**NUTTX on STM32F746g-DISCO at Lucy Electric**

**You will need CYGWIN (not PuTTY) if you are working on a windows PC to get suitable access to the linux server where the project is located. This is because you will need to use a graphical setup application called Kconfig\_frontends.**

**From the Cygwin terminal window secure shell into server ukbandvsl05v**

**The Arm toolchain should be installed (** sudo apt-get install gcc-arm-none-eabi **)**

You should choose a place somewhere along **your** **local path** to start **your** **working path** for the nuttx image.

We will end up with a working folder structure in which to work as below. Start by **creating** the directory **nx** in a path you want to work in.

**your local path (should not include any spaces in the folder names)**

**|**

**| your working path for nuttx**

**| /**

**|----nx**

**| |**

**| |----nuttx**

**| | |**

**| | |----tools**

**| |**

**| |----apps**

**| |**

**| |**

**| |**

**| |----misc**

**| | |**

**| | |----kconfig-frontends**

|  |  |
| --- | --- |
| **Get the nuttx os part** | |
| Get latest **nuttx** from [**http://downloads.sourceforge.net/project/nuttx/nuttx/nuttx-7.21/nuttx-7.21.tar.gz**](http://downloads.sourceforge.net/project/nuttx/nuttx/nuttx-7.21/nuttx-7.21.tar.gz) | |
| Then **extract** the archive below **nx** in the file structure and rename the resultant | **mv** **nuttx-7.21** **nuttx** |
| Change all the shell scripts below to be executable | **chmod 755 \*.sh** |
| **Get the nuttx apps part** | |
| Get latest **apps** from [**http://softlayer-dal.dl.sourceforge.net/project/nuttx/nuttx/nuttx-7.21/apps-7.21.tar.gz**](http://softlayer-dal.dl.sourceforge.net/project/nuttx/nuttx/nuttx-7.21/apps-7.21.tar.gz) | |
| Then **extract** the archive below **nx** in the file structure and rename the resultan t… | **mv apps-7.21** **apps** |
| Change all the shell scripts below to be executable | **chmod 755 \*.sh** |
| **Get the kconfig\_frontends tool configure part** | |
| Get latest kconfig-frontends from [**http://ymorin.is-a-geek.org/download/kconfig-frontends/**](http://ymorin.is-a-geek.org/download/kconfig-frontends/) | |
| then **extract** the archive & rename the folder | **mv kconfig-frontends\_3.10.0.0 kconfig-frontends** |
| Get further dependencies for  **kconfig-frontends** | **sudo apt-get install gperf libncurses5-dev flex bison** |
| Change directory to  **kconfig-frontends** | **cd "your local path"/nx/misc/kconfig-frontends/** |
| Then run the configure binary executable | **./configure** |
| Then build kconfig-frontends | **make** |
| Then install kconfig-frontends | **sudo make install** |
| Now go back to configure nuttx for the target board | **cd nx/nuttx/tools** |
| Then run the configuration script | **./configure.sh stm32f746g-Discovery/nsh** |
| Then go back up one directory level | **cd ..** |
| Then run the kconfig-frontends tool | **make menuconfig** |
| …this brings up a graphical interface 🡪  From the graphical interface 🡪  🡪  🡪 | [kconfig-frontends main view](http://paregov.net/images/manuals/nuttx/kconfig-frontends-main.png) |
| Build Host Platform should be set to Linux |
| System Type should be Generic GNU EABI toolchain under Linux (or other POSIX environment) |
| Make sure these settings are saved before exiting. |
| **Now compile the configured nuttx setup** | |
| Add to compiler path to env | **export PATH=$PATH:/usr/local/arm/gcc-arm-none-eabi-5\_4-2016q3/bin** |
| Finally build the Nuttx image | **make** |

Appendix

README

^^^^^^

o Environments

- Installing Cygwin

- Ubuntu Bash under Windows 10

o Installation

- Download and Unpack

- Semi-Optional apps/ Package

- Installation Directories with Spaces in the Path

- Downloading from Repositories

- Related Repositories

- Notes about Header Files

o Configuring NuttX

- Instantiating "Canned" Configurations

- Refreshing Configurations

- NuttX Configuration Tool

- Finding Selections in the Configuration Menus

- Reveal Hidden Configuration Options

- Make Sure that You on on the Right Platform

- Comparing Two Configurations

- Incompatibilities with Older Configurations

- NuttX Configuration Tool under DOS

o Toolchains

- Cross-Development Toolchains

- NuttX Buildroot Toolchain

o Shells

o Building NuttX

- Building

- Re-building

- Build Targets and Options

- Native Windows Build

- Installing GNUWin32

o Cygwin Build Problems

- Strange Path Problems

- Window Native Toolchain Issues

o Documentation

ENVIRONMENTS

^^^^^^^^^^^^

NuttX requires a POSIX development environment such as you would find under

Linux or OSX. NuttX may be also be installed and built on Windows system

if you also provide such a POSIX development environment. Options for a

POSIX development environment under Windows include:

- An installation of Linux on a virtual machine (VM) in Windows. I have

not been happy using a VM myself. I have had stability problems with

open source VMs and commercial VMs cost more than I want to spend.

Sharing files with Linux running in a VM is awkward; sharing devices

connected to the Windows box with Linux in a VM is, at the very least,

confusing; Using Windows tools (such as Segger J-Link) with files

built under the Linux VM is not a possibility.

- The Cygwin environment. Instructions for installation of Cygwin on a

Windows system are provided in the following paragraph, "Installing

Cygwin". Cygwin is a mature, well-tested, and very convenient

environment. It is especially convenient if you need to

integrate with Windows tools and files. Downsides are that the

installation time is very long and the compile times are slow.

- Ubuntu/Bash shell under Windows 10. This is a new option under

Windows 10. See the section "Ubuntu Bash under Windows 10" below.

This is an improvement over Cygwin if your concern is compile time;

its build performance is comparable to native Linux, certainly better

than the Cygwin build time. It also installs in a tiny fraction of

the time as Cygwin, perhaps 20 minutes for the basic Ubuntu install

(vs. more than a day for the complete Cygwin install).

- The MSYS environment. I have no experience using the MSYS environment

and that configuration will not be discussed in this README file.

See http://www.mingw.org/wiki/MSYS if you are interested in

using MSYS. People report to me that they have used MSYS

successfully. I suppose that the advantages of the MSYS environment

is that it is closer to a native Windows environment and uses only a

minimal of add-on POSIX-land tools.

- NuttX can also be installed and built on a native Windows system, but

with some potential tool-related issues (see the discussion "Native

Windows Build" under "Building NuttX" below). GNUWin32 is used to

provide compatible native windows tools.

Installing Cygwin

-----------------

Installing Cygwin on your Windows PC is simple, but time consuming. See

http://www.cygwin.com/ for installation instructions. Basically you just

need to download a tiny setup.exe program and it does the real, network

installation for you.

Some Cygwin installation tips:

1. Install at C:\cygwin

2. Install EVERYTHING: "Only the minimal base packages from the

Cygwin distribution are installed by default. Clicking on categories

and packages in the setup.exe package installation screen will

provide you with the ability to control what is installed or updated.

Clicking on the "Default" field next to the "All" category will

provide you with the opportunity to install every Cygwin package.

Be advised that this will download and install hundreds of megabytes

to your computer."

If you use the "default" installation, you will be missing many

of the Cygwin utilities that you will need to build NuttX. The

build will fail in numerous places because of missing packages.

NOTE: You don't really have to install EVERYTHING but I cannot

answer the question "Then what should I install?" I don't know

the answer to that and so will continue to recommend installing

EVERYTHING.

You should certainly be able to omit "Science", "Math", and

"Publishing". You can try omitting KDE, Gnome, GTK, and other

graphics packages if you don't plan to use them.

Perhaps a minimum set would be those packages listed below for the

"Ubuntu Bash under Windows 10" installation?

