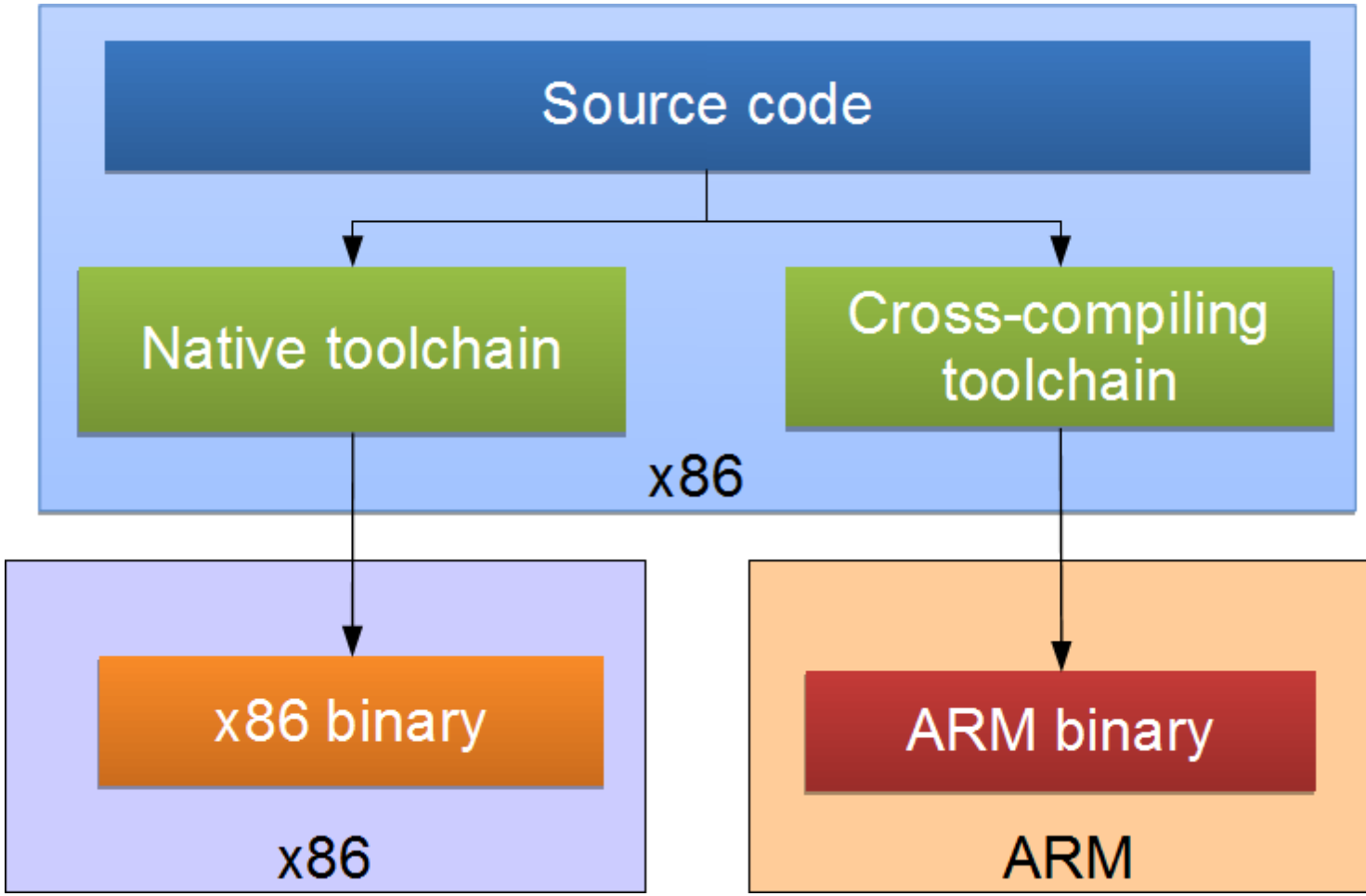
Cross-compiling toolchains

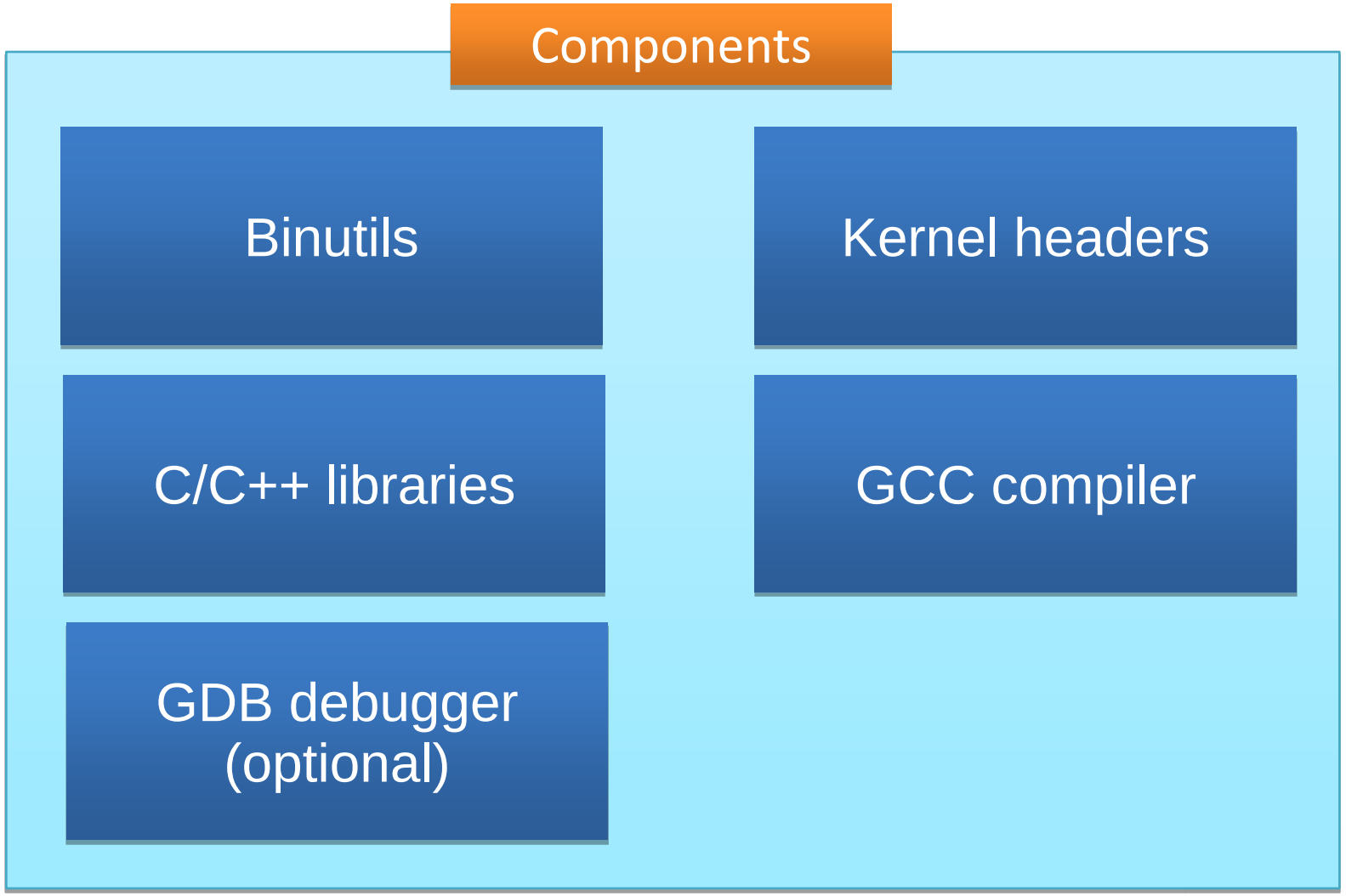
Cross-compiling toolchains

- ✓ The usual development tools available on a GNU/Linux workstation is a **native toolchain**
- ✓ This toolchain runs on your workstation and generates code for your workstation, usually x86
- ✓ For embedded system development, it is usually impossible or not interesting to use a native toolchain
- ✓ The target is too restricted in terms of storage and/or memory
- ✓ The target is very slow compared to your workstation
- ✓ You may not want to install all development tools on your target.
- ✓ Therefore, **cross-compiling toolchains** are generally used. They run on your workstation but generate code for your target.

Cross-compiling toolchains



Cross-compiling toolchains

