Ubuntu Setup

Lab Book

eMagii
http://www.emagii.com

Free Electrons
http://free-electrons.com

May 14, 2014

About this document

Updates to this document can be found on http://www.emagii.com/doc/training/sysdev/.

This document was generated from LaTeX sources found on https://github.com/emagii/training-materials.git.

More details about our training sessions can be found on http://www.emagii.com/training.

Copying this document

Corrections, suggestions, contributions and translations are welcome!

Setting up Ubuntu

General Setup of Ubuntu

Install Synaptic

```
$ sudo apt-get install synaptic
$ sudo su
$ synaptic&
```

Start `synaptic` and install

- `gnome`
- `nautilus-open-terminal`
- `git`
- `git-core`
- `samba`
- `system-config-samba`

Log out

Click on the white `Ubuntu` button when you get the login screen and select the `Gnome Classic` option. Log again.

`Gnome Classic` will from now be your default.

Open a terminal

Since you installed `nautilus-open-terminal`, you can open a terminal by right clicking the mouse, and select the terminal.

Make sure bash is the default shell

The normal Ubuntu installation uses the `dash` shell which won't work.

Changed to the `bash` shell.

```
$ cd /bin/
$ ls -l sh
lrwxrwxrwx 1 root root 9 dec  6 15:25 sh -> /bin/dash
$ sudo unlink sh
$ sudo ln -s /bin/bash sh
$ ls -l sh
lrwxrwxrwx 1 root root 9 dec  6 15:25 sh -> /bin/bash
```


full-sysdev-labs.pdf

Generate ssh keys

If you already have `rsa` keys in the `$HOME/.ssh` directory, you can skip this step.

If not, you generate the keys like this (Make sure you are not running as super-user)

```
$ ssh-keygen -t rsa
```

Use the default location and provide a password (twice).

This will generate

- Private Key: `$HOME/.ssh/id_rsa`
- Public Key: `$HOME/.ssh/id_rsa.pub`

The **Private Key** should **never** be give out to anyone else.

Update Ubuntu to the latest package versions

Run the update manager, to update the machine.

This will take some time.

`Program->System Tools->Administration->Update`

You probably have to restart the computer afterwards.

Setting up the git client

Objective: Get a working git

After this lab, you will be able to work with the **git** source code control system

Install the git client

If you do not have git installed, you need to do this now.

```
sudo apt-get install git git-core
```

You should now set up your git personal information

Use the example below, but with your own name/email:

```
git config --global user.name "Allan K Luring"
git config --global user.email "allan@luring.com"
```

You can get help on git, using the `git help` command.

Training setup

Download files and directories used in practical labs

Install lab data

For the different labs in the training, your instructor has prepared a set of data (kernel images, kernel configurations, root filesystems and more). Clone the lab directory to your home directory.

```
cd
git clone https://github.com/emagii/Training-Labs.git felabs
```

Lab data are now available in an `~/felabs/sysdev` directory. For each lab there is a directory containing various data. This directory will also be used as working space for each lab, so that the files that you produce during each lab are kept separate.

Install extra packages

You will need to have a number of packages installed on your machine.

Go to the `~/felabs/sysdev` directory and install required packages using the Makefile.

```
make prepare
```

Since the install requires `root` privilegues, you will have to supply the `root` password.

Configure Your lab network

Edit the `host.mk` and change the **SERVER_IP** and **IPADDR** variables if they conflict with your main network during the lab.

Check with `ifconfig` if you do not know which network you are using.

More guidelines

Can be useful throughout any of the labs

- Read instructions and tips carefully. Lots of people make mistakes or waste time because they missed an explanation or a guideline.
- Always read error messages carefully, in particular the first one which is issued. Some people stumble on very simple errors just because they specified a wrong file path and didn't pay enough attention to the corresponding error message.
- Never stay stuck with a strange problem more than 5 minutes. Show your problem to your colleagues or to the instructor.
- You should only use the `root` user for operations that require super-user privileges, such as: mounting a file system, loading a kernel module, changing file ownership, configuring the network. Most regular tasks (such as downloading, extracting sources, compiling...) can be done as a regular user.
- If you ran commands from a root shell by mistake, your regular user may no longer be able to handle the corresponding generated files. In this case, use the `chown -R` command to give the new files back to your regular user.
  Example: `chown -R myuser:myuser linux-3.4`

Samba Setup

Open up the lab directory

Install Samba

If you did not run `make prepare`, install `samba` and `system-config-samba` using `synaptic` or `apt-get`,

Create the `$HOME/common` directory.

Using the Menusystem, open the `samba` configuration utility.

```
Program->System Tools->Administration->Samba
```

In the server configuration, make sure your workgroup is **emagii** during the labs.

Export the `common` directory.

Make it visible and writeable and give access to everyone.

If you want the teacher to be able to access your lab directory, you should create the user `ulf`, and export the `~/felabs/sysdev` directory, making it visible and writeable to `ulf`.

After reconfiguring samba, you need to restart with the new configuration.

```
sudo service smbd restart
sudo service nmbd restart
```


Setting up the serial communication

Objective: Get a working serial communication for the console

After this lab, you will be able to communicate with your **Beaglebone Black** over the console

Install the picocom program

If you did not run `make prepare` in `~/felabs/sysdev` , you need to do this now.