After installing Cygwin, you will get lots of links for installed

tools and shells. I use the RXVT native shell. It is fast and reliable

and does not require you to run the Cygwin X server (which is neither

fast nor reliable). Unless otherwise noted, the rest of these

instructions assume that you are at a bash command line prompt in

either Linux or in Cygwin shell.

UPDATE: The last time I installed EVERTHING, the download was

about 5GiB. The server I selected was also very slow so it took

over a day to do the whole install!

Ubuntu Bash under Windows 10

----------------------------

A better version of a command-line only Ubuntu under Windows 10 (beta)

has recently been made available from Microsoft.

Installation

------------

Installation instructions abound on the Internet complete with screen

shots. I will attempt to duplicate those instructions in full here.

Here are the simplified installation steps:

- Open "Settings".

- Click on "Update & security".

- Click on "For Developers".

- Under "Use developer features", select the "Developer mode" option to

setup the environment to install Bash.

- A message box should pop up. Click "Yes" to turn on developer mode.

- After the necessary components install, you'll need to restart your

computer.

Once your computer reboots:

- Open "Control Panel".

- Click on "Programs".

- Click on "Turn Windows features on or off".

- A list of features will pop up, check the "Windows Subsystem for Linux

(beta)" option.

- Click OK.

- Once the components installed on your computer, click the "Restart

now" button to complete the task.

After your computer restarts, you will notice that Bash will not appear in

the "Recently added" list of apps, this is because Bash isn't actually

installed yet. Now that you have setup the necessary components, use the

following steps to complete the installation of Bash:

- Open "Start", do a search for bash.exe, and press "Enter".

- On the command prompt, type y and press Enter to download and install

Bash from the Windows Store. This will take a while.

- Then you'll need to create a default UNIX user account. This account

doesn't have to be the same as your Windows account. Enter the

username in the required field and press Enter (you can't use the

username "admin").

- Close the "bash.exe" command prompt.

Now that you completed the installation and setup, you can open the Bash

tool from the Start menu like you would with any other app.

Accessing Windows Files from Ubuntu

-----------------------------------

File systems will be mounted under "/mnt" so for example "C:\Program Files"

appears at "/mnt/c/Program Files". This is as opposed to Cgwin where

the same directory would appear at "/cygdrive/c/Program Files".

With these differences (perhaps a few other Windows quirks) the Ubuntu

install works just like Ubuntu running natively on your PC.

A good tip for file sharing is to use symbolic links within your Ubuntu

home directory. For example, suppose you have your "projects" directory

at C:\Documents\projects. Then you can set up a link to the projects/

directory in your Ubuntu directory like:

$ ln -s /mnt/c/Documents/projects projects

Accessing Ubuntu Files From Windows

-----------------------------------

In Ubuntu Userspace for Windows, the Ubuntu file system root directory is

at:

%localappdata%\lxss\rootfs

Or

C:\Users\Username\AppData\Local\lxss\rootfs

However, I am unable to see my files under the rootfs\home directory.

After some looking around, I find the home directory

%localappdata%\lxss\home.

With that trick access to the /home directory, you should actually be

able to use Windows tools outside of the Ubuntu sandbox with versions of

NuttX built within the sandbox using that path.

Executing Windows Tools from Ubuntu

-----------------------------------

You can also execute Windows tools from within the Ubuntu sandbox:

$ /mnt/c/Program\ Files\ \(x86\)/Microchip/xc32/v1.43/bin/xc32-gcc.exe --version

Unable to translate current working directory. Using C:\WINDOWS\System32

xc32-gcc.exe (Microchip Technology) 4.8.3 MPLAB XC32 Compiler v1.43 Build date: Mar 1 2017

...

The error message indicates that there are more issues: You cannot mix

Windows tools that use Windows style paths in an environment that uses

POSIX paths. I think you would have to use Linux tools only from within

the Ubuntu sandbox.

Install Ubuntu Software

-----------------------

Use "sudo apt-get install <package name>". As examples, this is how

you would get GIT:

$ sudo apt-get install git

This will get you a compiler for your host PC:

$ sudo apt-get install gcc

This will get you an ARM compiler for your target:

$ sudo apt-get install gcc-arm-none-eabi

NOTE: That is just an example. I am not sure if apt-get will give you a

current or usable compiler. You should carefully select your toolchain

for the needs of your project.]

You will also need to the get the kconfig-frontends configuration as

described below under "NuttX Configuration tool". In order build the

kconfig-frontends configuration tool you will also need: make, gperf,

flex, bison, and libncurses-dev.

That is enough to do a basic NuttX build.

Integrating with Windows Tools

------------------------------

If you want to integrate with Windows native tools, then you would need

deal with the same kind of craziness as with integrating Cygwin with

native toolchains, see the section "Cygwin Build Problems" below.

However, there is currently no build support for using Windows native

tools with Ubuntu under Windows. This tool combination is made to work

with Cygwin through the use of the 'cygpath -w' tool that converts paths

from say '/cydrive/c/Program Files' to 'C:\Program Files'. There is,

however, no corresponding tool to convert '/mnt/c/Program Files' in the

Ubuntu environment.

Graphics Support

----------------

The Ubuntu version support by Microsoft is a command-line only version.

There is no support for Linux graphics utilities.

This limititation is not a limitation of Ubuntu, however, only in what

Microsoft is willing to support. If you install a X-Server, then you

can also use basic graphics utilities. See for example:

http://www.howtogeek.com/261575/how-to-run-graphical-linux-desktop-applications-from-windows-10s-bash-shell/

Many Linux graphics programs would, however, also require a graphics

framework like GTK or Qt. So this might be a trip down the rabbit hole.

INSTALLATION

^^^^^^^^^^^^

There are two ways to get NuttX: You may download released, stable

tarballs from wither the Bitbucket or Sourceforge download locations.

Or you may get NuttX by cloning the Bitbucket GIT repositories. Let's

consider the released tarballs first:

Download and Unpack

-------------------

Download and unpack the NuttX tarball. If you are reading this, then

you have probably already done that. After unpacking, you will end

up with a directory called nuttx-version (where version is the NuttX

version number). You might want to rename that directory nuttx to

match the various instructions in the documentation and some scripts

in the source tree.

Download locations:

https://bitbucket.org/nuttx/nuttx/downloads

https://sourceforge.net/projects/nuttx/files/nuttx/

Semi-Optional apps/ Package

---------------------------

All NuttX libraries and example code used to be in included within

the NuttX source tree. As of NuttX-6.0, this application code was

moved into a separate tarball, the apps tarball. If you are just

beginning with NuttX, then you will want to download the versioned

apps tarball along with the NuttX tarball. If you already have your

own product application directory, then you may not need the apps

tarball.

It is call "Semi-optional" because if you don't have some apps/

directory, NuttX will \*fail\* to build! You do not necessarily need

to use the NuttX apps tarball but may, instead, provide your own

custom application directory. Such a custom directory would need

to include a valid Makefile to support the build and a valid Kconfig

file to support the configuration. More about these file later.

Download then unpack the apps tarball in the same directory where you

unpacked the NuttX tarball. After you unpack the apps tarball, you

will have a new directory called apps-version (where the version

should exactly match the version of the NuttX tarball). Again, you

might want to rename the directory to simply apps/ to match what

you read in the documentation

After unpacking (and renaming) the apps tarball, you will have two

directories side by side like this:

|

+----+----+

| |

nuttx/ apps/

This is important because the NuttX build will expect to find the

apps directory in that (default) location. That default location

can be changed by modifying your NuttX configuration file, but that

is another story.

Installation Directories with Spaces in the Path

------------------------------------------------

**The nuttx build directory should reside in a path that contains no**

**spaces in any higher level directory name**. For example, under

Cygwin, your home directory might be formed from your first and last

names like: "/home/First Last". That will cause strange errors when

the make system tries to build.

