AN192

Application Note

THE GNU BASED ARM-LINUX TOOLCHAIN FOR USE WITH ROYAL LINUX

TABLE OF CONTENTS

1. INTRODUCTION

This document focuses on the GNU based arm-linux toolchain for development on a Linux host machine. Included are useful links to precompiled binaries, and information for building the toolchain from scratch. It is assumed that the reader use Linux kernel source code from ISD Corporation's Royal Linux for the EDB7211 or EDB7212. If another variety of Linux is used, substitute the correct kernel source code path for the Royal Linux source code path.

1.1 Style Conventions

The following style conventions are used in this document:

- `Courier New font` – Is used to denote C code and shell commands, file and directory names. The code text is indented to differentiate to from rest of the text in the document
- <TARGET_DIR> – The intended installation directory for the cross compile tools. For this exercise the target directory was /usr/local/arm.
- <ROYALLINUX_DIR> – The path to the kernel source code from ISD Corporation's Royal Linux package for the EP7211.
- <BUILD_MACHINE> – The machine where the cross development tools are being built, needed for the glibc build. For this exercise the machine used to build the toolchain was a Pentium® 2 PC with a Linux® OS, and therefore for this exercise <BUILD_MACHINE> is "i686-linux." For more options for <BUILD_MACHINE>, look at the file INSTALL in the `glibc` directory created by extracting the glibc archive.

2. VARIATIONS OF THE GNU ARM TOOLCHAIN

The variations of the GNU ARM toolchain are:

- **`arm-linux`** – Generates Linux dependent ELF binaries for an ARM target
- **`arm-linuxaout`** – Generates Linux dependent a.out binaries for an ARM target. This toolchain is not used very often.
- **`arm-aout`** – Generates standalone a.out binaries for an ARM target
- **`arm-coff`** – Generates standalone COFF binaries for an ARM target
- **`arm-elf`** – Generates standalone ELF binaries for an ARM target.

The subject of this document is the arm-linux toolchain for a Linux host machine.

3. ACQUIRING ARM-LINUX PRECOMPILED BINARIES

While binary distribution of the arm-linux precompiled binaries is somewhat sparse, there are some options available. Here is one site which offers precompiled arm-linux binaries,

http://www.empeg.mars.org/devel/software/toolchain.php3

This site has a link to the binaries as well as instructions on how to set up the tools.

Another option is the Enhanced Royal Linux Package for the EDB7211 or EDB7212 from ISD Corporation. This package includes a complete set of arm-linux precompiled binaries. To obtain this package, contact: Art Lee from ISD Corporation at the mailing address, telephone number, or email address listed on the following page.

Art Lee
Integrated Software & Devices Corporation
3317 Vintage Dr.
Round Rock, TX. 78664
(512) 238-8230 - Voice
(512) 238-8816 - Fax
Art_Lee@ISDCorp.com

4. BUILDING THE ARM-LINUX TOOLCHAIN FROM SCRATCH

There are three parts to a complete arm-linux toolchain, the binutils, gcc, and libc. To rebuild the Linux kernel itself, only the binutils and gcc packages are needed. To build an Linux applications, the glibc package is needed in addition to the other two packages. The following versions were used for this example:

- binutils – 2.9.5.0.34
- gcc – 2.95.3
- glibc – 2.1.3

4.1 Install and Build binutils

Follow these steps to install and build binutils:

1) Download the binutils package. Try the following site:

   ftp://ftp.varesearch.com/pub/support/hjl/binutils/

2) Extract the package with the following commands from the shell prompt:

```
gunzip binutils-2.9.5.0.34.tar.gz
tar xvf binutils-2.9.5.0.34.tar
```

3) Change directories into the directory where the binutils code was extracted:

```
cd binutils-2.9.5.0.34
```

4) Configure the binutils source code for the cross build:

```
./configure --prefix=<TARGET_DIR> --target=arm-linux
```

5) Build and install the binutils:

```
make
make install
```

4.2 Install and Build gcc

To install and build gcc, follow these steps:

1) To build a cross gcc for a target different from the machine where the tools are built, the Linux kernel has to be configured for the target. Once configured for the target, the Linux header files, needed by gcc and glibc, should be copied to the toolchain installation directory. Use the following process to configure the kernel source code,

2) Change into the kernel source code directory

```
cd <ROYALLINUX_DIR>
```

3) Copy the file `myconfig` to `.config`.

```
cp myconfig .config
```

4) Configure the kernel source code using the settings in the file .config

```
make oldconfig
make dep
```

5) Copy the Linux kernel header files the target directory, <TARGET_DIR>. For the Royal Linux package, use the following commands:

```
cp –dR <ROYALLINUX_DIR>/include/linux <TARGET_DIR>/arm-linux/include/linux
cp –dR <ROYALLINUX_DIR>/include/asm-arm <TARGET_DIR>/arm-linux/include/asm
```

6) If kernel header files from the Royal Linux package are used, which is recommended, one of the header files must to be modified. Make the following modifications to the file `<ROYALLINUX_DIR>/include/linux/sysctl.h`

   a) Add `CTL_BUS=8` to the following enumeration in `sysctl.h`:

```
enum
{
CTL_KERN=1,     /* General Kernel info and control */
CTL_VM=2,       /* VM management */
…
CTL_DEV=7,      /* Devices        */
CTL_BUS=8       /* Added for Buses */
};
```

   b) Add the following new enumerations to `sysctl.h`:

```
/* CTL_BUS names */
enum
{
BUS_ISA=1         /* ISA */
};
enum
{
BUS_ISA_MEM_BASE=1,
BUS_ISA_PORT_BASE=2,
BUS_ISA_PORT_SHIFT=3
};
```

