How to Compile a multilib RISC-V GNU Toolchain for Windows

Ryno Alberts (https://shadowcode.io/author/rynoalberts/) |

June 15, 2020 |

Code (https://shadowcode.io/category/code/) / Firmware (https://shadowcode.io/category/firmware/) |

0 Comments (https://shadowcode.io/compile-risc-v-gnu-toolchain-for-windows/#respond)

Table of Contents

Introduction

Compiling toolchains in general can be hard work and complicated. Thankfully as with everything else in life these days there are easier ways to get things done. Here we will look at an “easy” way on how to compile a RISC-V GNU toolchain for Windows with multilib and nano support. We will also be able to select which libraries we want and we will build the “nano” variants of those libraries also.

Requirements

- Ubuntu (I’m running Ubuntu 20.04 on a virtual machine)
  - It is easier to build a Windows toolchain on Linux than building a Windows toolchain on Windows, go figure...

And thats all we need to compile a RISC-V GNU Toolchain for Windows.

Prerequisites

First we need to install the prerequisites that Crosstool-NG (Hereafter refered to as ct-ng) needs.

Enter this into the terminal.

```
sudo apt-get install -y gcc g++ gperf bison flex texinfo he
```

Next we clone a customized version of Crosstool-NG from Stephanos Ioannidis on Github called

. You can use the vanilla Crosstool-NG if you wish, however it will not build the “nano” libraries. All the steps are identical on both. I opted to put the clone inside the “ct-ng” directory because its shorter to type.

```
git clone --recursive https://github.com/stephanosio/zephyr
```

Enter directory we just cloned.

```
cd ct-ng
```

Checking the contents of the directory we have

```
ryno@shadowcode-linux:~/Desktop/ct-ng$ ls
bash-completion  configure.ac  ct-ng.in  issue_template.md
bootstrap        contrib       debian    kconfig
config           COPYING       docs      LICENSE
ryno@shadowcode-linux:~/Desktop/ct-ng$
```

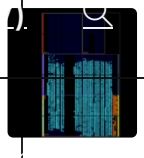
Pre-install configuration

Before we continue we need to add the “

” configuration to the samples folder.

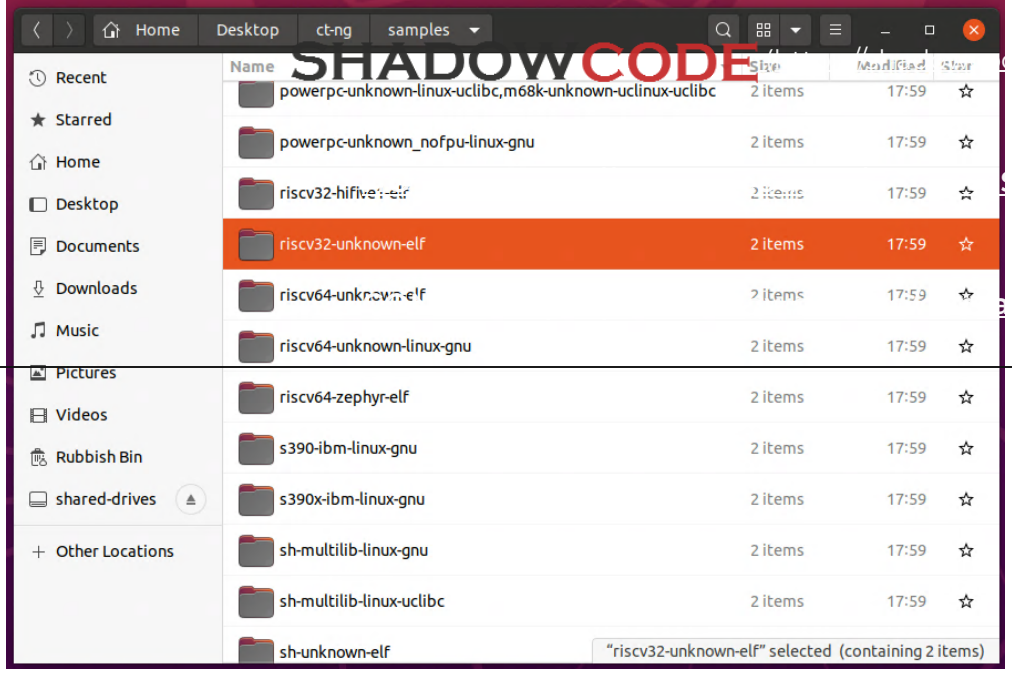
Inside the /ct-ng/samples directory copy the riscv32-unknown-elf folder.

MAY 5, 2020 /
0 COMMENTS
(HTTPS://SHADOWCODE.IO/VITIS-SREC-SPI-BOOTLOADER-SOFTWARE/#RESPOND)