[Actually, that problem is probably not to difficult to fix. Some

Makefiles probably just need some paths within double quotes]

**I work around spaces in the home directory name, by creating a**

**new directory that does not contain any spaces, such as /home/nuttx.**

**Then I install NuttX in /home/nuttx and always build from**

**/home/nuttx/nuttx-code.**

Downloading from Repositories

-----------------------------

Cloning the Repository

The current NuttX du jour is available in from a GIT repository. Here are

instructions for cloning the core NuttX RTOS (corresponding to the nuttx

tarball discussed above)::

git clone https://bitbucket.org/nuttx/nuttx.git nuttx

And the semi-optional apps/ application directory and be cloned like:

git clone https://bitbucket.org/nuttx/apps.git apps

That will give you the same directory structure like this:

|

+----+----+

| |

nuttx/ apps/

Configuring the Clones

The following steps need to be performed for each of the repositories.

After changing to the clone directory:

Set your identity:

git config --global user.name "My Name"

git config --global user.email my.name@example.com

Colorized diffs are much easier to read:

git config --global color.branch auto

git config --global color.diff auto

git config --global color.interactive auto

git config --global color.status auto

Checkout other settings

git config --list

Cloning NuttX Inside Cygwin

If you are cloning the NuttX repository, it is recommended to avoid

automatic end of lines conversions by git. These conversions may break

some scripts like configure.sh. Before cloning, do the following:

git config --global core.autocrlf false

Related Repositories

--------------------

These are standalone repositories:

\* https://bitbucket.org/nuttx/apps

This directory holds an optional package of applications and libraries

can be used with the NuttX RTOS. There is a README.txt file there that

will provide a more information about that package.

\* https://bitbucket.org/nuttx/nxwidgets

This is the NuttX C++ graphics support. This includes NxWM, the tiny

NuttX Window Manager.

\* https://bitbucket.org/nuttx/uclibc

This repository contains a version of the uClibc++ C++ library. This code

originates from http://cxx.uclibc.org/ and has been adapted for NuttX by the

RGMP team (http://rgmp.sourceforge.net/wiki/index.php/Main\_Page).

\* https://bitbucket.org/nuttx/buildroot

A environment that you can to use to build a custom, NuttX GNU toolchain.

\* https://bitbucket.org/nuttx/tools

There are snapshots of some tools here that you will need to work with

NuttX: kconfig-frontends, genromfs, and others.

\* https://bitbucket.org/nuttx/drivers

A few drivers that are not integrated into the main NuttX source tree due

to licensing issues.

\* https://bitbucket.org/nuttx/pascal

Yes, this really is a Pascal compiler. The Pascal p-code run-time and

pcode debugger can be built as a part of NuttX.

Notes about Header Files

------------------------

Other C-Library Header Files.

When a GCC toolchain is built, it must be built against a C library.

The compiler together with the contents of the C library completes the

C language definition and provides the complete C development

environment. NuttX provides its own, built-in C library. So the

complete, consistent C language definition for use with NuttX comes from

the combination of the compiler and the header files provided by the

NuttX C library.

When a GCC toolchain is built, it incorporates the C library header

files into the compiler internal directories and, in this way, the C

library really becomes a part of the toolchain. If you use the NuttX

buildroot toolchain as described below under under "NuttX Buildroot

Toolchain", your GCC toolchain will build against the NuttX C library

and will incorporate the NuttX C library header files as part of the

toolchain.

If you use some other, third-party tool chain, this will not be the

case, however. Those toolchains were probably built against some

other, incompatible C library distribution (such as newlib). Those

tools will have incorporated the incompatible C library header files

as part of the toolchain. These incompatible header files must \*not\*

be used with NuttX because the will conflict with definitions in the

NuttX built-in C-Library. For such toolchains that include header

files from a foreign C-Library, NuttX must be compiled without using

the standard header files that are distributed with your toolchain.

This prevents including conflicting, incompatible header files such

as stdio.h.

The math.h and stdarg.h are probably the two most trouble some header

files to deal with. These troublesome header files are discussed in

more detail below.

Header Files Provided by Your Toolchain.

Certain header files, such as setjmp.h, stdarg.h, and math.h, may still

be needed from your toolchain and your compiler may not, however, be able

to find these if you compile NuttX without using standard header files

(ie., with -nostdinc). If that is the case, one solution is to copy

those header file from your toolchain into the NuttX include directory.

Duplicated Header Files.

There are also a few header files that can be found in the nuttx/include

directory which are duplicated by the header files from your toolchain.

stdint.h and stdbool.h are examples. If you prefer to use the stdint.h

and stdbool.h header files from your toolchain, those could be copied

into the nuttx/include/ directory. Using most other header files from

your toolchain would probably cause errors.

math.h

Even though you should not use a foreign C-Library, you may still need

to use other, external libraries with NuttX. In particular, you may

need to use the math library, libm.a. NuttX supports a generic, built-in

math library that can be enabled using CONFIG\_LIBM=y. However, you may

still want to use a higher performance external math library that has

been tuned for your CPU. Sometimes such such tuned math libraries are

bundled with your toolchain.

The math libary header file, math.h, is a then special case. If you do

nothing, the standard math.h header file that is provided with your

toolchain will be used.

If you have a custom, architecture specific math.h header file, then

that header file should be placed at arch/<cpu>/include/math.h. There

is a stub math.h header file located at include/nuttx/lib/math.h. This stub

header file can be used to "redirect" the inclusion to an architecture-

specific math.h header file. If you add an architecture specific math.h

header file then you should also define CONFIG\_ARCH\_MATH\_H=y in your

NuttX Configuration file. If CONFIG\_ARCH\_MATH\_H is selected, then the

top-level Makefile will copy the stub math.h header file from

include/nuttx/lib/math.h to include/math.h where it will become the system

math.h header file. The stub math.h header file does nothing other

than to include that architecture-specific math.h header file as the

system math.h header file.

float.h

If you enable the generic, built-in math library, then that math library

will expect your toolchain to provide the standard float.h header file.

The float.h header file defines the properties of your floating point

implementation. It would always be best to use your toolchain's float.h

header file but if none is available, a default float.h header file will

provided if this option is selected. However, there is no assurance that

the settings in this float.h are actually correct for your platform!

stdarg.h

In most cases, the correct version of stdarg.h is the version provided

with your toolchain. However, sometimes there are issues with with

using your toolchains stdarg.h. For example, it may attempt to draw in

header files that do not exist in NuttX or perhaps the header files that

is uses are not compatible with the NuttX header files. In those cases,

you can use an architecture-specific stdarg.h header file by defining

CONFIG\_ARCH\_STDARG\_H=y.

See the discussion above for the math.h header. This setting works

exactly the same for the stdarg.h header file.

CONFIGURING NUTTX

^^^^^^^^^^^^^^^^^

Instantiating "Canned" Configurations

-------------------------------------

"Canned" NuttX configuration files are retained in:

configs/<board-name>/<config-dir>

Where <board-name> is the name of your development board and <config-dir>

is the name of the sub-directory containing a specific configuration for

that board. Configuring NuttX requires only copying three files from the

<config-dir> to the directory where you installed NuttX (TOPDIR) (and

sometimes one additional file to the directory the NuttX application

package (APPSDIR)):

Copy configs/<board-name>/<config-dir>/Make.def to ${TOPDIR}/Make.defs

Make.defs describes the rules needed by you tool chain to compile

and link code. You may need to modify this file to match the

specific needs of your toolchain.

Copy configs/<board-name>/<config-dir>/defconfig to ${TOPDIR}/.config

The defconfig file holds the actual build configuration. This

file is included by all other make files to determine what is

included in the build and what is not. This file is also used

to generate a C configuration header at include/nuttx/config.h.

Copy other, environment-specic files to ${TOPDIR

This might include files like .gdbinit or IDE configuration files

like .project or .cproject.