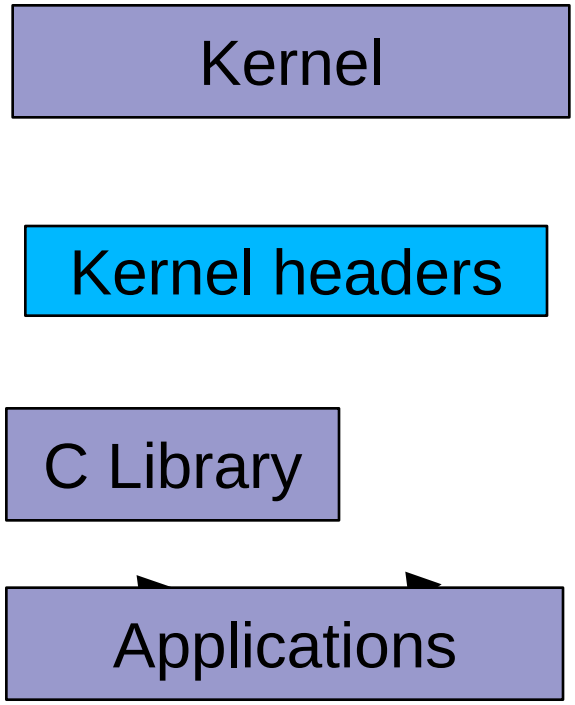


Binutils

- ✓ **Binutils** is a set of tools to generate and manipulate binaries for a given CPU architecture
- ✓ as, the assembler, that generates binary code from assembler source code
- ✓ ld, the linker
- ✓ ar, ranlib, to generate .a archives, used for libraries
- ✓ objdump, readelf, size, nm, strings, to inspect binaries. Very useful analysis tools !
- ✓ strip, to strip useless parts of binaries in order to reduce their size
- ✓ http://www.gnu.org/software/binutils/
- ✓ GPL license

Kernel headers

- ✓ The C library and compiled programs needs to interact with the kernel
- ✓ Available system calls and their numbers
- ✓ Constant definitions
- ✓ Data structures, etc.
- ✓ Therefore, compiling the C library requires kernel headers, and many applications also require them.