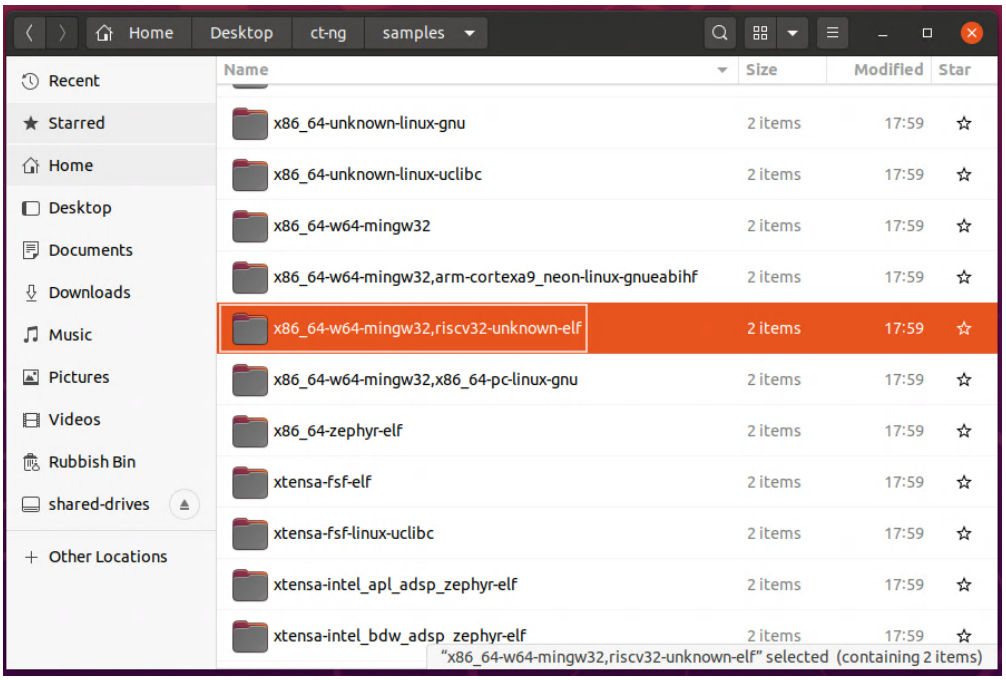


APRIL 27, 2020 /
0 COMMENTS
(HTTPS://SHADOWCODE.IO/MICROBL
-SREC-SPI-BOOTLOADER-
HARDWARE/#RESPOND)

Categories



Rename the folder to “x86_64-w64-mingw32,riscv32-unknown-elf”



Enter the folder you just created and open the “crosstool.config” file, add the following two lines of text to the bottom of the file.

```
CT_CANADIAN=y
CT_HOST="x86_64-w64-mingw32"
```

```
CT_CONFIG_VERSION="3"
CT_EXPERIMENTAL=y
CT_ARCH_RISCV=y
CT_TARGET_VENDOR=""
CT_LIBC_NONE=y
# CT_CC_GCC_LDBL_128 is not set
CT_CANADIAN=y
CT_HOST="x86_64-w64-mingw32"
```


Bootstrap & Configure

Run the bootstrap script next,

```
./bootstrap
```

After this completes run the configure script and optionally supply a folder name where we want ct-ng installed.

```
./configure --prefix=/home/ryno/Desktop/zt-ng
```

I chose to install it to a folder called zt-ng, you need to supply an absolute path if you wish to store it somewhere else.

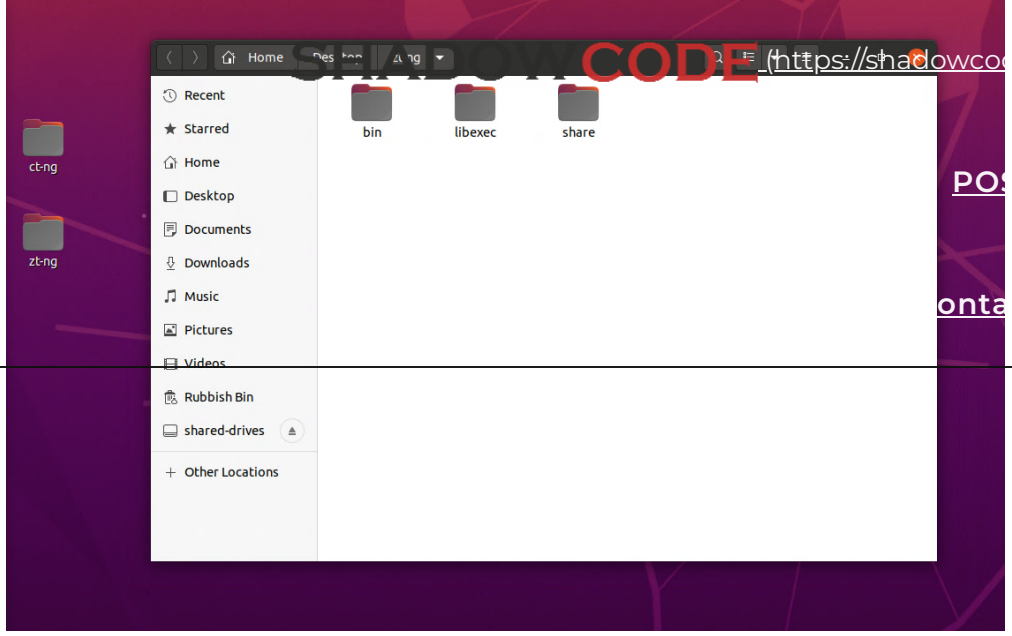
Run make.

```
make
```

Then make install.

```
make install
```

This is what you should have on your desktop now.



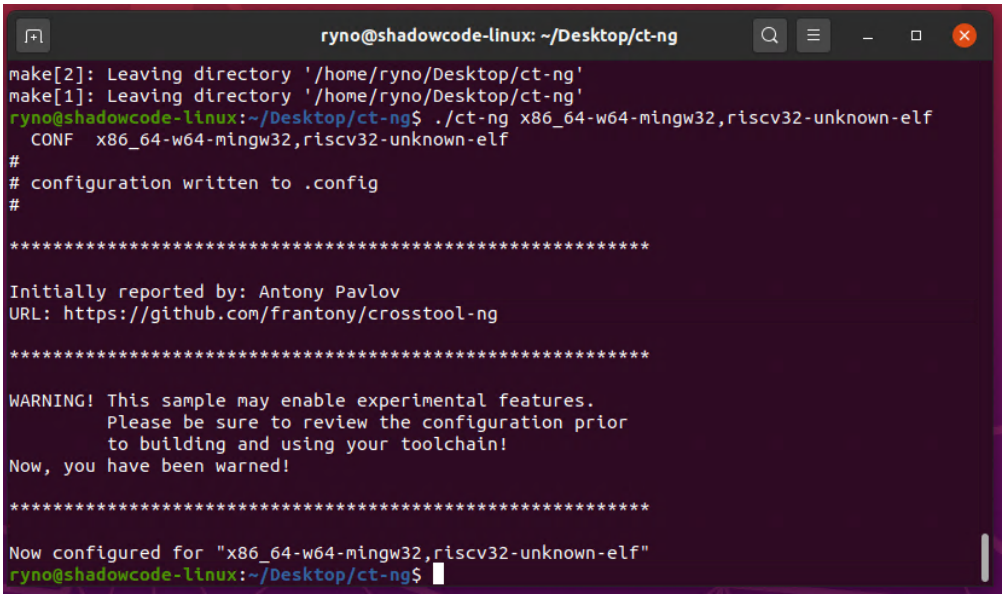
Toolchain Configuration

Enter the “bin” directory we just installed to.