General information about configuring NuttX can be found in:

${TOPDIR}/configs/README.txt

${TOPDIR}/configs/<board-name>/README.txt

There is a configuration script in the tools/ directory that makes does

all of the above steps for you. It is used as follows:

cd ${TOPDIR}/tools

./configure.sh <board-name>/<config-dir>

There is an alternative Windows batch file that can be used in the

windows native environment like:

cd ${TOPDIR}\tools

configure.bat <board-name>\<config-dir>

And, to make sure that other platform is supported, there is also a

C program at tools/configure.c that can be compiled to establish the

board configuration.

See tools/README.txt for more information about these scripts.

Refreshing Configurations

-------------------------

Configurations can get out of date. As new configuration settings are

added or removed or as dependencies between configuration settings

change, the contents of a default configuration can become out of synch

with the build systems. Hence, it is a good practice to "refresh" each

configuration after configuring and before making. To refresh the

configuration, use the NuttX Configuration Tool like this:

make oldconfig

AFTER you have instantiated the NuttX configuration as described above.

The configuration step copied the .config file into place in the top-level

NuttX directory; 'make oldconfig' step will then operate on that .config

file to bring it up-to-date.

If you configuration is out of date, you will be prompted by 'make oldconfig'

to resolve the issues detected by the configuration tool, that is, to

provide values for the new configuration options in the build system. Doing

this can save you a lot of problems down the road due to obsolete settings in

the default board configuration file. The NuttX configuration tool is

discussed in more detail in the following paragraph.

Confused about what the correct value for a new configuration item should

be? Enter ? in response to the 'make oldconfig' prompt and it will show

you the help text that goes with the option.

If you don't want to make any decisions are are willing to just accept the

recommended default value for each new configuration item, an even easier

way is:

make oldefconfig

The olddefconfig target will simply bring you configuration up to date with

the current Kconfig files, setting any new options to the default value.

No questions asked.

NuttX Configuration Tool

------------------------

An automated tool has been incorported to support re-configuration

of NuttX. This automated tool is based on the kconfig-frontends

application available at http://ymorin.is-a-geek.org/projects/kconfig-frontends

(A snapshot of this tool is also available fromo the tools repository at

https://bitbucket.org/nuttx/tools). This application provides a tool

called 'kconfig-mconf' that is used by the NuttX top-level Makefile.

The following make target is provided:

make menuconfig

This make target will bring up NuttX configuration menus.

WARNING: Never do 'make menuconfig' on a configuration that has

not been converted to use the kconfig-frontends tools! This will

damage your configuration (see

http://www.nuttx.org/doku.php?id=wiki:howtos:convertconfig).

How do we tell a new configuration from an old one? See "Incompatibilities

with Older Configurations" below.

The 'menuconfig' make target depends on two things:

1. The Kconfig configuration data files that appear in almost all

NuttX directories. These data files are the part that is still

under development (patches are welcome!). The Kconfig files

contain configuration information for the configuration settings

relevant to the directory in which the Kconfig file resides.

NOTE: For a description of the syntax of this configuration file,

see kconfig-language.txt in the tools repository at

https://bitbucket.org/nuttx/tools

2. The 'kconfig-mconf' tool. 'kconfig-mconf' is part of the

kconfig-frontends package. You can download that package from

the website http://ymorin.is-a-geek.org/projects/kconfig-frontends

or you can use the snapshot in the tools repository at

https://bitbucket.org/nuttx/tools.

Building kconfig-frontends under Linux may be as simple as

'configure; make; make install' but there may be some build

complexities, especially if you are building under Cygwin. See

the more detailed build instructions in the top-level README.txt

file of the tools repository at https://bitbucket.org/nuttx/tools.

The 'make install' step will, by default, install the 'kconfig-mconf'

tool at /usr/local/bin/mconf. Where ever you choose to

install 'kconfig-mconf', make certain that your PATH variable includes

a path to that installation directory.

The kconfig-frontends tools will not build in a native Windows

environment directly "out-of-the-box". For the Windows native

case, you should should the modified version of kconfig-frontends

that can be found at

http://uvc.de/posts/linux-kernel-configuration-tool-mconf-under-windows.html

The basic configuration order is "bottom-up":

- Select the build environment,

- Select the processor,

- Select the board,

- Select the supported peripherals

- Configure the device drivers,

- Configure the application options on top of this.

This is pretty straight forward for creating new configurations

but may be less intuitive for modifying existing configurations.

If you have an environment that supports the Qt or GTK graphical systems

(probably KDE or gnome, respectively, or Cygwin under Windows with Qt or

GTK installed), then you can also build the graphical kconfig-frontends,

kconfig-qconf and kconfig-gconf. In these case, you can start the

graphical configurator with either:

make qconfig

or

make gconfig

Some keyboard shortcus supported by kconfig-mconf, the tool that runs

when you do 'make menuconfig':

- '?' will bring up the mconfig help display.

- '/' can be used find configuration selections.

- 'Z' can be used to reveal hidden configuration options

These last to shortcuts are described further in the following

paragraphs.

Finding Selections in the Configuration Menus

---------------------------------------------

The NuttX configuration options have gotten complex and it can be very

difficult to find options in the menu trees if you are not sure where

to look. The "basic configuration order" describe above can help to

narrow things down.

But if you know exactly what configuration setting you want to select,

say CONFIG\_XYZ, but not where to find it, then the 'make memconfig'

version of the tool offers some help: By pressing the '/' key, the

tool will bring up a menu that will allow you to search for a

configuration item. Just enter the string CONFIG\_XYZ and press 'ENTER'.

It will show you not only where to find the configuration item, but

also all of the dependencies related to the configuration item.

Reveal Hidden Configuration Options

-----------------------------------

If you type 'Z', then kconfig-mconf will change what is displayed.

Normally, only enabled features that have all of their dependencies met

are displayed. That is, of course, not very useful if you would like to

discover new options or if you are looking for an option and do not

realize that the dependencies have not yet been selected and, hence, it

is not displayed.

But if you enter 'Z', then every option will be shown, whether or not its

dependencies have been met. You can the see everything that could be

selected with the right dependency selections. These additional options

will be shown the '-' for the selection and for the value (since it

cannot be selected and has no value). About all you do is to select

the <Help> option to see what the dependencies are.

Make Sure that You on on the Right Platform

-------------------------------------------

Saved configurations may run on Linux, Cygwin (32- or 64-bit), or other

platforms. The platform characteristics can be changed use 'make

menuconfig'. Sometimes this can be confusing due to the differences

between the platforms. Enter sethost.sh

sethost.sh is a simple script that changes a configuration to your

host platform. This can greatly simplify life if you use many different

configurations. For example, if you are running on Linux and you

configure like this:

$ cd tools

$ ./configure.sh board/configuration

$ cd ..

The you can use the following command to both (1) make sure that the

configuration is up to date, AND (2) the configuration is set up

correctly for Linux:

$ tools/sethost.sh -l

Or, if you are on a Windows/Cygwin 64-bit platform:

$ tools/sethost.sh -w

Other options are available from the help option built into the

script. You can see all options with:

$ tools/sethost.sh -h

Comparing Two Configurations

----------------------------

If you try to compare to configurations using 'diff', you will probably

not be happy with the result. There are superfluous things added to

the configuration files that makes comparisons with the human eye

difficult.

There is a tool at nuttx/tools/cmpconfig.c that can be build to simplify

these comparisons. The output from this difference tools will show only

the meaningful differences between two configuration files. This tools

built as follows:

cd nuttx/tools

make -f Makefile.host

This will crate a program called 'cmpconfig' or 'comconfig.exe' on Windows.

Why would you want to compare two configuration files? Here are a few

of reasons why I do this:

1. When I create a new configuration I usually base it on an older

configuration and I want to know, "What are the options that I need to

change to add the new feature to the older configurations?" For example,

suppose that I have a boardA/nsh configuration and I want to create a

boardA/nxwm configuration. Suppose I already have boardB/nsh and

boardB/nxwm configurations. Then by comparing the boardB/nsh with the

boardB/nxwm I can see the modifications that I would need to make to my

boardA/nsh to create a new boardA/nxwm.