Available in <linux/...> and <asm/...> and a few other directories corresponding to the ones visible in include/ in the kernel sources

Kernel headers

- System call numbers, in <asm/unistd.h>

```
#define __NR_exit                1
#define __NR_fork                2
#define __NR_read                3
```

- Constant definitions, here in <asm-generic/fcntl.h>, included from <asm/fcntl.h>, included from <linux/fcntl.h>

```
#define O_RDWR          00000002
```

- Data structures, here in <asm/stat.h>

```
struct stat {
        unsigned long  st_dev;
        unsigned long  st_ino;
                [...]
};
```

Kernel headers

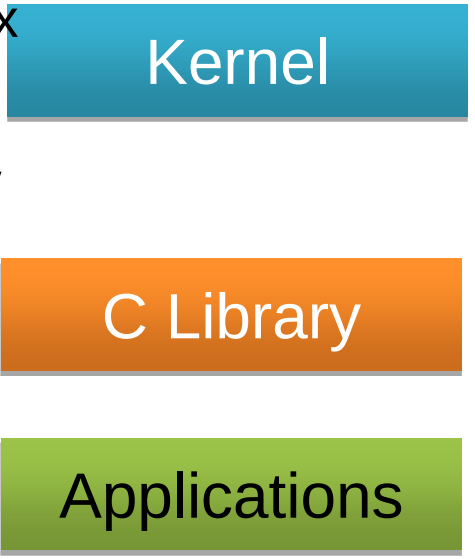
- ✓ The kernel-to-userspace ABI is backward compatible
- ✓ Binaries generated with a toolchain using kernel headers older than the running kernel will work without problem, but won't be able to use the new system calls, data structures, etc.
- ✓ Binaries generated with a toolchain using kernel headers newer than the running kernel might work on if they don't use the recent features, otherwise they will break
- ✓ Using the latest kernel headers is not necessary, unless access to the new kernel features is needed
- ✓ The kernel headers are extracted from the kernel sources using the headers_install kernel Makefile target.

GCC compiler

- ✓ GNU C Compiler, the famous free software compiler
- ✓ Can compile C, C++, Ada, Fortran, Java, Objective-C, Objective-C++, and generate code for a large number of CPU architectures, including ARM, AVR, Blackfin, CRIS, FRV, M32, MIPS, MN10300, PowerPC, SH, v850, i386, x86_64, IA64, Xtensa, etc.
- ✓ http://gcc.gnu.org/
- ✓ Available under the GPL license, libraries under the LGPL.

C library

- ✓ The C library is an essential component of a Linux system
- ✓ Interface between the applications and the kernel
- ✓ Provides the well-known standard C API to ease application development
- ✓ Several C libraries are available:
  glibc, uClibc, eglibc, dietlibc, newlib, etc.
- ✓ The choice of the C library must be made at the time of the cross-compiling toolchain generation, as the GCC compiler is compiled against a specific C library.



glibc

http://www.gnu.org/software/libc/

- ✓ License: LGPL
- ✓ C library from the GNU project
- ✓ Designed for performance, standards compliance and portability
- ✓ Found on all GNU / Linux host systems
- ✓ Of course, actively maintained
- ✓ Quite big for small embedded systems: approx 2.5 MB on arm (version 2.9 - libc: 1.5 MB, libm: 750 KB)

uClibc

http://www.uclibc.org/ from CodePoet Consulting

- ✓ License: LGPL
- ✓ Lightweight C library for small embedded systems
- ✓ High configurability: many features can be enabled or disabled through a menuconfig interface
- ✓ Works only with Linux/uClinux, works on most embedded architectures
- ✓ No stable ABI, different ABI depending on the library configuration
- ✓ Focus on size rather than performance
- ✓ Small compile time

uClibc

- ✓ Most of the applications compile with uClibc. This applies to all applications used in embedded systems.
- ✓ Size (arm): 4 times smaller than glibc!
  uClibc 0.9.30.1: approx. 600 KB (libuClibc: 460 KB, libm: 96KB)
  glibc 2.9: approx 2.5 MB
- ✓ Used on a large number of production embedded products, including consumer electronic devices
- ✓ Actively maintained, large developer and user base
- ✓ Now supported by MontaVista, TimeSys and Wind River.

uClibc

After compilation and installation, the following components are available

- ✓ Standard headers, stdio.h, stdlib.h, unistd.h and others, and Linux kernel headers, integrated with the C library headers.
- ✓ The libraries themselves, with mainly
- ✓ libuClibc, the C library itself
- ✓ ld-uClibc, the dynamic loader, responsible for loading the shared libraries at the beginning of a program's execution
- ✓ librt, the library implementing the real-time related functions
- ✓ libstdc++, the C++ standard library
- ✓ libpthread, the threads library
- ✓ libm, the mathematic library

eglibc

« Embedded glibc », under the LGPL

- ✓ Variant of the GNU C Library (GLIBC) designed to work well on embedded systems
- ✓ Strives to be source and binary compatible with GLIBC
- ✓ eglibc's goals include reduced footprint, configurable components, better support for cross-compilation and cross-testing.
- ✓ Can be built without support for NIS, locales, IPv6, and many other features.
- ✓ Supported by a consortium, with Freescale, MIPS, MontaVista and Wind River as members.
- ✓ http://www.eglibc.org

Other smaller C libraries

- ✓ Several other smaller C libraries have been developed, but none of them have the goal of allowing the compilation of large existing applications
- ✓ They need specially written programs and applications
- ✓ Choices :
- ✓ Dietlibc, http://www.fefe.de/dietlibc/. Approximately 70 KB.
- ✓ Newlib, http://sources.redhat.com/newlib/
- ✓ Klibc, http://www.kernel.org/pub/linux/libs/klibc/, designed for use in an initramfs or initrd at boot time.

