Scratchbox2 environment for Kindle Touch development

Release

Leonid Borisenko

August 21, 2015

Contents

This is a "laboratory notes" on how to setup cross-compiling environment for Kindle Touch with using of scratchbox2.

Contents:

1 Install cross-compilers

Cross-compiling is a process for producing binary code for *target* system, while compiling source code on *host* system. So, for example, one could compile source code on powerful computer with Intel x86 processor, while producing code for relatively "weak" ARM-based computer (like Kindle Touch).

1.1 GCC

GCC includes compilers for C and C++ (and some others programming languages).

Not only compiler is needed for producing of finished binary code, but also some other utilites (like assembler, linker). GNU compiler from GCC combined with these utilities is called *GNU toolchain*.

There are two ways to install cross-compiling GNU toolchain: either compile it from sources (maybe, by automated process with the help of specialized tools, like crosstool-NG), or install a pre-built toolchain.

Popular pre-built GNU toolchains, suitable for Kindle Touch development, are:

- Linaro
- Sourcery CodeBench Lite Edition (built for *GNU/Linux* target OS)

Note: Stock Kindle Touch binary files were built with following GNU toolchains (number shows how many files were built with this toolchain):

```
799    (Linaro GCC 4.5-2011.05-0) 4.5.4 20110505 (prerelease)
 11    (4.4.4_09.06.2010) 4.4.4
  1    (Ubuntu 4.4.3-4ubuntu5) 4.4.3
```

This information was extracted with the following shell command:

```
find /tmp/kindle-touch-5.1.2-rootfs -type f \
| xargs arm-linux-gnueabi-readelf --string-dump=.comment 2>/dev/null \
| awk '/0\]\s+GCC:/ { for (i = 1; i < 4; i++) $i = ""; print }' \
| sort | uniq --count | sort --numeric-sort --reverse
```

(4.4.4_09.06.2010) 4.4.4 is a Freescale-provided toolchain. Freescale is the producer of ARM-based system on chip installed in Kindle Touch.

There is some value in using the same toolchain that had been used for stock system (less incompatibilities in compiling process), but newer toolchins also bring some gains (more capabilities).

Install cross GNU toolchain from APT repository

Information above about available pre-built GNU toolchains was provided just for reference. Fortunately, there is APT repository with recent GCC cross-compilers, so they could be installed in Debian via standard *apt-get*. This repository is supported by emdebian.net.

Install emdebian keyring package:

```
# apt-get install emdebian-archive-keyring
```

Create file `/etc/apt/sources.list.d/emdebian.list`:

```
# emdebian.net cross toolchains
deb http://www.emdebian.org/debian/ unstable main
deb-src http://www.emdebian.org/debian/ unstable main
```

Update packages cache:

```
# apt-get update
```

Install C and C++ cross-compilers:

```
# apt-get install gcc-4.7-arm-linux-gnueabi g++-4.7-arm-linux-gnueabi
```

Now it's possible to cross-compile primitive C programs, like `helloworld.c`:

```
#include <stdio.h>

int main() {
    printf("Hello world!");
    return 0;
}
```

with command:

```
$ arm-linux-gnueabi-gcc-4.7 helloworld.c -o helloworld
```

1.2 Go

Google's original Go compiler (**gc**) is inherently a cross-compiler. It's very easy to setup cross-compiling ARM-targeted **gc**.

However, it's not so easy to enable interoperability with C code in cross-compiling **gc**. This feature is called cgo and there is no official way to enable *cgo* when cross-compiling with **gc**.

So it's necessary to perform some additional manipulations according to this post from *golang-nuts* group.

For applying of required manipulations, **gc** must be built from sources. For the reference, here are official instructions about this process: http://golang.org/doc/install/source. But step-by-step instructions also listed below.

Clone repository with Go source

Install Mercurial SCM and CA certificates, required for getting source code of **gc**, and compiler infrastructure, required for building of **gc**:

```
# apt-get install mercurial ca-certificates gcc libc6-dev
```

Enable progress Mercurial extension, so that cloning of repository will proceed with displaying of informative progress bar. Add following lines to file `.hgrc` in your home directory:

```
[extensions]
progress =
```

Clone repository with **gc** source code and update to *tip* revision:

```
$ hg clone --updaterev tip https://code.google.com/p/go
```

Tip revision is a last commited revision, so it's a bleeding-edge source. *Tip* is required, because **gc** from *release* revision (currently pointed at *go1.0.3*) doesn't support some ARM ELF relocations. This lack of support will result in errors in compilation with *cgo*.