2. But the most common reason that I use the 'cmpconfig' program to to

check the results of "refreshing" a configuration with 'make oldconfig'

(see the paragraph "Refreshing Configurations" above). The 'make

oldconfig' command will make changes to my configuration and using

'cmpconfig', I can see precisely what those changes were and if any

should be of concern to me.

3. The 'cmpconfig' tool can also be useful when converting older, legacy

manual configurations to the current configurations based on the

kconfig-frontends tools. See the following paragraph.

Incompatibilities with Older Configurations

-------------------------------------------

\*\*\*\*\* WARNING \*\*\*\*\*

The current NuttX build system supports \*only\* the new configuration

files generated using the kconfig-frontends tools. Support for the

older, legacy, manual configurations was eliminated in NuttX 7.0; all

configuration must now be done using the kconfig-frontends tool. The

older manual configurations and the new kconfig-frontends configurations

are not compatible. Old legacy configurations can \*not\* be used

with the kconfig-frontends tool and, hence, cannot be used with releases

of NuttX 7.0 and beyond:

If you run 'make menuconfig' with a legacy configuration the resulting

configuration will probably not be functional.

Q: How can I tell if a configuration is a new kconfig-frontends

configuration or an older, manual configuration?

A: Only old, manual configurations will have an appconfig file

Q: How can I convert a older, manual configuration into a new,

kconfig-frontends toolchain.

A: Refer to http://www.nuttx.org/doku.php?id=wiki:howtos:convertconfig

\*\*\*\*\* WARNING \*\*\*\*\*

As described above, whenever you use a configuration, you really should

always refresh the configuration the following command \*before\* you make

NuttX:

make oldconfig

This will make sure that the configuration is up-to-date in the event that

it has lapsed behind the current NuttX development (see the paragraph

"Refreshing Configurations" above). But this only works with \*new\*

configuration files created with the kconfig-frontends tools

Never do 'make oldconfig' (OR 'make menuconfig') on a configuration that

has not been converted to use the kconfig-frontends tools! This will

damage your configuration (see

http://www.nuttx.org/doku.php?id=wiki:howtos:convertconfig).

NuttX Configuration Tool under DOS

----------------------------------

Recent versions of NuttX support building NuttX from a native Windows

console window (see "Native Windows Build" below). But kconfig-frontends

is a Linux tool. At one time this was a problem for Windows users, but

now there is a specially modified version of the kconfig-frontends tools

that can be used:

http://uvc.de/posts/linux-kernel-configuration-tool-mconf-under-windows.html

It is also possible to use the version of kconfig-frontends built

under Cygwin outside of the Cygwin "sandbox" in a native Windows

environment:

1. You can run the configuration tool using Cygwin. However, the

Cygwin Makefile.win will complain so to do this will, you have

to manually edit the .config file:

a. Delete the line: CONFIG\_WINDOWS\_NATIVE=y

b. Change the apps/ directory path, CONFIG\_APPS\_DIR to use Unix

style delimiters. For example, change "..\apps" to "../apps"

And of course, after you use the configuration tool you need to

restore CONFIG\_WINDOWS\_NATIVE=y and the correct CONFIG\_APPS\_DIR.

2) You can, with some effort, run the Cygwin kconfig-mconf tool

directly in the Windows console window. In this case, you do not

have to modify the .config file, but there are other complexities:

a. You need to temporarily set the Cgywin directories in the PATH

variable then run kconfig-mconf manually like:

kconfig-mconf Kconfig

There is a Windows batch file at tools/kconfig.bat that automates

these steps:

tools/kconfig menuconfig

b. There is an issue with accessing DOS environment variables from

the Cygwin kconfig-mconf running in the Windows console. The

following change to the top-level Kconfig file seems to work

around these problems:

config APPSDIR

string

- option env="APPSDIR"

+ default "../apps"

TOOLCHAINS

^^^^^^^^^^

Cross-Development Toolchains

----------------------------

In order to build NuttX for your board, you will have to obtain a cross-

compiler to generate code for your target CPU. For each board,

configuration, there is a README.txt file (at configs/<board-name>/README.txt).

That README file contains suggestions and information about appropriate

tools and development environments for use with your board.

In any case, the PATH environment variable will need to be updated to

include the loction where the build can find the toolchain binaries.

NuttX Buildroot Toolchain

-------------------------

For many configurations, a DIY set of tools is available for NuttX. These

tools can be downloaded from the NuttX Bitbucket.org file repository. After

unpacking the buildroot tarball, you can find instructions for building

the tools in the buildroot/configs/README.txt file.

Check the README.txt file in the configuration director for your board

to see if you can use the buildroot toolchain with your board (this

README.txt file is located in configs/<board-name>/README.txt).

This toolchain is available for both the Linux and Cygwin development

environments.

Advantages: (1) NuttX header files are built into the tool chain,

and (2) related support tools like NXFLAT tools, the ROMFS

genromfs tools, and the kconfig-frontends tools can be built into your

toolchain.

Disadvantages: This tool chain is not was well supported as some other

toolchains. GNU tools are not my priority and so the buildroot tools

often get behind. For example, until recently there was no EABI support

in the NuttX buildroot toolchain for ARM.

NOTE: For Cortex-M3/4, there are OABI and EABI versions of the buildroot

toolchains. If you are using the older OABI toolchain the prefix for

the tools will be arm-nuttx-elf-; for the EABI toolchain the prefix will

be arm-nuttx-eabi-. If you are using the older OABI toolchain with

an ARM Cortex-M3/4, you will need to set CONFIG\_ARMV7M\_OABI\_TOOLCHAIN

in the .config file in order to pick the right tool prefix.

If the make system ever picks the wrong prefix for your toolchain, you

can always specify the prefix on the command to override the default

like:

make CROSSDEV=arm-nuttx-elf

SHELLS

^^^^^^

The NuttX build relies on some shell scripts. Some are inline in the

Makefiles and many are executable scripts in the tools/. directory. The

scripts were all developed using bash and many contain bash shell

dependencies.

Most of the scripts begin with #!/bin/bash to specifically select the

bash shell. Some still have #!/bin/sh but I haven't heard any complaints

so these must not have bash dependencies.

There are two shell issues that I have heard of:

1. Linux where /bin/sh refers to an incompatible shell (like ksh or csh).

In this case, bash is probably available and the #!/bin/bash at the

beginning of the file should do the job. If any scripts with #!/bin/sh

fail, try changing that to #!/bin/bash and let me know about the change.

2. FreeBSD with the Bourne Shell and no bash shell.

The other, reverse case has also been reported on FreeBSD setups that

have the Bourne shell, but not bash. In this base, #!/bin/bash fails

but #!/bin/sh works okay. My recommendation in this case is to create

a symbolic link at /bin/bash that refers to the Bourne shell.

There may still be issues, however, with certain the bash-centric scripts

that will require modifications.

BUILDING NUTTX

^^^^^^^^^^^^^^

Building

--------

NuttX builds in-place in the source tree. You do not need to create

any special build directories. Assuming that your Make.defs is setup

properly for your tool chain and that PATH environment variable contains

the path to where your cross-development tools are installed, the

following steps are all that are required to build NuttX:

cd ${TOPDIR}

make

At least one configuration (eagle100) requires additional command line

arguments on the make command. Read ${TOPDIR}/configs/<board-name>/README.txt

to see if that applies to your target.

Re-building

-----------

Re-building is normally simple -- just type make again.

But there are some things that can "get you" when you use the Cygwin

development environment with Windows native tools. The native Windows

tools do not understand Cygwin's symbolic links, so the NuttX make system

does something weird: It copies the configuration directories instead of

linking to them (it could, perhaps, use the NTFS 'mklink' command, but it

doesn't).