Building a toolchain

- ✓ Three machines must be distinguished when discussing toolchain creation
- ✓ The build machine, where the toolchain is built.
- ✓ The host machine, where the toolchain will be executed.
- ✓ The target machine, where the binaries created by the toolchain will be executed.
- ✓ Four build types are possible

Native build

used to build the normal gcc of a workstation

Cross build

used to build a toolchain that runs on your workstation but generates binaries for the target

The most common solution in embedded

Cross-native build

used to build a toolchain that runs on your target and generates binaries for the target

Canadian build

used to build on architecture A a toolchain that runs on architecture B and generates binaries for architecture C

Toolchain building utilities

Another solution is to use utilities that automate the process of building the toolchain

- ✓ Same advantage as the pre-compiled toolchains: you don't need to mess up with all the details of the build process
- ✓ But also offers more flexibility in terms of toolchain configuration, component version selection, etc.
- ✓ They also usually contain several patches that fix known issues with the different components on some architectures
- ✓ Identical principle: shell scripts or Makefile that automatically fetch, extract, configure, compile and install the different components

Toolchain building utilities

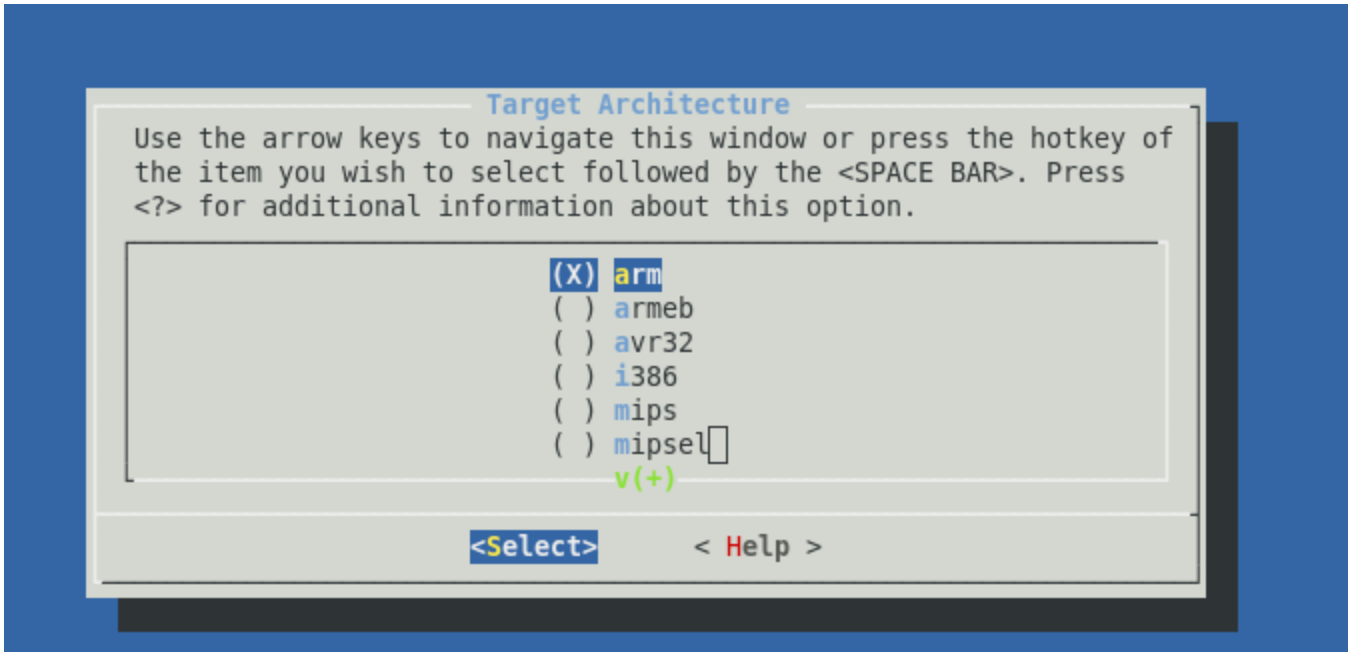
Many root filesystem building systems also allow the construction of cross-compiling toolchain

- ✓ Buildroot
- ✓ Makefile-based, uClibc only, maintained by the community
- ✓ http://buildroot.uclibc.org
- ✓ PTXdist
- ✓ Makefile-based, uClibc or glibc, maintained mainly by Pengutronix
- ✓ http://www.pengutronix.de/software/ptxdist/index_en.html
- ✓ OpenEmbedded
- ✓ The feature-full, but complex building system
- ✓ http://www.openembedded.org/

