embsys 110, lesson one

Ken Smith

kgsmith@gmail.com

ABSTRACT

There is more than one way to do it for most values of ’it’. This lesson introduces an alternative development environment which you are free to use for the remainder of the course. For some, it will not be as pleasant as the environment you’ve been using for the last two courses. For others, it will be just what the doctor ordered. Different strokes for different folks. Regardless, I hope to add a few tools to your toolbelt that you will either prefer or may encounter in your future work.

primer

There are only a few things you actually need to develop for a deeply embedded MCU. You need a way to edit code, a cross compiler toolchain and a way to get the program to the MCU. If you’re lucky, you will have a way of single-stepping the code once its on the device. The rest of the tools become invaluable as you gain confidence and familiarity with them but none of them are as essential as these.

the venerable command line

The GUI can be a wonderful boon. It can also be an obfuscating crutch, hiding the actual goings-on behind cleverly drawn buttons and sliders. The things that are going on are often uncomplicated. With only a bit of memorization, you might find that you are more efficient on the command line. Regardless, it is useful to understand what is actually happening when you click a button in a GUI so you can more quickly recover from unexpected behavior or failures in the tools.

linux

As development environments go, I haven’t found anything that beats Linux in terms of tools. There are tools for inspecting every bit of the operating system itself and their are myriad tools for playing with things external to the system. The learning curve is relatively sharp but the reward in terms of productivity and self-sufficiency make it worth it imho.

There are many variants (distros) of Linux out there. The most popular for newcomers is Ubuntu which descends from Debian. In the enterprise, Redhat-based operating systems such as Fedora and CentOS have a lot of mindshare. For tinkerers and students of Linux, Gentoo and Arch offer transparency into the inner workings and greater flexibilty with what the final system is optimized to do. Of these last two, Arch is newer and strikes a fair balance between supporting the user and giving the user plenty of rope.

assignment one

the vm

In this assignment, you'll deploy Arch in a virtual machine (VM) on your Windows, Mac OS, or Linux machine. Obviously, if you already use Linux, you can install the tools natively but I encourage you to create a VM unless you're already comfortable with doing so. It's nice to be fluent in VM construction, snapshotting, and cloning for those times when you need to support diverse platforms or need to do interesting things with a virtualizable OS.

Happily, a blogger wrote a tutorial recently on setting up Arch in a Virtual Box VM. You can feel free to use VMWare or Parallels if you already prefer either one. If you don't, Virtual Box is free and is fairly mature and stable. Here's the blog article.

http://wideaperture.net/blog/?p=3851

The author is pleasant to read and takes you on a leisurely walk through the process with numerous useful excursions to explain things in more detail. For the "what's in it for me" crowd (of which I'm a card carrying member), feel free to skip ahead to "Step One: Assemble the Components". If something mystifies you, you might want to go back to read the preamble or just ask in the forums.

I deviated from the instructions in a few ways. You might not have to but here are a few notes I took along the way to getting my environment set up.

syslinux-install_update didn't accept the -a parameter which sets the boot flag of the first partition. I ran it with -i and -m only. Later in the process, he has you run sgdisk with the right flags to do what -a wouldn't do.

useradd -d didn't create my home directory for me. (I accidentally ran it once before I noticed the -d -- a downside to being perhaps too much about "what's in it for me".) I had to

```
   mkdir /home/user
   chown user:user /home/user
```

Where 'user' in my case is 'ken'.

I also commented in the comment section of the blog post but I encourage you not to directly edit /etc/sudoers. There is a program which checks your configuration before it commits the changes to make sure you haven't broken your ability to sudo. This is especially important on distros like Ubuntu which don't have you set a root password by default and rely on the sudo mechanism for administrator level access to the machine. Do this, instead.

```
   EDITOR=/usr/bin/nano visudo
```

Rather than install Gnome, I installed xfce. It is lighter weight than Gnome and I prefer it. Use whichever you wish. If you want to try xfce, install xfce4 and xfce4-goodies, accepting the defaults for the questions that follow. Then read this.

https://wiki.archlinux.org/index.php/Xfce

Start xfce with startxfce4.

In addition to the packages the tutorial has you install, please also install these.