A consequence of this is that you can easily get confused when you edit

a file in one of the linked (i.e., copied) directories, re-build NuttX,

and then not see your changes when you run the program. That is because

build is still using the version of the file in the copied directory, not

your modified file!

Older versions of NuttX did not support dependiencies in this

configuration. So a simple work around this annoying behavior in this

case was the following when you re-build:

make clean\_context all

This 'make' command will remove of the copied directories, re-copy them,

then make NuttX.

However, more recent versions of NuttX do support dependencies for the

Cygwin build. As a result, the above command will cause everything to be

rebuilt (beause it removes and will cause recreating the

include/nuttx/config.h header file). A much less gracefully but still

effective command in this case is the following for the ARM configuration:

rm -rf arch/arm/src/chip arch/arm/src/board

This "kludge" simple removes the copied directories. These directories

will be re-created when you do a normal 'make' and your edits will then be

effective.

Build Targets and Options

-------------------------

Build Targets:

Below is a summary of the build targets available in the top-level

NuttX Makefile:

all

The default target builds the NuttX executable in the selected output

formats.

clean

Removes derived object files, archives, executables, and temporary

files, but retains the configuration and context files and directories.

distclean

Does 'clean' then also removes all configuration and context files.

This essentially restores the directory structure to its original,

unconfigured stated.

Application housekeeping targets. The APPDIR variable refers to the user

application directory. A sample apps/ directory is included with NuttX,

however, this is not treated as part of NuttX and may be replaced with a

different application directory. For the most part, the application

directory is treated like any other build directory in the Makefile script.

However, as a convenience, the following targets are included to support

housekeeping functions in the user application directory from the NuttX

build directory.

apps\_clean

Perform the clean operation only in the user application directory

apps\_distclean

Perform the distclean operation only in the user application directory.

The apps/.config file is preserved so that this is not a "full" distclean

but more of a configuration "reset."

export

The export target will package the NuttX libraries and header files into

an exportable package. Caveats: (1) These needs some extension for the KERNEL

build. (2) The logic in tools/mkexport.sh only supports GCC and, for example,

explicitly assumes that the archiver is 'ar'

download

This is a helper target that will rebuild NuttX and download it to the target

system in one step. The operation of this target depends completely upon

implementation of the DOWNLOAD command in the user Make.defs file. It will

generate an error an error if the DOWNLOAD command is not defined.

The following targets are used internally by the make logic but can be invoked

from the command under certain conditions if necessary.

depend

Create build dependencies. (NOTE: There is currently no support for build

dependencies under Cygwin using Windows-native toolchains.)

context

The context target is invoked on each target build to assure that NuttX is

properly configured. The basic configuration steps include creation of the

the config.h and version.h header files in the include/nuttx directory and

the establishment of symbolic links to configured directories.

clean\_context

This is part of the distclean target. It removes all of the header files

and symbolic links created by the context target.

Build Options:

Of course, the value any make variable an be overridden from the make command

line. However, there is one particular variable assignment option that may

be useful to you:

V=1

This is the build "verbosity flag." If you specify V=1 on the make command

line, you will see the exact commands used in the build. This can be very

useful when adding new boards or tracking down compile time errors and

warnings (Contributed by Richard Cochran).

Native Windows Build

--------------------

The beginnings of a Windows native build are in place but still not often

used as of this writing. The build was functional but because of lack of

use may find some issues to be resolved with this build configuration.

The windows native build logic initiated if CONFIG\_WINDOWS\_NATIVE=y is

defined in the NuttX configuration file:

This build:

- Uses all Windows style paths

- Uses primarily Windows batch commands from cmd.exe, with

- A few extensions from GNUWin32

In this build, you cannot use a Cygwin or MSYS shell. Rather the build must

be performed in a Windows console window. Here is a better terminal than the

standard issue, CMD.exe terminal: ConEmu which can be downloaded from:

https://sourceforge.net/projects/conemu/ or https://conemu.github.io/ .

Build Tools. The build still relies on some Unix-like commands. I use

the GNUWin32 tools that can be downloaded from http://gnuwin32.sourceforge.net/

using the 'Download all' selection. Individual packages can be download

instead if you know what you are doing and want a faster download (No, I

can't tell you which packages you should or should not download).

Host Compiler: I use the MingGW GCC compiler which can be downloaded from

http://www.mingw.org/. If you are using GNUWin32, then it is recommended

the you not install the optional MSYS components as there may be conflicts.

This capability should still be considered a work in progress because:

(1) It has not been verified on all targets and tools, and

(2) it still lacks some of the creature-comforts of the more mature

environments.

Installing GNUWin32

-------------------

The Windows native build will depend upon a few Unix-like tools that can be

provided either by MSYS or GNUWin32. The GNUWin32 are available from

http://gnuwin32.sourceforge.net/. GNUWin32 provides ports of tools with a

GPL or similar open source license to modern MS-Windows (Microsoft Windows

2000 / XP / 2003 / Vista / 2008 / 7). See

http://gnuwin32.sourceforge.net/packages.html for a list of all of the tools

available in the GNUWin32 package.

The SourceForge project is located here:

http://sourceforge.net/projects/gnuwin32/. The project is still being

actively supported (although some of the Windows ports have gotten very old).

Some commercial toolchains include a subset of the GNUWin32 tools in the

installation. My recommendation is that you download the GNUWin32 tools

directly from the sourceforge.net website so that you will know what you are

using and can reproduce your build environment.

GNUWin32 Installation Steps:

The following steps will download and execute the GNUWin32 installer.

1. Download GetGNUWin32-x.x.x.exe from

http://sourceforge.net/projects/getgnuwin32/files/. This is the

installer. The current version as of this writing is 0.6.3.

2. Run the installer.

3. Accept the license.

4. Select the installation directory. My recommendation is the

directory that contains this README file (<this-directory>).

5. After running GetGNUWin32-0.x.x.exe, you will have a new directory

<this-directory>/GetGNUWin32

Note that the GNUWin32 installer didn't install GNUWin32. Instead, it

installed another, smarter downloader. That downloader is the GNUWin32

package management tool developed by the Open SSL project.

The following steps probably should be performed from inside a DOS shell.

6. Change to the directory created by GetGNUWin32-x.x.x.exe

cd GetGNUWin32

7. Execute the download.bat script. The download.bat script will download

about 446 packages! Enough to have a very complete Linux-like environment

under the DOS shell. This will take awhile. This step only downloads

the packages and the next step will install the packages.

download

8. This step will install the downloaded packages. The argument of the

install.bat script is the installation location. C:\gnuwin32 is the

standard install location:

install C:\gnuwin32

NOTE: This installation step will install \*all\* GNUWin32 packages... far

more than you will ever need. If disc space is a problem for you, you might

need to perform a manual installation of the individual ZIP files that you

will find in the <this directory>/GetGNUWin32/packages directory.

CYGWIN BUILD PROBLEMS

^^^^^^^^^^^^^^^^^^^^^

Strange Path Problems

---------------------

If you see strange behavior when building under Cygwin then you may have

a problem with your PATH variable. For example, if you see failures to

locate files that are clearly present, that may mean that you are using

the wrong version of a tool. For example, you may not be using Cygwin's

'make' program at /usr/bin/make. Try:

$ which make

/usr/bin/make

When you install some toolchains (such as Yargarto or CodeSourcery tools),

they may modify your PATH variable to include a path to their binaries.

At that location, they make have GNUWin32 versions of the tools. So you

might actually be using a version of make that does not understand Cygwin

paths.

The solution is either:

1. Edit your PATH to remove the path to the GNUWin32 tools, or

2. Put /usr/local/bin, /usr/bin, and /bin at the front of your path:

$ export PATH=/usr/local/bin:/usr/bin:/bin:$PATH

Window Native Toolchain Issues

------------------------------

There are many popular Windows native toolchains that may be used with NuttX.