7) Download the gcc package. Try the following Web site:

ftp://egcs.cygnus.com/pub/egcs/releases/

8) Extract the package with the following commands from the shell prompt:

```
gunzip gcc-2.95.3.tar.gz
tar xvf gcc-2.95.3.tar
```

9) Change directories into the directory where the gcc code was extracted:

```
cd gcc-2.95.3
```

10) Since this is the first time a cross gcc will be built and installed, some header files, which are installed with the cross gcc, are missing from the <TARGET_DIR>. To get around this problem, add the flags –`Dinhibit_libc` and –`D__gthr_posix_h` to `TARGET_LIBGCC2_CFLAGS`, in the file `gcc-2.95.3/gcc/config/arm/t-linux`. The line should look like this:

```
TARGET_LIBGCC2_CFLAGS = -fomit-frame-pointer –fPIC -Dinhibit_libc
–D__gthr_posix_h.
```

11) Configure the gcc source code for the cross build. To build gcc with support for languages other than c requires the base gcc cross compiler with c support. Therefore, if support for other languages (Fortran, C++, etc) is desired, gcc must be built and installed twice, the first with only c support enabled, the second time any other language support enabled. For this document gcc is built with only c support:

```
./configure --prefix=<TARGET_DIR> --target=arm-linux ––with-
headers=<TARGET_DIR>/arm-linux/include -–enable-languages=”c” –disable-threads
```

12) Build and install gcc:

```
make
make install
```

4.3 Install and Build Glibc

To install and build Glibc, follow these steps:

1) The glibc package comes in three pieces, the glibc archive, the linux threads archive, and the crypt archive. Download the three archives, glibc, linuxthreads, crypt. Try the following site:

ftp://sourceware.cygnus.com/pub/glibc/

If you can’t find the crypt archive at the above site, try the following site:

ftp://ftp.funet.fi/pub/gnu/funet/

2) Extract the glibc package with the following commands from the shell prompt:

```
gunzip glibc-2.1.3.tar.gz
tar xvf glibc-2.1.3.tar
```

3) Copy the glibc add-on archives into the newly created glibc directory:

```
cp glibc-linuxthreads-2.1.3.tar.gz glibc-2.1.3
cp glibc-crypt-2.1.tar.gz glibc-2.1.3
```

4) Change directories into the glibc directory:

```
cd glibc-2.1.3
```

5) Extract the linuxthreads package with the following commands from the shell prompt:

```
gunzip glibc-linuxthreads-2.1.3.tar.gz
tar xvf glibc-linuxthreads-2.1.3.tar
```

6) Extract the crypt package with the following commands from the shell prompt:

```
gunzip glibc-crypt-2.1.tar.gz
tar xvf glibc-crypt-2.1.tar
```

7) Before configuring and building glibc, several environment variables must be set to point to the newly created arm-linux cross development tools. From a bash shell use the following commands:

```
export CC=arm-linux-gcc
export BUILD_CC=gcc
export AR=arm-linux-ar
export RANLIB=arm-linux-ranlib
export PATH=$PATH:<TARGET_DIR>/bin
```

8) Configure the binutils source code for the cross build.

```
./configure –-host=arm-linux –-build=<BUILD_MACHINE> –-enable-add-
ons=crypt,linuxthreads –-with-headers=<TARGET_DIR>/arm-linux/include --
prefix=<TARGET_DIR>/arm-linux
```

9) Build and install glibc

```
make
make install
```

5. POSSIBLE PROBLEM WITH THE BUILD

The following problem was encountered during this exercise. Included is a problem description and resolution

Problem: During configuration of glibc, the configuration tool tests the cross gcc by building sample applications. The test build failed with this error: “ld: unrecognized emulation mode: elf32arm”. The reason for the error is an incompatibility between the binutils version and the gcc version. Somewhere around version 2.9.5.034 of the binutils the emulation names changed. Gcc wasn’t updated with the emulation name changes until version 2.95.3. This problem can be fixed one of two ways:

1) Update the gcc package to version 2.95.3 or later, which is how the problem was resolved for this exercise, or
2) modify the gcc spec file to reflect the newer emulation names (armelf_linux, armelf_linux26, armelf).

6. OTHER USEFUL LINKS

http://inkvine.fluff.org/~chris/arm-tools.html
– *Building the GNU Toolchain for ARM Targets* by Chris Rutter. This document is one of the most thorough guides for building an arm-linux toolchain, and is a major source for this application note. It goes through the process of downloading the sources, applying patches, configuring the sources, building, and finally installing each of the three packages for the complete toolchain.

http://www.objsw.com/CrossGCC/ – This Web site is the cross gcc Web page. This site hosts the crossgcc mailing list, which is a great forum to discuss issues related to building cross development tools. This web site and the crossgcc mailing list are not specific to the ARM architecture, or to Linux.

http://www.tazenda.demon.co.uk/phil/armlinux/
– Phil Blundel's ARM-Linux page. There is a great deal of information here, not only about arm-linux tools, but also about current arm-linux projects.

• Notes •

Maverick™

from
CIRRUS LOGIC