- vim
  - the world's best editor
- screen
  - terminal multiplexing for the old-at-heart
- tmux
  - terminal multiplexing for those who don't already use screen
- zsh
  - the world's best shell
- git
  - the world's best version control system
- openssh
  - the world's best remote shell
- whois
  - check the identity of an IP or FQDN
- xclip
  - commandline access to the clipboard
- mpfr
- mpc
- gmp

- isl
- cloog
  - These five are required to build our cross compiler
- wget
  - Used by the cross compiler build process
- openocd
  - Our friend from Windows/Eclipse prepackaged for us by the Arch people
- cgdb
  - You have to install this from "the AUR" (https://aur.archlinux.org/)
    - Download the tarball
    - cd Downloads
    - tar xzvf cgdb.tar.gz
    - cd cgdb
    - less PKGBUILD
      - look to make sure everything looks cool (no h4x0r5 trying to pwn you)
    - makepkg -s
    - Follow dialogs
    - sudo pacman -U cgdb-0.6.7-1-x86_64.pkg.tar.gz
      - or whatever version/architecture it built.
        - run 'ls' to find out

As usual, you just run this to install any of the above.

```
pacman -S package
```

Replace 'package' with the name in the list. You can list them all on one command line if you want. That will give you plenty of time to grab a coffee while the gnomes at Arch do all the heavy lifting.

If you get 404 errors when pacman tries to download the packages, run this.

```
pacman -Sy package
```

The '-y' flag instructs pacman to update its package lists before looking for the packages. You should only need to do that once in a while.

github

Cloning from github requires a free account there and some configuration. Once you have your account, do this in a terminal.

```
ssh-keygen
```

Github recommends using a passphrase.

https://help.github.com/articles/working-with-ssh-key-passphrases

The passphrase is not technically required so whether you use one is up to you. Once you have your cryptographic identity (RSA keypair), give github the public part by logging in to github, going to account settings, clicking on ssh keys, and clicking add ssh key. In a terminal do this.

```
cat ~/.ssh/id_rsa.pub | xclip
```

This copies the public part of your RSA keypair to the X windows clipboard. In the github add key's dialog, middle click* in the text box to paste the key and submit. If what ends up in the text box doesn't look something like this,

*If you don't have a middle click, then try this.

```
cat ~/.ssh/id_rsa.pub
```

Then, highlight the stuff that is displayed and right click (option click, etc.) to bring up the contextual menu in the terminal. Select copy, then right click in the text box, and choose paste. If that doesn't work, google it out or ask in the forums.

a note about cryptographic identities

A cryptographic identity, in our case an RSA keypair, is easy to get wrong. For example, if in the previous step, you accidentally copied id_rsa instead of id_rsa.pub, you gave github your private key. That means that they can impersonate you. Luckily there is an easy way to recover. Delete your id_rsa and id_rsa.pub and start again. If you've already been using your crypto ID for a while, then you'd want to update all the places you had been using it with using your new id_rsa.pub the same way you'd update your password if someone had gotten hold of it. The great thing about the cryptographic identity is that, once it is set up, it provides a strong mechanism to do passwordless authentication with a remote server over an untrusted connection like the internet. To learn more, search for "cryptographic identity", "PKI", "X.509", "RSA algorithm", "elliptic curve cryptography", etc.

building the cross compiler toolchain

This is actually a big piece of work. I've struggled with this for years and years using the numerous tutorials out there until I finally codified it into something which seems to work ok for now. The process to build GCC is documented here.

http://gcc.gnu.org/install/

Unfortunately, when you're building a cross compiler, things get even more dicey than they already are. We've already installed all the prerequisites so the process becomes.

- build the bootstrap compiler
- build binutils
- build a standard library (eg. newlib)
- use these to build the final compiler
- build gdb

It's gnarly. A while back, I wrote a makefile that (hopefully) still works. Get it from github.

```
git clone git://github.com/kensmith/build-arm-none-eabi-gxx.git
```

You'll notice it's called arm-none-eabi and not arm-elf. arm-elf is deprecated and is not binary compatible with arm-none-eabi. To learn more about the new EABI, read "Procedure Call Standard for the ARM Architecture".

http://infocenter.arm.com/help/topic/com.arm.doc.ihi0042e/IHI0042E_aapcs.pdf

dns errors with github

I got this error when I tried to access github.