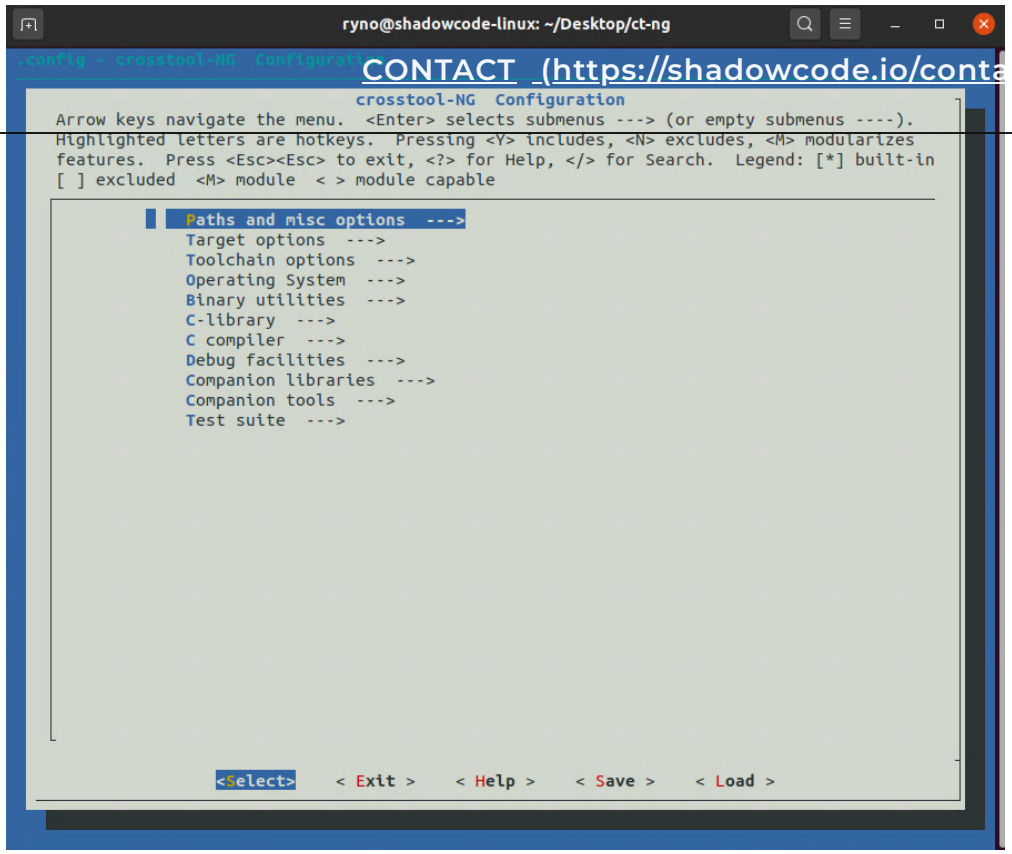
```
cd ../zt-ng/bin
```

Next we will load the configuration script we created earlier.

```
./ct-ng x86_64-w64-mingw32,riscv32-unknown-elf
```