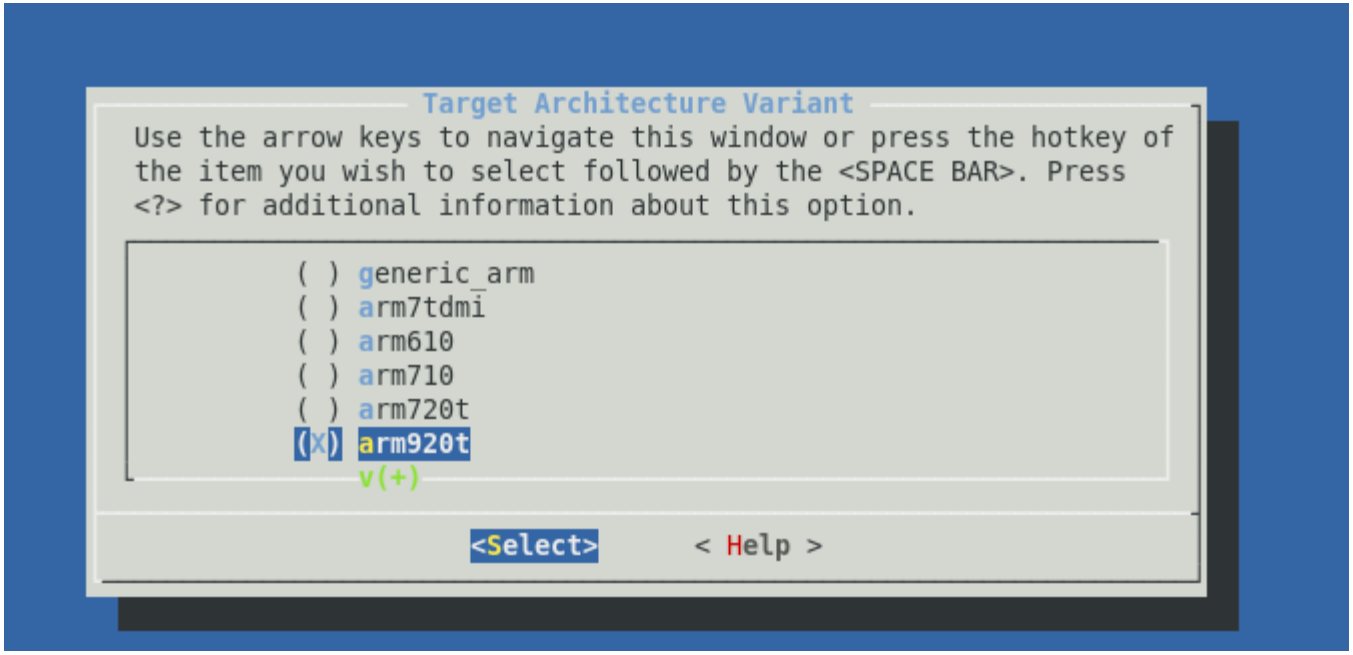
Buildroot

Steps to build cross toolchain for arm AT920T using buildroot

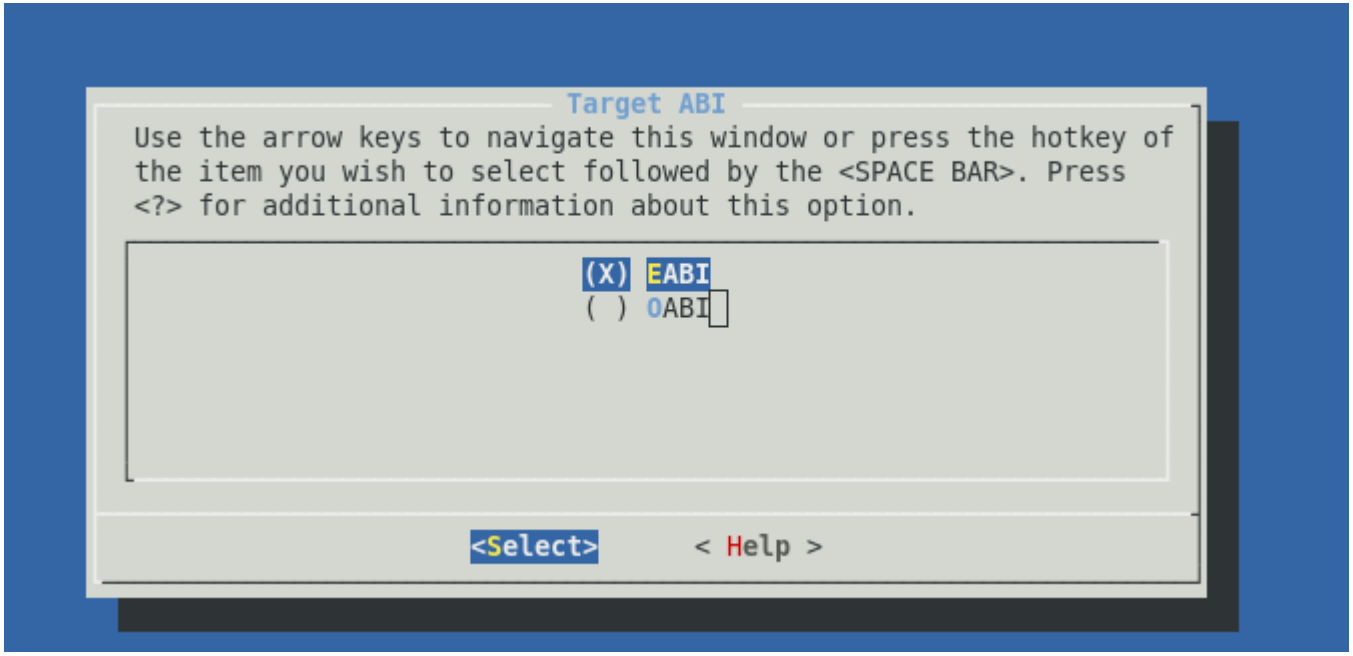
1. untar buildroot source
2. tar xvf buildroot-2010.11.tar.gz
3. make menuconfig