Bootstrap Go tree

Go tree is a set of compiler, tools, standard library etc. Building process should be customized with environment variables:

- `GOROOT_FINAL` – path to directory, where built Go tree will be located. It's optional; default value is the directory where source repository had been cloned
- `GOARCH` – target architecture for compiled Go programs
- `GOOS` – target OS for compiled Go programs
- `GOARM` – this environment variable controls emitting of VFP (hardware floating point unit) instructions in compiling to ARM binaries. *5* means using only software floating point implementation, *6* – emitting VFP instructions, suitable for ARMv6 architecture, and *7* – emitting also VFP instructions available on ARMv7 architecture.

(Full list of environment variables controlling building process could be found in official instructions about building Go compiler from sources.)

```
$ cd go/src
$ GOROOT_FINAL=$HOME/go-for-kt GOARCH=arm GOOS=linux GOARM=7 ./make.bash
$ cd ..
```

Copy built Go tree to final destination (required only if `GOROOT_FINAL` was set at building):

```
$ mkdir -p $HOME/go-for-kt
$ for what in api bin doc favicon.ico lib pkg robots.txt src; do \
>   cp -r $what $HOME/go-for-kt \
> done
$ cd $HOME/go-for-kt
```

Make fake "native" compile of Go with *cgo* enabled

Now (quoting from post, referenced earlier) trick command:*cmd/go* into thinking it is doing native compile so that it will allow to enable *cgo*.

Make GNU C cross-compiler available under the name **gcc**. Create file `bin/gcc` with the following content:

```
#!/bin/sh
/usr/bin/arm-linux-gnueabi-gcc-4.7 "$@"
```

Make it executable:

```
$ chmod u+x bin/gcc
```

Rebuild **gc** with *cgo* enabled and **gcc** cross-compiler in `$PATH`:

```
$ PATH="$PWD/bin:$PATH" GOARCH=arm GOOS=linux GOARM=7 CGO_ENABLED=1 bin/go install -a -v std
```

Remove build artifacts (to considerably decrease built Go tree size without loss of any cross-compiling functionality):

- `bin/linux_arm`, `pkg/tool/linux_arm` – command tools built for executing on ARM architecture
- `pkg/obj`, `pkg/$GOHOSTOS_$GOHOSTARCH` – libraries for native (non-cross-compile) build

```
$ rm -rfv bin/linux_arm pkg/tool/linux_arm pkg/obj "pkg/$(bin/go env GOHOSTOS)_$(bin/go env GOHOSTARC
```

Wrap `bin/go`

Now, wrap real **`bin/go`** command into shell script with:

- prepending to `$PATH` path to `bin` directory with **gcc** cross-compiler
- setting appropriate environment variables to enable cross-compiling
- aborting when `$GOPATH` is undefined. By default, `$GOPATH` is set to value of `$GOROOT`, i.e. to directory containing built Go tree. It means that all user-installed and user-built packages and executables will be placed into `$GOROOT`. But it's better to place them into some other directory to make distinction between standard and user-installed tools/packages. Read also official $GOPATH documentation.

Note: `$GOARM` variable set at Go tree build time is embedded into Go linker and applied by default. It is recent addition to Go. `$GOARM` still could be set at cross-compiling to override default value.

`$CGO_ENABLED` value set at Go tree build time is also applied by default and also could be overrided.

Rename `bin/go`:

```
$ mv bin/go bin/true.go.binary
```

Create new file `bin/go` with the following content (change `$BINDIR` definition appropriately):

```
#!/bin/sh
if [ "x$1" = "xget" -o "x$1" = "xinstall" ]; then
  : ${GOPATH:?"undefined; default to GOROOT, but it's bad"}
fi
BINDIR=$HOME/go-for-kt/bin
PATH="${BINDIR}:${PATH}" GOARCH=arm GOOS=linux ${BINDIR}/true.go.binary "$@"
```

Make it executable:

```
$ chmod u+x bin/go
```

Now it's possible to cross-compile Go programs with (or without) *cgo*. by invoking `$HOME/go-for-kt/bin/go`.

There is one example of Go program with *cgo* in original Go source. It's located at `misc/cgo/life/main.go`. If source was cloned into `/tmp/go`, execute:

```
$ $HOME/go-for-kt/bin/go build /tmp/go/misc/cgo/life/main.go
```

It must proceed without errors and produce ARM-targeted **`main`** executable in current directory:

```
$ arm-linux-gnueabi-readelf --file-header main | grep Machine
  Machine:                           ARM
```

Note: Starting from GCC 4.7 there is also Go compiler included. It's available in emdebian.net repository and could be installed with command:

```
# apt-get install gccgo-4.7-arm-linux-gnueabi
```

and then Go programs could be cross-compiled with:

```
$ arm-linunx-gnueabi-gccgo-4.7 program.go -o program
```

(Look also at official instructions about using of gccgo.)

gccgo from GCC 4.7.x couldn't be used with **go** tool from *tip* revision of **gc** source (like *go -compiler gccgo*), because *tip* **go** expects some options introduced only in GCC 4.8 (which is currently not available in *emdebian.net* repository).

---

2 License

Index