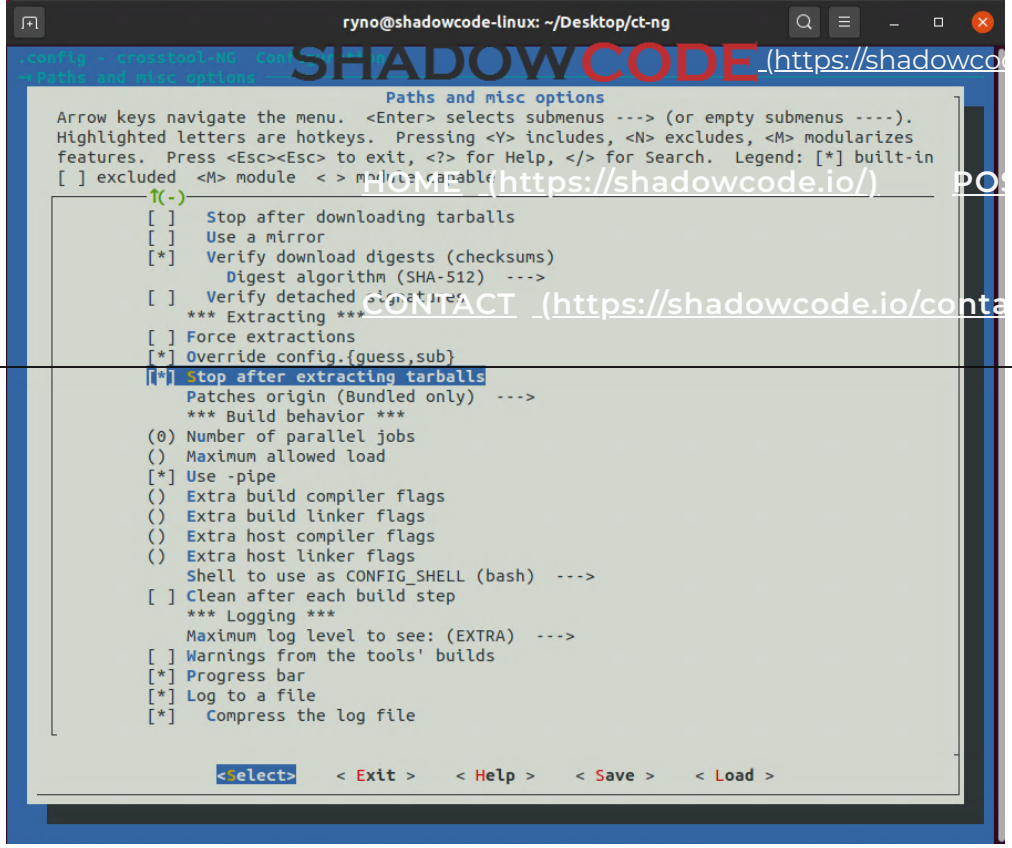


```
./ct-ng menuconfig
```



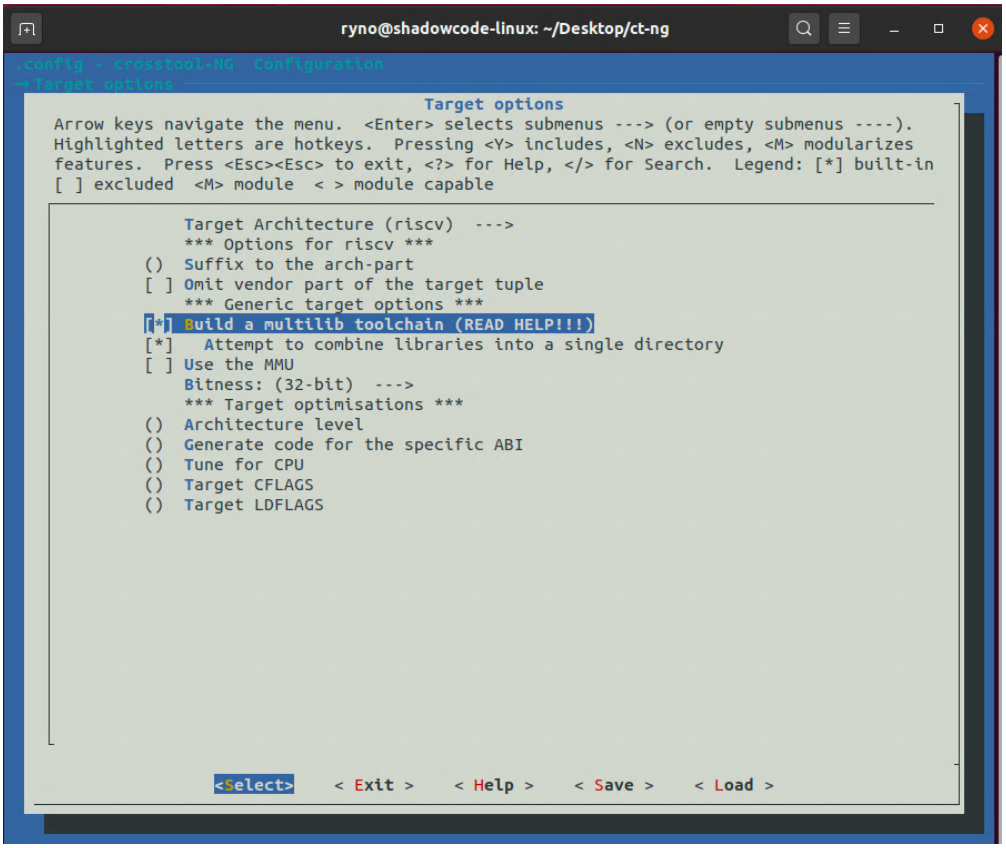
We have several settings to modify now.

Enter the “Paths and misc options” submenu, scroll down and select “Stop after extracting tarballs”. We do this initially because we need to edit some files in the GCC package that ct-ng downloads. After we have edited those files we will come back here and disable this setting which will allow ct-ng to progress past this point and finish the build process. ct-ng will not re-download or extract the tarballs a second time.

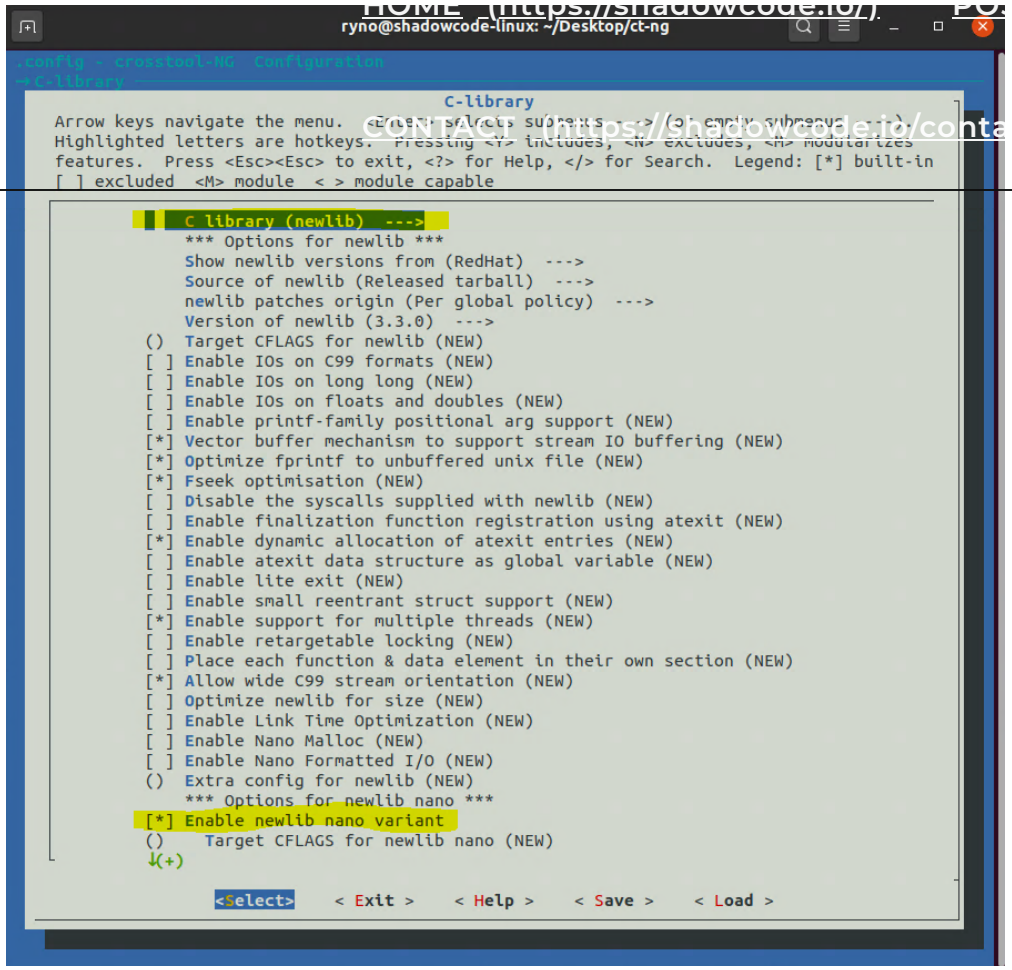


Go back to the main menu and enter the “Target options” submenu.