Examples include CodeSourcery (for Windows), devkitARM, and several vendor-

provied toolchains. There are several limitations with using a and Windows

based toolchain in a Cygwin environment. The three biggest are:

1. The Windows toolchain cannot follow Cygwin paths. Path conversions are

performed automatically in the Cygwin makefiles using the 'cygpath' utility

but you might easily find some new path problems. If so, check out 'cygpath -w'

2. Windows toolchains cannot follow Cygwin symbolic links. Many symbolic links

are used in Nuttx (e.g., include/arch). The make system works around these

problems for the Windows tools by copying directories instead of linking them.

But this can also cause some confusion for you: For example, you may edit

a file in a "linked" directory and find that your changes had no effect.

That is because you are building the copy of the file in the "fake" symbolic

directory. If you use a Windows toolchain, you should get in the habit of

making like this:

make clean\_context all

An alias in your .bashrc file might make that less painful. The rebuild

is not a long as you might think because there is no dependency checking

if you are using a native Windows toolchain. That bring us to #3:

General Pre-built Toolchain Issues

To continue with the list of "Window Native Toolchain Issues" we can add

the following. These, however, are really just issues that you will have

if you use any pre-built toolchain (vs. building the NuttX toolchain from

the NuttX buildroot package):

There may be incompatibilities with header files, libraries, and compiler

built-in functions at detailed below. For the most part, these issues

are handled in the existing make logic. But if you are breaking new ground,

then you may encounter these:

4. Header Files. Most pre-built toolchains will build with a foreign C

library (usually newlib, but maybe uClibc or glibc if you are using a

Linux toolchain). This means that the header files from the foreign

C library will be built into the toolchain. So if you "include <stdio.h>",

you will get the stdio.h from the incompatible, foreign C library and

not the nuttx stdio.h (at nuttx/include/stdio.h) that you wanted.

This can cause really confusion in the builds and you must always be

sure the -nostdinc is included in the CFLAGS. That will assure that

you take the include files only from

5. Libraries. What was said above header files applies to libraries.

You do not want to include code from the libraries of any foreign

C libraries built into your toolchain. If this happens you will get

perplexing errors about undefined symbols. To avoid these errors,

you will need to add -nostdlib to your CFLAGS flags to assure that

you only take code from the NuttX libraries.

This, however, may causes other issues for libraries in the toolchain

that you do want (like libgcc.a or libm.a). These are special-cased

in most Makefiles, but you could still run into issues of missing

libraries.

6. Built-Ins. Some compilers target a particular operating system.

Many people would, for example, like to use the same toolchain to

develop Linux and NuttX software. Compilers built for other

operating systems may generate incompatible built-in logic and,

for this reason, -fno-builtin should also be included in your

C flags

And finally you may not be able to use NXFLAT.

7. NXFLAT. If you use a pre-built toolchain, you will lose all support

for NXFLAT. NXFLAT is a binary format described in

Documentation/NuttXNxFlat.html. It may be possible to build

standalone versions of the NXFLAT tools; there are a few examples

of this in the buildroot repository at https://bitbucket.org/nuttx/buildroot

However, it is possible that there could be interoperability issues

with your toolchain since they will be using different versions of

binutils and possibly different ABIs.

Building Original Linux Boards in Cygwin

Some default board configurations are set to build under Linux and others

to build under Windows with Cygwin. Various default toolchains may also

be used in each configuration. It is possible to change the default

setup. Here, for example, is what you must do in order to compile a

default Linux configuration in the Cygwin environment using the

CodeSourceery for Windows toolchain. After instantiating a "canned"

NuttX configuration, run the target 'menuconfig' and set the following

items:

Build Setup->Build Host Platform->Windows

Build Setup->Windows Build Environment->Cygwin

System Type->Toolchain Selection->CodeSourcery GNU Toolchain under Windows

In Windows 7 it may be required to open the Cygwin shell as Administrator

("Run As" option, right button) you find errors like "Permission denied".

Recovering from Bad Configurations

Many people make the mistake of configuring NuttX with the "canned"

configuration and then just typing 'make' with disastrous consequences;

the build may fail with mysterious, uninterpretable, and irrecoverable

build errors. If, for example, you do this with an unmodified Linux

configuration in a Windows/Cgwin environment, you will corrupt the

build environment. The environment will be corrupted because of POSIX vs

Windows path issues and with issues related to symbolic links. If you

make the mistake of doing this, the easiest way to recover is to just

start over: Do 'make distclean' to remove every trace of the corrupted

configuration, reconfigure from scratch, and make certain that the set

the configuration correctly for your platform before attempting to make

again.

Just fixing the configuration file after you have instantiated the bad

configuration with 'make' is not enough.

DOCUMENTATION

^^^^^^^^^^^^^

Additional information can be found in the Documentation/ directory and

also in README files that are scattered throughout the source tree. The

documentation is in HTML and can be access by loading the following file

into your Web browser:

Documentation/index.html

NuttX documentation is also available online at http://www.nuttx.org.

Below is a guide to the available README files in the NuttX source tree:

nuttx/

|

|- arch/

| |

| |- arm/

| | `- src

| | `- lpc214x/README.txt

| |- renesas/

| | |- include/

| | | `-README.txt

| | |- src/

| | | `-README.txt

| |- x86/

| | |- include/

| | | `-README.txt

| | `- src/

| | `-README.txt

| `- z80/

| | `- src/

| | |- z80/README.txt

| | `- z180/README.txt, z180\_mmu.txt

| `- README.txt

|- audio/

| `-README.txt

|- binfmt/

| `-libpcode/

| `-README.txt

|- configs/

| |- amber/

| | `- README.txt

| |- arduino-mega2560/

| | `- README.txt

| |- arduino-due/

| | `- README.txt

| |- avr32dev1/

| | `- README.txt

| |- b-l475e-iot01a/

| | `- README.txt

| |- bambino-200e/

| | `- README.txt

| |- c5471evm/

| | `- README.txt

| |- cc3200-launchpad/

| | `- README.txt

| |- clicker2-stm32

| | `- README.txt

| |- cloudctrl

| | `- README.txt

| |- demo0s12ne64/

| | `- README.txt

| |- dk-tm4c129x/

| | `- README.txt

| |- ea3131/

| | `- README.txt

| |- ea3152/

| | `- README.txt

| |- eagle100/

| | `- README.txt

| |- efm32-g8xx-stk/

| | `- README.txt

| |- efm32gg-stk3700/

| | `- README.txt

| |- ekk-lm3s9b96/

| | `- README.txt

| |- ez80f910200kitg/

| | |- ostest/README.txt

| | `- README.txt

| |- ez80f910200zco/

| | |- dhcpd/README.txt

| | |- httpd/README.txt

| | |- nettest/README.txt

| | |- nsh/README.txt

| | |- ostest/README.txt

| | |- poll/README.txt

| | `- README.txt

| |- fire-stm32v2/

| | `- README.txt

| |- freedom-k64f/

| | `- README.txt

| |- freedom-k66f/

| | `- README.txt

| |- freedom-kl25z/

| | `- README.txt

| |- freedom-kl26z/

| | `- README.txt

| |- hymini-stm32v/

| | `- README.txt

| |- kwikstik-k40/

| | `- README.txt

| |- launchxl-tms57004/

| | `- README.txt

| |- lincoln60/

| | `- README.txt

| |- lm3s6432-s2e/

| | `- README.txt

| |- lm3s6965-ek/

| | `- README.txt

| |- lm3s8962-ek/

| | `- README.txt

| |- lpc4330-xplorer/

| | `- README.txt

| |- lpc4337-ws/

| | `- README.txt

| |- lpc4357-evb/

| | `- README.txt

| |- lpc4370-link2/

| | `- README.txt

| |- lpcxpresso-lpc1115/

| | `- README.txt

| |- lpcxpresso-lpc1768/

| | `- README.txt

| |- maple/

| | `- README.txt

| |- mbed/

| | `- README.txt

| |- mcu123-lpc214x/

| | `- README.txt

| |- micropendous3/

| | `- README.txt

| |- mikroe-stm32f/

| | `- README.txt

| |- mirtoo/

| | `- README.txt

| |- misoc/

| | `- README.txt

| |- moteino-mega/

| | `- README.txt

| |- mx1ads/

| | `- README.txt

| |- ne63badge/

| | `- README.txt

| |- ntosd-dm320/

| | |- doc/README.txt

| | `- README.txt

| |- nucleo-144/

| | `- README.txt

| |- nucleo-f072rb/

| | `- README.txt

| |- nucleo-f091rc/

| | `- README.txt

| |- nucleo-f303re/

| | `- README.txt

| |- nucleo-f334r8/

| | `- README.txt

| |- nucleo-f4x1re/

| | `- README.txt

| |- nucleo-l432kc/

| | `- README.txt

| |- nucleo-l452re/

| | `- README.txt

| |- nucleo-l476rg/

| | `- README.txt

| |- nucleo-l496zg/

| | `- README.txt

| |- nutiny-nuc120/

| | `- README.txt

| |- olimex-efm32g880f129-stk/

| | `- README.txt

| |- olimex-lpc1766stk/

| | `- README.txt

| |- olimex-lpc2378/

| | `- README.txt

| |- olimex-lpc-h3131/

| | `- README.txt

| |- olimex-stm32-h405/

| | `- README.txt

| |- olimex-stm32-h407/

| | `- README.txt

| |- olimex-stm32-p107/

| | `- README.txt

| |- olimex-stm32-p207/

| | `- README.txt

| |- olimex-stm32-p407/

| | `- README.txt

| |- olimex-strp711/

| | `- README.txt

| |- open1788/

| | `- README.txt

| |- p112/

| | `- README.txt

| |- pcblogic-pic32mx/

| | `- README.txt

| |- pcduino-a10/

| | `- README.txt

| |- pic32mx-starterkit/

| | `- README.txt

| |- pic32mx7mmb/

| | `- README.txt

| |- pic32mz-starterkit/

| | `- README.txt

| |- photon/

| | `- README.txt

| |- qemu-i486/

| | `- README.txt

| |- sabre-6quad/

| | `- README.txt

| |- sama5d2-xult/

| | `- README.txt

| |- sama5d3x-ek/

| | `- README.txt

| |- sama5d3-xplained/

| | `- README.txt

| |- sama5d4-ek/

| | `- README.txt

| |- samd20-xplained/

| | `- README.txt

| |- samd21-xplained/

| | `- README.txt

| |- saml21-xplained/

| | `- README.txt

| |- sam3u-ek/

| | `- README.txt

| |- sam4cmp-db

| | `- README.txt

| |- sam4e-ek/

| | `- README.txt

| |- sam4l-xplained/

| | `- README.txt

| |- sam4s-xplained/

| | `- README.txt

| |- sam4s-xplained-pro/

| | `- README.txt

| |- same70-xplained/

| | `- README.txt

| |- samv71-xult/

| | `- README.txt

| |- sim/

| | |- include/README.txt

| | `- README.txt

| |- shenzhou/

| | `- README.txt

| |- skp16c26/

| | `- README.txt

| |- spark/

| | `- README.txt

| |- stm3210e-eval/

| | |- RIDE/README.txt

| | `- README.txt

| |- stm3220g-eval/

| | |-ide/nsh/iar/README.txt

| | |-ide/nsh/uvision/README.txt

| | `- README.txt

| |- stm3240g-eval/

| | `- README.txt

| |- stm32\_tiny/

| | `- README.txt

| |- stm32f103-minumum/

| | `- README.txt

| |- stm32f3discovery/

| | `- README.txt

| |- stm32f4discovery/

| | `- README.txt

| |- stm32f411e-disco/

| | `- README.txt

| |- stm32f429i-disco/

| | |- ide/ltcd/uvision/README.txt

| | |- ltdc/README.txt

| | `- README.txt

| |- stm32f746g-disco/

| | `- README.txt

| |- stm32l476-mdk/

| | `- README.txt

| |- stm32l476vg-disco/

| | `- README.txt

| |- stm32ldiscovery/

| | `- README.txt

| |- stm32vldiscovery/

| | `- README.txt

| |- sure-pic32mx/

| | `- README.txt

| |- teensy-2.0/

| | `- README.txt

| |- teensy-3.x/

| | `- README.txt

| |- teensy-lc/

| | `- README.txt

| |- tm4c123g-launchpad/

| | `- README.txt

| |- tm4c1294-launchpad/

| | `- README.txt

| |- twr-k60n512/

| | `- README.txt

| |- twr-k64f120m/

| | `- README.txt

| |- u-blox-co27/

| | `- README.txt

| |- ubw32/

| | `- README.txt

| |- us7032evb1/

| | `- README.txt

| |- viewtool-stm32f107/

| | `- README.txt

| |- xmc5400-relax/

| | `- README.txt

| |- xtrs/

| | `- README.txt

| |- z16f2800100zcog/

| | |- ostest/README.txt

| | |- pashello/README.txt

| | `- README.txt

| |- z80sim/

| | `- README.txt

| |- z8encore000zco/

| | |- ostest/README.txt

| | `- README.txt

| |- z8f64200100kit/

| | |- ostest/README.txt

| | `- README.txt

| |- zkit-arm-1769/

| | `- README.txt

| |- zp214xpa/

| | `- README.txt

| `- README.txt

|- drivers/

| |- eeprom/

| | `- README.txt

| |- lcd/

| | | README.txt

| | `- pcf8574\_lcd\_backpack\_readme.txt

| |- mtd/

| | `- README.txt

| |- sensors/

| | `- README.txt

| |- syslog/

| | `- README.txt

| `- README.txt

|- fs/

| |- binfs/

| | `- README.txt

| |- mmap/

| | `- README.txt

| |- nxffs/

| | `- README.txt

| |- smartfs/

| | `- README.txt

| |- procfs/

| | `- README.txt

| `- unionfs/

| `- README.txt

|- graphics/

| `- README.txt

|- lib/

| `- README.txt

|- libc/

| |- zoneinfo

| | `- README.txt

| `- README.txt

|- libnx/

| `- README.txt

|- libxx/

| `- README.txt

|- mm/

| |- shm/

| | `- README.txt

| `- README.txt

|- net/

| |- sixlowpan

| | `- README.txt

| `- README.txt

|- syscall/

| `- README.txt

`- tools/

`- README.txt

Below is a guide to the available README files in the semi-optional apps/

source tree:

apps/

|- examples/

| |- bastest/README.txt

| |- json/README.txt

| |- pashello/README.txt

| `- README.txt

|- gpsutils/

| `- minmea/README.txt

|- graphics/

| |- tiff/README.txt

| `- traveler/tools/tcledit/README.txt

|- interpreters/

| |- bas

| | `- README.txt

| |- ficl

| | `- README.txt

| `- README.txt

|- modbus/

| `- README.txt

|- netutils/

| |- discover

| | `- README.txt

| |- ftpc

| | `- README.txt

| |- json

| | `- README.txt

| |- telnetd

| | `- README.txt

| `- README.txt

|- nshlib/

| `- README.txt

|- NxWidgets/

| `- README.txt

|- system/

| |- cdcacm

| | `- README.txt

| |- i2c

| | `- README.txt

| |- inifile

| | `- README.txt

| |- install

| | `- README.txt

| |- nxplayer

| | `- README.txt

| |- symtab/

| | `- README.txt

| |- usbmsc

| | `- README.txt

| `- zmodem

| `- README.txt

`- README.txt

Additional README.txt files in the other, related repositories:

NxWidgets/

|- Doxygen

| `- README.txt

|- tools

| `- README.txt

|- UnitTests

| `- README.txt

`- README.txt

buildroot/

`- README.txt

tools/

`- README.txt

uClibc++/

`- README.txt

pascal/

`- README.txt