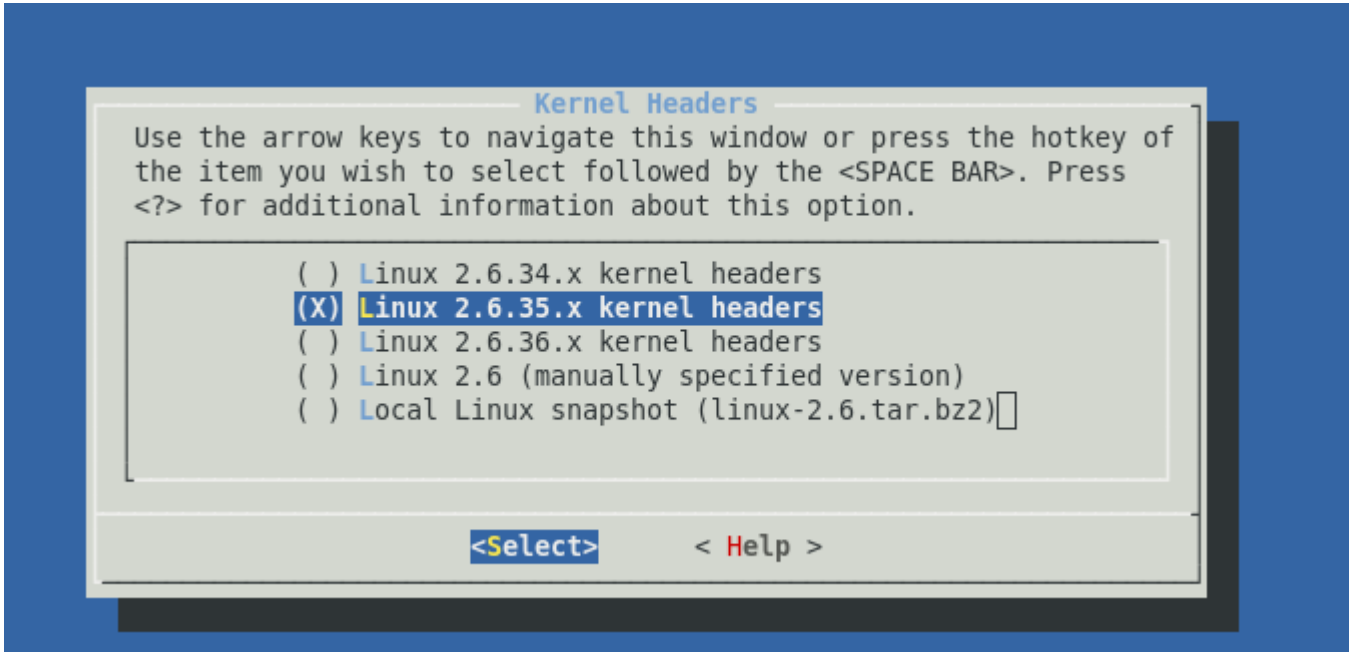
Buildroot


Target Architecture Variant
Use the arrow keys to navigate this window or press the hotkey of the item you wish to select followed by the <SPACE BAR>. Press <?> for additional information about this option.
( ) generic_arm
( ) arm7tdmi
( ) arm610
( ) arm710
( ) arm720t
(X) arm920t
v(+)
<Select>
< Help >

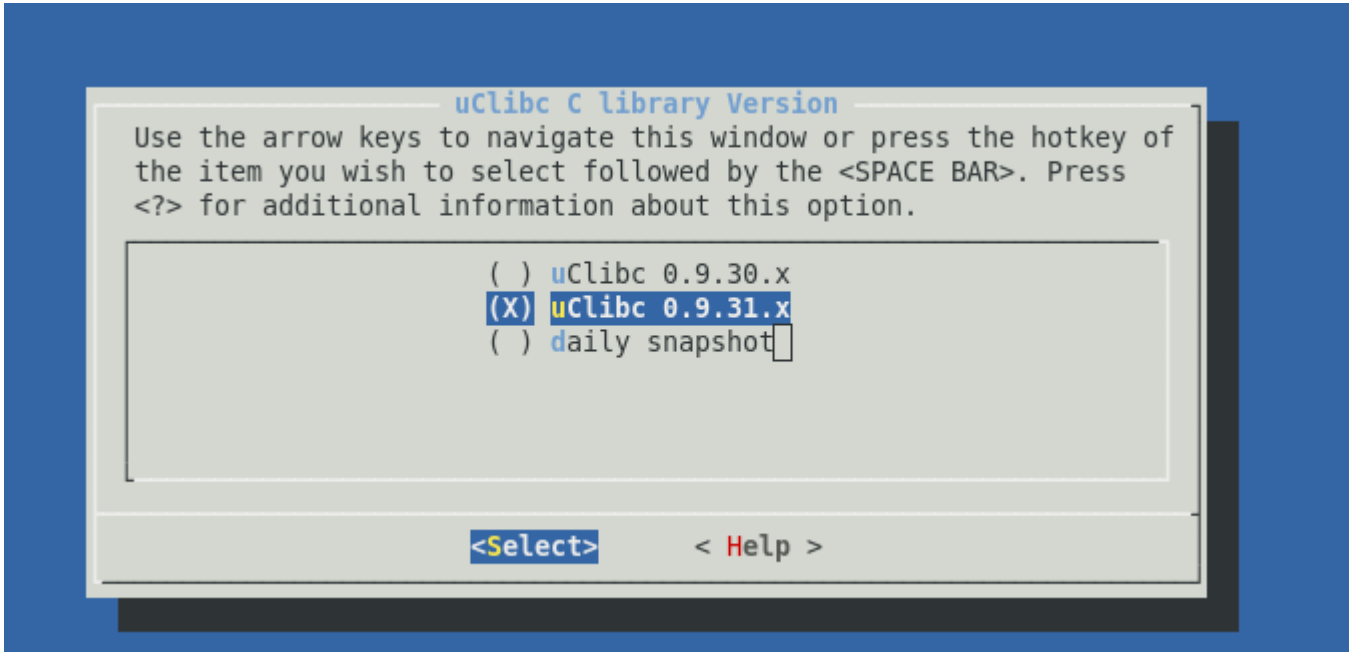
Buildroot


Target ABI
Use the arrow keys to navigate this window or press the hotkey of the item you wish to select followed by the <SPACE BAR>. Press <?> for additional information about this option.
(X) EABI
( ) OABI
<Select>
< Help >