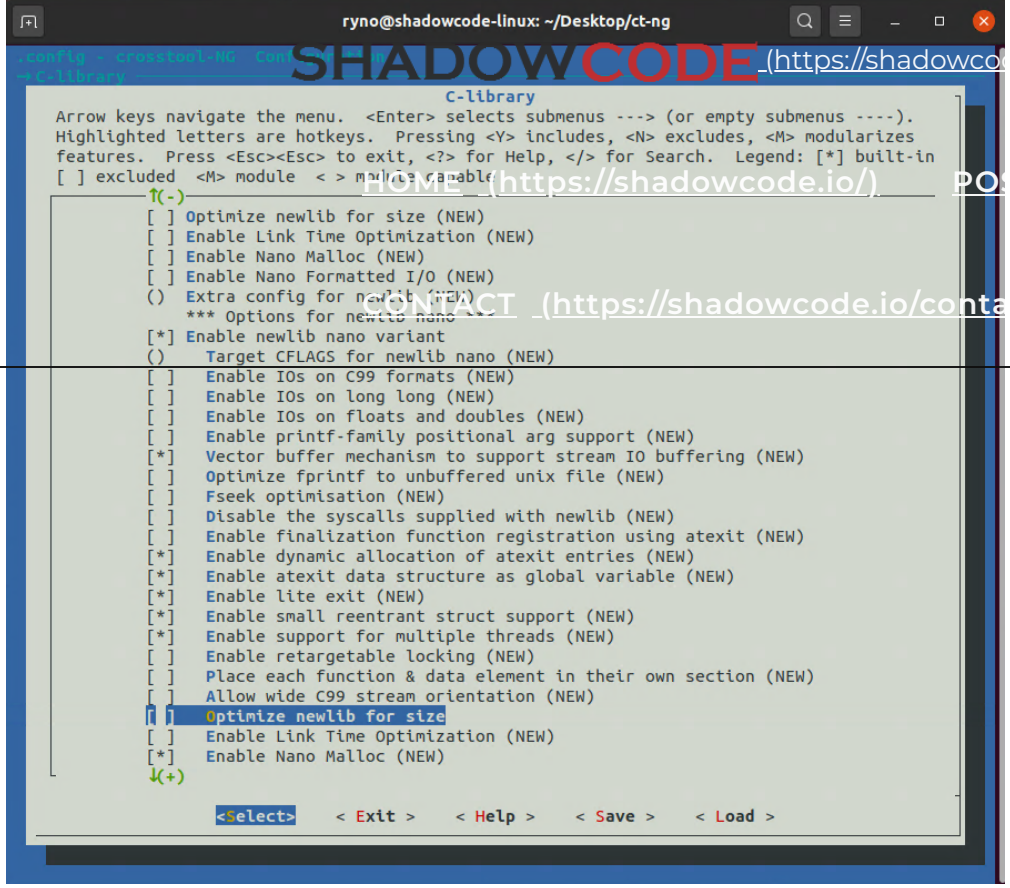
Select “Build a multilib toolchain”.



Go back to the main menu and enter the “C-Library” submenu. Here select “Newlib” as the C library. Several more options will be enabled. Scroll down and select “Enable newlib nano variant” as well.

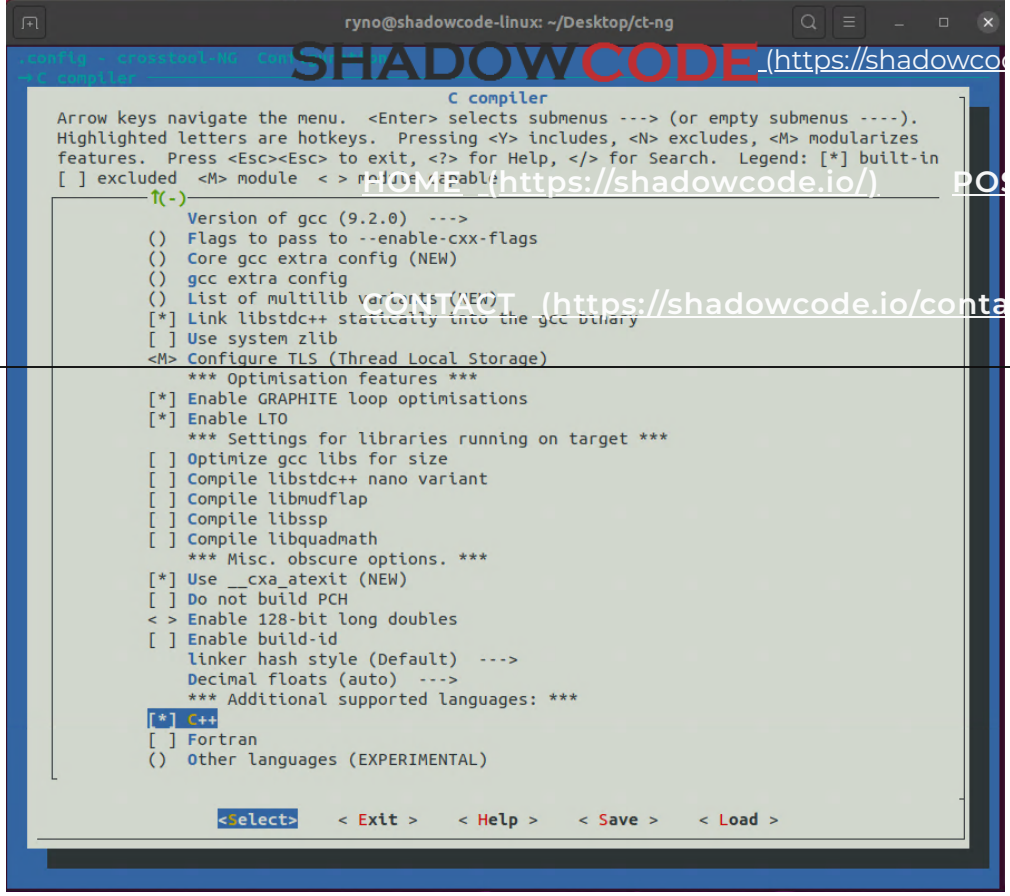


[Optional] Under the options for “Enable Newlib nano variant” deselect the “Optimize newlib for size” option. You can select this option if you wish, however you will need to manually strip the binaries afterwards because that does not happen as part of the size optimizations so you end up with binaries that are actually larger.