"fatal: Unable to look up github.com (port 9418) Name or service not known)"

I'm pretty sure it was something wrong with the DNS on the internet connection at the microbrewery where I was testing all this out. DNS is managed by /etc/resolv.conf which is in turn managed by dhcpcd or other network autoconfiguration mechanism you use. To temporarily work around this issue, I hand edited /etc/resolv.conf and changed it from this.

```
# Generated by dhcpcd from wlan0
nameserver 192.168.0.1
```

to this

```
# Generated by dhcpcd from wlan0
nameserver 8.8.8.8
nameserver 192.168.0.1
```

8.8.8.8 is a public DNS server that Google owns. Find out more like this.

```
whois 8.8.8.8
```

building the cross compiler toolchain (cont.)

Next, build the toolchain.

```
cd build-arm-none-eabi-gxx
make
```

This takes quite a while, especially on my little machine running in virtualization. Once it’s done, you’ll know it worked if you can do this.

```
arm-none-eabi-g++ -v
```

and get this or something close to it.

```
Using built-in specs.
COLLECT_GCC=arm-none-eabi-g++
COLLECT_LTO_WRAPPER=/usr/local/libexec/gcc/arm-none-eabi/4.8.0/lto-wrapper
Target: arm-none-eabi
Configured with: ../../gcc-4.8.0/configure
--target=arm-none-eabi --with-newlib --disable-threads
--disable-libmudflap --disable-libssp --disable-libgomp
--disable-libquadmath --disable-shared --disable-zlib
--disable-nls --enable-multilib --enable-interwork
--enable-languages=c,c++
Thread model: single
gcc version 4.8.0 (GCC)
```

testing the tools

Grab this sample program.

```
git clone git://github.com/kensmith/embsys.git
cd embsys/hello-embsys
```

You should be able to build this now.

```
make
```

You’ll see this.

```
mkdir -p build
assm crt.s crt.o
comp init.cpp init.o
comp print.cpp print.o
comp main.cpp main.o
comp led.cpp led.o
link crt.o print.o init.o main.o led.o app.elf
copy app.elf app.bin
copy app.elf app.lst
```

You can run this to see the actual commands.

```
make show=t
```

Plug in your ARM-USB-OCD and attach it to your VM from the Devices->USB Devices menu. In another terminal (or in another GNU screen / tmux window) run this.

```
sudo openocd
```

You should see something like this.

```
Open On-Chip Debugger 0.7.0 (2013-05-05-22:01)
Licensed under GNU GPL v2
For bug reports, read
        http://openocd.sourceforge.net/doc/doxygen/bugs.html
Info : only one transport option; autoselect 'jtag'
Runtime Error: openocd.cfg:3: invalid command name "arm7_9"
in procedure 'script'
at file "embedded:startup.tcl", line 58
at file "openocd.cfg", line 3
Warning - assuming default core clock 4MHz! Flashing may
fail if actual core clock is different.
trst_and_srst separate srst_gates_jtag trst_push_pull
srst_open_drain connect_deassert_srst
adapter_nsrst_delay: 100
jtag_ntrst_delay: 100
adapter speed: 500 kHz
Info : clock speed 500 kHz
Info : JTAG tap: lpc2378.cpu tap/device found: 0x4f1f0f0f
(mfg: 0x787, part: 0xf1f0, ver: 0x4)
Info : Embedded ICE version 7
Error: EmbeddedICE v7 handling might be broken
Info : lpc2378.cpu: hardware has 2 breakpoint/watchpoint
units
```

Back in the original window where you built the sample code, do this.

```
make flash
```

It should pause and you should see activity in the openocd window. When it finishes, which should be in a couple seconds, run the debugger and check it out.

```
cgdb -d arm-none-eabi-gdb
break main
reboot
```

You should see your breakpoint at the top of main.

```
c
```

This should let the program run and you should see the LED blinking as it did in the first assignment.

'play is the highest form of research' - albert e.

From here, I invite you to play. Rebuild old assignments using these tools. This toolchain supports most of C++11 and the example has been reworked using some of its features. (See embsys/hello-embsys/doc/cpp-register-access.pdf and the slides in a subdirectory there for details.)

You may of course continue using Yagarto/Eclipse to do your assignments but you are welcome to use these updated tools as well. The version of GCC you just built was released in March 2013. Our yagarto toolchain uses GCC 4.2.2 which was released in October 2007. Much has transpired in the past 5.5 years. See the changelogs for details.