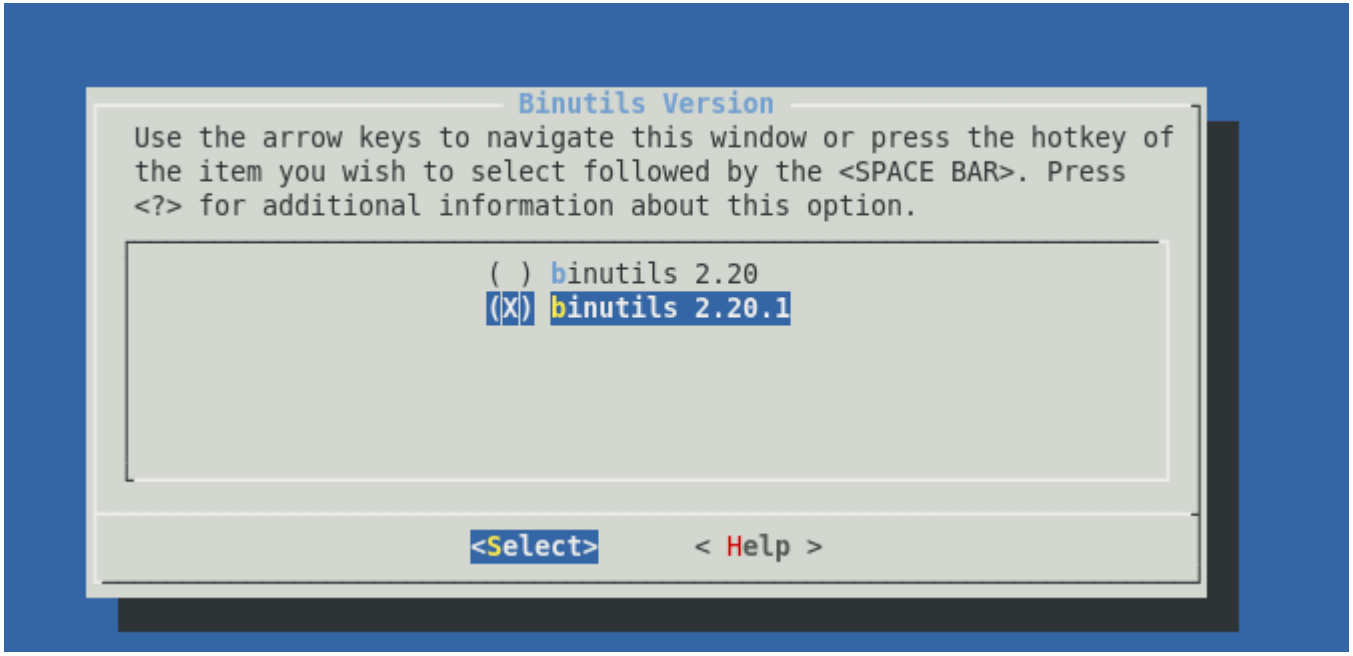
Buildroot



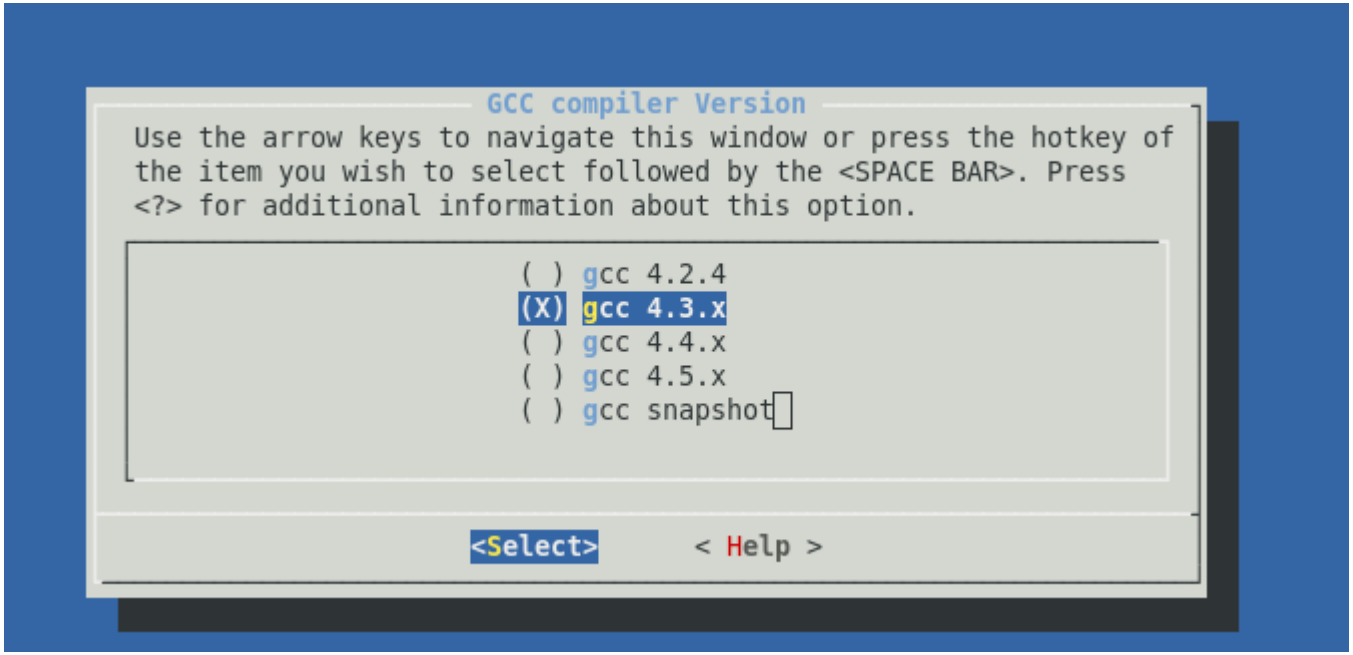
Buildroot



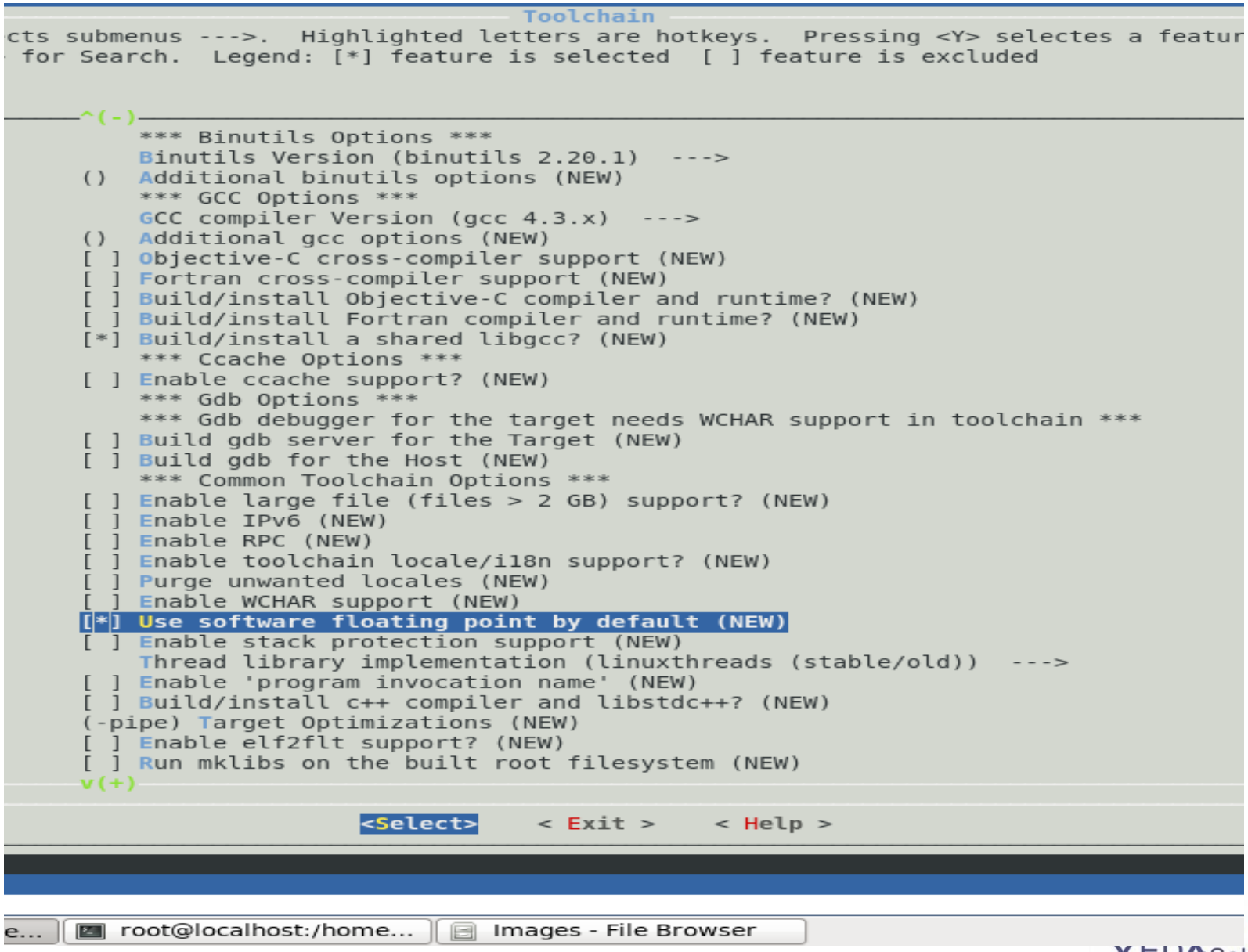
Buildroot


Binutils Version
Use the arrow keys to navigate this window or press the hotkey of the item you wish to select followed by the <SPACE BAR>. Press <?> for additional information about this option.
( ) binutils 2.20
(X) binutils 2.20.1
<Select>
< Help >

Buildroot


GCC compiler Version
Use the arrow keys to navigate this window or press the hotkey of the item you wish to select followed by the <SPACE BAR>. Press <?> for additional information about this option.
( ) gcc 4.2.4
(X) gcc 4.3.x
( ) gcc 4.4.x
( ) gcc 4.5.x
( ) gcc snapshot
<Select>
< Help >

Buildroot


Toolchain
cts submenus --->. Highlighted letters are hotkeys. Pressing <Y> selectes a featur
for Search. Legend: [*] feature is selected [ ] feature is excluded
*** Binutils Options ***
Binutils Version (binutils 2.20.1) --->
() Additional binutils options (NEW)
*** GCC Options ***
GCC compiler Version (gcc 4.3.x) --->
() Additional gcc options (NEW)
[ ] Objective-C cross-compiler support (NEW)
[ ] Fortran cross-compiler support (NEW)