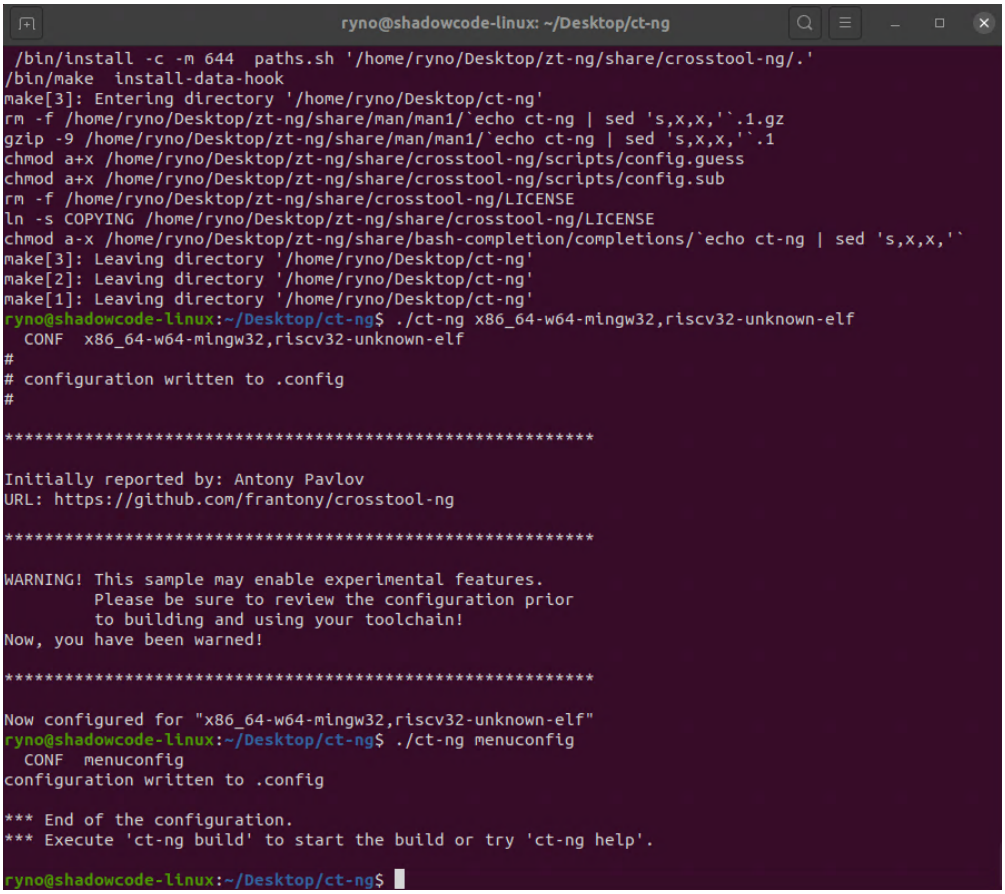


If you know what you are doing there are a whole host of settings to tweak on this page.

[Optional] Go back to the main menu and enter the “C Compiler” submenu, scroll to the bottom and select “C++”.



Thats all we need to do for now. Once you are back at the main menu press “Esc” to exit, hit “Enter” to save your configuration file.



Download and extract packages

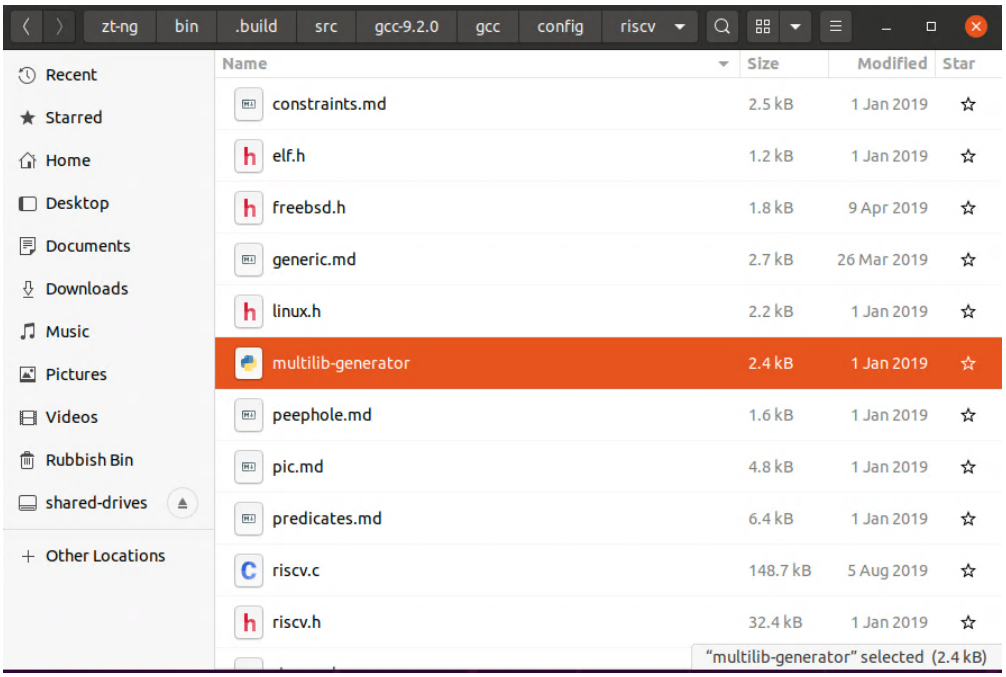
Enter the following command and wait for ct-ng to download and extract all the packages that it needs. It will stop automatically after extracting the tarballs becauase we enabled that setting earlier in the menu.

```
./ct-ng build
```

Change GCC multilib options

[Optional] Next we need to tell gcc which multilibs we want. To do this we need to run a python script called “multilib generator” located in the following directory. Note: /.build/ is hidden by default, you need to enable “show hidden files” to see it.

