie:/video/new_or32# cd /video/new_or32/tools/or32-uclinux
ralphie:/video/new_or32/tools/or32-uclinux# ln -s include sys-include

Ok so now lets go back to the base station:
ralphie:/video/new_or32/tools/or32-uclinux# cd ../../

And it is time to make yet another build directory this time for GCC:
ralphie:/video/new_or32/tools/or32-uclinux# mkdir b-gcc

Time to unpack the compiler beast:
ralphie:/video/new_or32/# bunzip2 gcc-3.4.4.tar.bz2
ralphie:/video/new_or32/# tar -xvf gcc-3.4.4.tar

And DON'T forget to patch it:
ralphie:/video/new_or32/# cd gcc-3.4.4
ralphie:/video/new_or32/gcc-3.4.4/# bzip2 -dc ../gcc-3.4.4-or32-unified.diff.bz2 |patch -p1

Back to base station:
ralphie:/video/new_or32/gcc-3.4.4/# cd ../
We are FINALLY just about ready to build it:
ralphie:/video/new_or32/gcc-3.4.4/# cd b-gcc

Let's configure it:
ralphie:/video/new_or32/b-gcc/# ../gcc-3.4.4/configure --target=or32-uclinux --prefix=/video/new_or32/tools/or32-uclinux --with-local-prefix=/video/new_or32/tools/or32-uclinux/or32-uclinux --with-gnu-as --with-gnu-ld --verbose --enable-languages=c

Lastly let's do it:
ralphie:/video/new_or32/b-gcc/# make all install

Oh before we leave let's change directories back to our linux directory and build enough
of the kernel to get a version header:
ralphie:/video/new_or32/b-gcc/# cd ../linux-2.6.20
ralphie:/video/new_or32/linux-2.6.20/# make vmlinux ARCH=or32 CROSS_COMPILE= \
/video/new_or32/tools/or32-uclinux/bin/or32-uclinux-

More skullduggery for you? Yes? Next No?Previous Start

Top

Openrisc Toolchain Development Factory

Home

---

uClibc-0.9.28.3

uClibc pronounced "you-see-lib-see" is what is known as a "C" library, uClibc is the product of the crew at: uClibc they are also the folks that do BusyBox BusyBox and Buildroot Buildroot.

Like the Opencores crew they have donated mightly to embedded software and hardware engineering. (BTW it is often from their own pockets!!)

If you cannot donate time or money yourself make damn sure you hound your company (who may be benefitting from this work) to either donate or send business to these excellent organizations.

Ok so the philosopy is out of the way --let's get started.

First unpack uClibc-0.9.28 that we downloaded long ago:
ralphie:/video/new_or32# bunzip2 uClibc-0.9.28.3.tar.bz2
ralphie:/video/new_or32# tar -xvf uClibc-0.9.28.3.tar

Ok so now patch it:
ralphie:/video/new_or32# cd uClibc-0.9.28.3
ralphie:/video/new_or32/uClibc-0.9.28.3# bzip2 -dc ../uClibc-0.9.28-or32-unified.bz2 |patch -p1
Next then cd to uClibc-0.9.28.3/libc
ralphie:/video/new_or32/uClibc-0.9.28.3/libc# bzip2 -dc ../../uClibc-0.9.28-or32-libc-support.bz2 |patch -p1

Alright so now we have to muck with some internals same as they do on the Opencores page Opencores Projects

ralphie:/video/new_or32/uClibc-0.9.28.3# ln -s extra/Configs/Config.or32 Config

Now we need to set the CC environment variable since we to use the cross to build uClibc
ralphie:/video/new_or32/uClibc-0.9.28.3# export CC=or32-uclinux-gcc

Next we also need to set the KERNEL_SOURCE and DEVEL_PREFIX so we will use make menuconfig to do so:
ralphie:/video/new_or32/uClibc-0.9.28.3# make menuconfig
Under Target Features and options set the your kernel (e.g /video/new_or32/linux-2.6.19)
Under Library Installation Options set the path to your toolchain in the development directory.