[ ] Build/install Objective-C compiler and runtime? (NEW)
[ ] Build/install Fortran compiler and runtime? (NEW)
[*] Build/install a shared libgcc? (NEW)
*** Ccache Options ***
[ ] Enable ccache support? (NEW)
*** Gdb Options ***
*** Gdb debugger for the target needs WCHAR support in toolchain ***
[ ] Build gdb server for the Target (NEW)
[ ] Build gdb for the Host (NEW)
*** Common Toolchain Options ***
[ ] Enable large file (files > 2 GB) support? (NEW)
[ ] Enable IPv6 (NEW)
[ ] Enable RPC (NEW)
[ ] Enable toolchain locale/i18n support? (NEW)
[ ] Purge unwanted locales (NEW)
[ ] Enable WCHAR support (NEW)
[*] Use software floating point by default (NEW)
[ ] Enable stack protection support (NEW)
Thread library implementation (linuxthreads (stable/old)) --->
[ ] Enable 'program invocation name' (NEW)
[ ] Build/install c++ compiler and libstdc++? (NEW)
(-pipe) Target Optimizations (NEW)
[ ] Enable elf2flt support? (NEW)
[ ] Run mklibs on the built root filesystem (NEW)
<Select> < Exit > < Help >
root@localhost:/home...
Images - File Browser

make

On success, cross-tool-chain gets installed under
$<buildroot_src>/output/staging/usr/bin/ directory with a “arm-linux-” prefix

Thank you