/home/<your name>/Desktop/zt-ng/bin/.build/src/gcc-9.2.0/gcc/config/riscv/



We have to copy the output of this script to a file called “t-elf-multilib” which is in the same directory. More information on this script can be found in the related sifive blog,

.

Here is the default contents of the “t-elf-multilib” file.

```
# This file was generated by multilib generator with the co
#  ./multilib-generator rv32i-ilp32--c rv32im-ilp32--c rv32
MULTILIB_OPTIONS = march=rv32i/march=rv32ic/march=rv32im/ma
MULTILIB_DIRNAMES = rv32i \
rv32ic \
rv32im \
rv32imc \
rv32iac \
rv32imac \
rv32imafc \
rv32imafdc \
rv32gc \
rv64imac \
rv64imafdc \
rv64gc ilp32 \
ilp32f \
lp64 \
lp64d
MULTILIB_REQUIRED = march=rv32i/mabi=ilp32 \
march=rv32im/mabi=ilp32 \
march=rv32iac/mabi=ilp32 \
march=rv32imac/mabi=ilp32 \
march=rv32imafc/mabi=ilp32f \
march=rv64imac/mabi=lp64 \
march=rv64imafdc/mabi=lp64d
MULTILIB_REUSE = march.rv32i/mabi.ilp32=march.rv32ic/mabi.i
march.rv32im/mabi.ilp32=march.rv32imc/mabi.ilp32 \
march.rv32imafc/mabi.ilp32f=march.rv32imafdc/mabi.ilp32f \
march.rv32imafc/mabi.ilp32f=march.rv32gc/mabi.ilp32f \
march.rv64imafdc/mabi.lp64d=march.rv64gc/mabi.lp64d
```


Eessentially we feed the script with a string that will determine which multilibs gcc will build. The string has the following format,

[arch]-[abi]-[additional arch]-[additional suffix]

- [arch] – The acrhitecture we want a library for.
- [abi] – The abi this arhictecture targets

- [additional arch] – we can specifiy additional architectures here in a comma seperated list. I will be using a third party
- [suffix] – We can add additional suffix's that will be linked to this library. eg. 'c', 'd', 'a' etc.

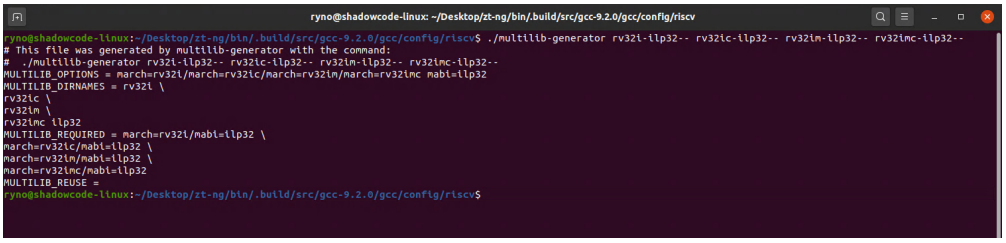
I need libraries to use with a core, which can be configured as either a rv32i, rv32ic, rv32im or rv32imc core. So my configure string will look like this,

```
./multilib-generator rv32i-ilp32-- rv32ic-ilp32-- rv32im-il
```

Which can also be expressed like this.

```
./multilib-generator rv32i-ilp32--c rv32im-ilp32--c
```

You do not have to change any of this if all the multilibs you require are already part of the default configurations. You also dont have to remove any like I just did, if the one you want is not there you can just add it on. I stripped all the other libs because I wont use them and I wanted to speed up the build time as much as possible.



Copy the script output to the "t-elf-multilib" file which is in the same directory and save it.

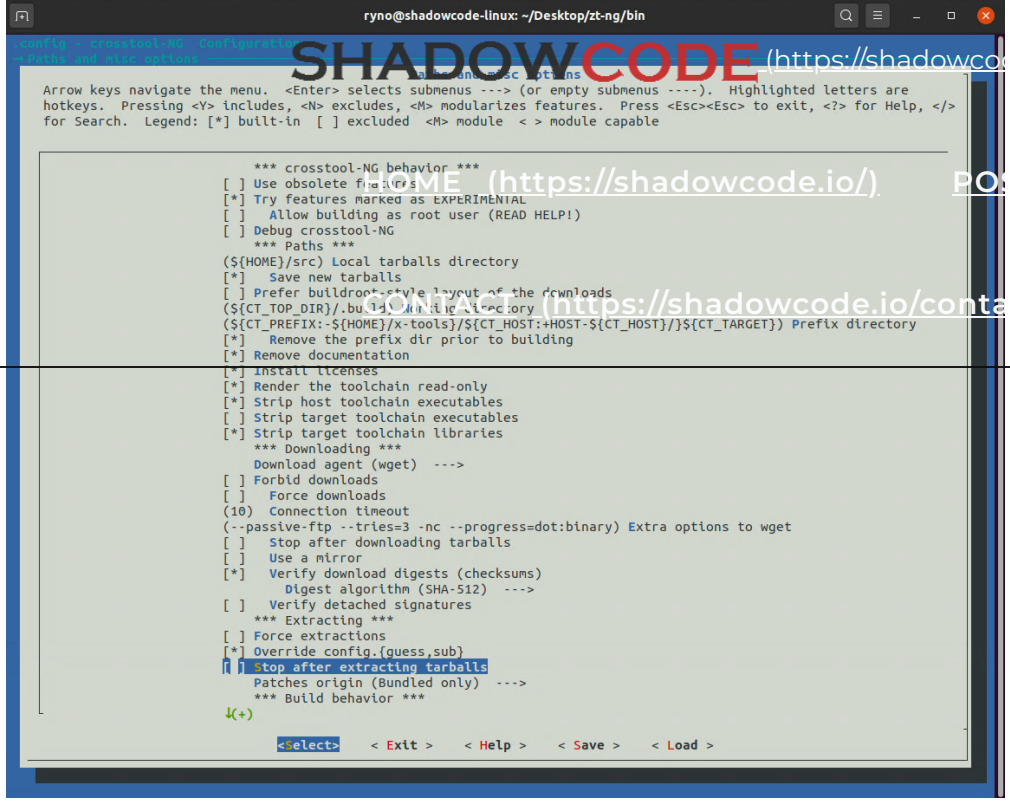
Building the RISC-V Windows toolchain

OK, we're all done messing around, now its time to build our RISC-V GNU Toolchain for Windows. Lets go back to the menuconfig again so we can disable the setting to stop the build after extracting the tarballs, this will allow the build to progress as normal and use the mutilib options we just configured for gcc.