Please start with this sample .config: sampledotconfig
Let's let it rip
ralphie:/video/new_or32/uClibc-0.9.28.3# make all install
If you happen to get some header errors in asm just go to those files
and remove the "#include linux/config.h" since it is depracated.

Finish the voodoo ? Yes? Next No?Previous Start

Top

Openrisc Toolchain Development Factory

Home

GCC (Part II that's right part deux)

Ok so now we are headed back to the beast known as GCC, this step will integrate our uClibc into GCC so we have a working C compiler.

First order of business let's unset the CC variable we set earlier then we wanted to cross-compile uClibc but we damn sure do not want to cross-compile our cross-compiler do we!!

Let's get back to base station:
ralphie:/video/new_or32/uClibc-0.9.27# cd ../
ralphie:/video/new_or32# cd b-gcc now let's unset that damn variable:
ralphie:/video/new_or32/b-gcc# unset CC

Ok so now we have to reconfigure the GCC:
ralphie:/video/new_or32/b-gcc# ../gcc-3.4.4/configure --target=or32-uclinux --prefix=/video/new_or32/tools/or32-uclinux --with-local-prefix=/video/new_or32/tools/or32-uclinux/or32-uclinux --with-gnu-as --with-gnu-ld --verbose --enable-languages=c

And once again:
ralphie:/video/new_or32/b-gcc# make all install

Last steps:
ralphie:/video/new_or32/b-gcc# cd /video/new_or32/tools/or32-uclinux/or32-uclinux
ralphie:/video/new_or32/tools/or32-uclinux/or32-uclinux# ln -s ../include sys-include
ralphie:/video/new_or32/tools/or32-uclinux/or32-uclinux# cd lib
ralphie:/video/new_or32/tools/or32-uclinux/or32-uclinux/lib# cp ../../lib/*.* .

And the build of the tool chain is DONE!!

Piece of Cake ? Yes? Start No?Previous

Top

Here are my steps/hacks for g++:

1) Build and configure everything as you normally would using uClibc-0.9.28.3 (gcc-3.4.4) but go to tools/or32-uclinux/or32-uclinux/ and do this:
ln -s ../include sys-include
and go to tools/or32-uclinux/or32-uclinux/lib and do this:
cp ../../lib/*.* .
!!BEFORE you re-build gcc the second time.

2)Second pass gcc configure add --with-newlib --enable-languages=c,c++ but before you do go to /music/vivace_c++/gcc-3.4.4/libstdc++-v3/configure.ac and add this around DLOPEN:
if test "x${with_newlib}" != "xyes"; then
AC_LIBTOOL_DLOPEN
fi
The "newlib" line just tricks configure to skip the tests.

3) Next go to
/music/vivace_c++/gcc-3.4.4/libiberty/strsignal.c
and change the arguments in psignal as follows:
void
psignal (signo, message)
/*unsigned*/int signo;
const char *message

4) Next add the following:
#define _U 01 //upper
#define _L 02 //lower
#define _N 04 //number
#define _S 010 //space
#define _P 020 //punctuation

#define _C 040 //control
#define _X 0100 //hex
#define _B 0200 //blank
to:
b-gcc/or32-uclinux/libstdc++-v3/include/or32-uclinux/bits/ctype_base.h

5) Then add the following:
char _ctype_[257];
to:
b-gcc/or32-uclinux/libstdc++-v3/include/or32-uclinux/bits/ctype_noninline.h
I didn't initialize _ctype_ we may have to if the
iostream stuff seg faults.

6) Last go to
b-gcc/or32-uclinux/libstdc++-v3/src/c++locale.cc and
remove the glibc dependendies:
_GLIBCXX_HAVE_STRTOF
_GLIBCXX_HAVE_FINITEF
_GLIBCXX_HAVE_FINITE
_GLIBCXX_HAVE_ISIN
I just commented them out for now and forced the
defaults, I had to this with the MIPs compiler too
newlib config line causes. We will have to fix this in
the future.

Things should build and you should be able to at least
build "hello world".

I plan to clean this up once we get it to work.
The hacks in the build area (b-gcc) really suck. I
borrowed most of the code ideas from the Nios guys.

Home