Go to the /zt-ng/bin directory and run,

```
./ct-ng menuconfig
```

Under the "Paths and misc options" submenu scroll down and de-select "Stop after extracting tarballs"



Exit and hit enter to save the configuration.

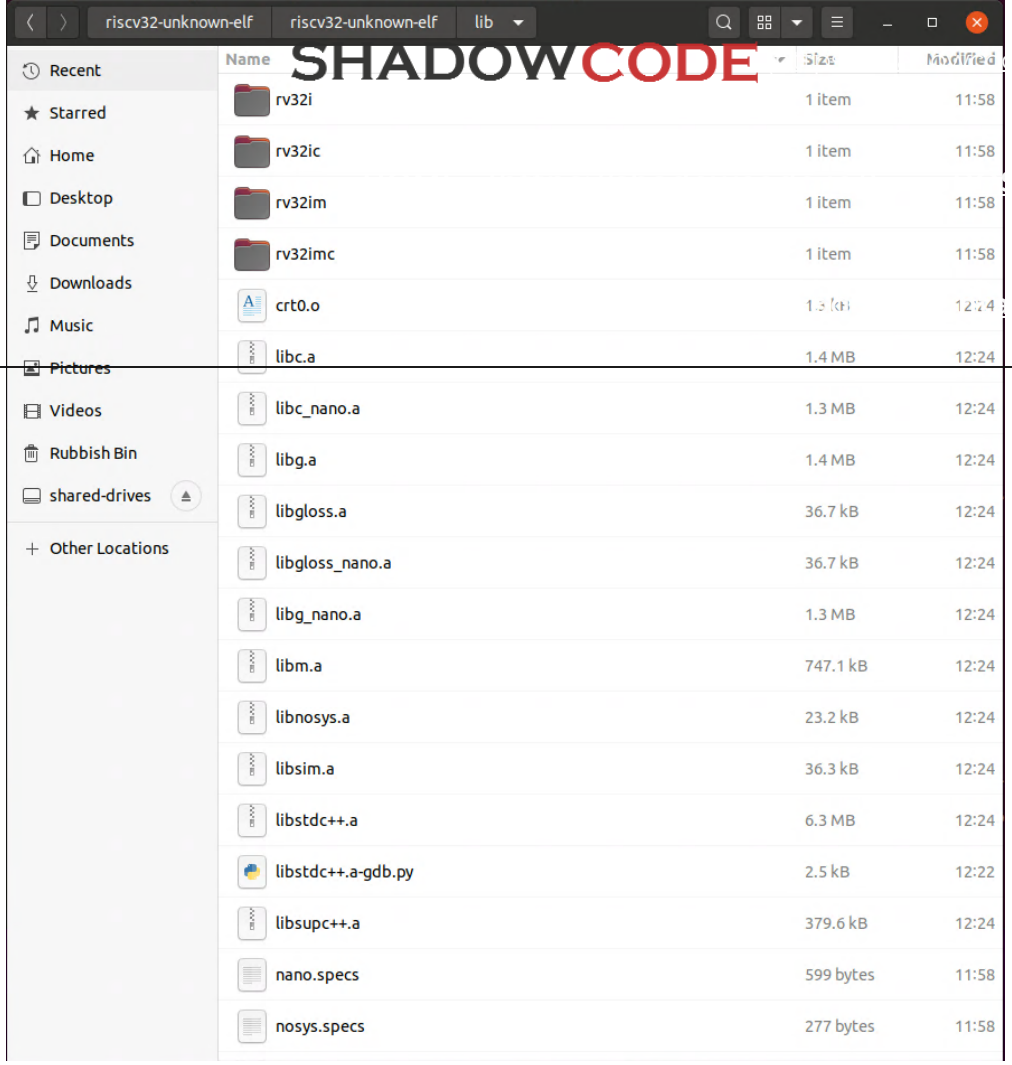
Now enter,

```
./ct-ng build
```

and wait while crosstool-ng builds your toolchain, this will take a while.

When its done you will have a folder called “x-tools” in the /home/<your name>/ directory. Inside this folder you will find your newly build toolchain.

If we look inside the riscv32-unknown-elf/riscv32-unknown-elf/lib directory we can see the libraries we built and the nano variants.



After copying the files over to Windows and adding the path of the "/riscv32-unknown-elf/bin" folder to the "Path" environment variables and running "riscv32-unknown-elf-gcc -print-multi-lib" I get the below output. Success.

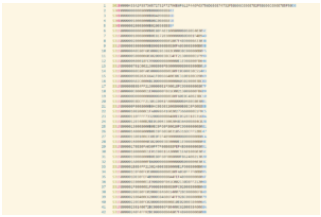
```
C:\Users\Ryno>riscv32-unknown-elf-gcc -print-multi-lib
.;
rv32i/ilp32;@march=rv32i@mabi=ilp32
rv32ic/ilp32;@march=rv32ic@mabi=ilp32
rv32im/ilp32;@march=rv32im@mabi=ilp32
rv32imc/ilp32;@march=rv32imc@mabi=ilp32

C:\Users\Ryno>
```

Hope you found how to compile a RISC-V GNU toolchain for Windows with multilib and nano support useful. Happy coding.

SHADOWCODE

PLEASE SHARE THIS

YOU MIGHT ALSO LIKE



May 5, 2020

Leave a Reply

SHADOWCODE

Your comment here...

Name (required)

Email (required)

Website

Save my name, email, and website in this browser for the next time I comment.

I am human

Privacy - Terms

Your Email