This will install `picocom` and make your user belong to the `dialout` group to be allowed to write to the serial console:

Alternatively:

```
sudo apt-get install picocom
sudo adduser ${USER} dialout
```

Setting up serial communication with the board

Make sure that the USB-to-Serial cable to the **Beaglebone Black** is disconnected from your computer.

Check if there are any other USB serial ports in the system

```
ls /dev/ttyU*
```

Plug the **Beaglebone Black** on your computer using the provided USB-to-serial cable. When plugged-in, a serial port should appear as `/dev/ttyUSB0` if there are no other USB - Serial ports. Otherwise find out which serial port was just activated.

You can also see this device appear by looking at the output of `dmesg`.

You need to log out and in again for the group change to be effective.

Run

```
picocom -b 115200 /dev/ttyUSB0
```

to start serial communication on `/dev/ttyUSB0`, with a baudrate of 115200.

If you wish to exit `picocom`, press `[Ctrl][a]` followed by `[Ctrl][x]`.

SD-card setup

Objectives: Set up an SD-card for use in later labs

MMC/SD card setup

The Beaglebone can boot using files from a FAT filesystem on the MMC/SD card. However, the MMC/SD card must be carefully partitioned, and the filesystem carefully created in order to be recognized by the ROM monitor. Here are special instructions to format an MMC/SD card for the Sitara-based platforms.

First, clean out your system log buffer by `sudo dmesg -c`.

Then list all disk devices by `ls /dev/sd*`.

Connect your card reader to your workstation, with the MMC/SD card inside and again list all the disk devices by `ls /dev/sd*`.

If you see a new disk device appearing, that will be the new SD-card. If you see several new devices like `ls /dev/sde /dev/sde1 /dev/sde2`, this is because the SD-card has several partitions.

If your PC has an internal MMC/SD card reader, the device may also been seen as `/dev/mmcblk0`, and the first partition as `mmcblk0p1`. [1]. You will see that the MMC/SD card is seen in the same way by the Beaglebone Black board.

If you still fail to detect the disk, then type the `dmesg` command to see which device is used by your workstation. In case the device is `/dev/sde`, you will see something like:

```
sd 3:0:0:0: [sde] 3842048 512-byte hardware sectors: (1.96 GB/1.83 GiB)
```

In the following instructions, we will assume that your MMC/SD card is seen as `/dev/sde` by your PC workstation.

> **Caution: read this carefully before proceeding. You could destroy existing partitions on your PC!**
> **Do not make the confusion between the device that is used by your board to represent your MMC/SD disk (probably `/dev/sda`), and the device that your workstation uses when the card reader is inserted (probably `/dev/sde`).**
> **So, don't use the `/dev/sda` device to reflash your MMC disk from your workstation. People have already destroyed their Windows partition by making this mistake.**

You can also run `cat /proc/partitions` to list all block devices in your system. Again, make sure to distinguish the SD/MMC card from the hard drive of your development workstation!

[1] This is not always the case with internal MMC/SD card readers. On some PCs, such devices are behind an internal USB bus, and thus are visible in the same way external card readers are

Type the `mount` command to check your currently mounted partitions. If MMC/SD partitions are mounted, unmount them:

```
$ sudo umount /dev/sde1
$ sudo umount /dev/sde2
...
```

Now, clear possible MMC/SD card contents remaining from previous training sessions:

```
$ sudo dd if=/dev/zero of=/dev/sde bs=1M count=256
```

As we explained earlier, the TI Sitara ROM monitor needs special partition geometry settings to read partition contents. The MMC/SD card must have 255 heads and 63 sectors.

Let's use the `cfdisk` command to create a first partition with these settings:

```
sudo cfdisk -h 255 -s 63 /dev/sde
```

In the `cfdisk` interface create three primary partitions, starting from the beginning:

- BOOT: 512 MB size, a `Bootable` type and a `0C` type (`W95 FAT32 (LBA)`)
- ROOTFS: 2048 MB size and a `83` type (`Linux`)
- DATA: 512 MB size and a `83` type (`Linux`)

Press `Write` when you are done.

If you used `fdisk` before, you should find `cfdisk` much more convenient!

Format the first partition to FAT32, with the `boot` label (name):

```
sudo mkfs.vfat -n BOOT -F 32 /dev/sde1
```

Then format the second partition to `EXT4`.

```
sudo mkfs -t ext4 -L rootfs /dev/sde2
```

Format the third and last partition to `EXT4`.

```
sudo mkfs -t ext4 -L data /dev/sde3
```

Then, remove and insert your card again.

Your MMC/SD card is ready for use.

Setting up a TFTP Server

Objectives: Set up TFTP communication with the development workstation.

Install the TFTPD server

Later on, we will transfer files from the development workstation to the board using the TFTP protocol, which works on top of an Ethernet connection.

Normally, if you did `make prepare` before, you should have the TFTP server running on your system. Otherwise do:

```
sudo apt-get install tftpd tftp xinetd
```

Configuring the TFTPD server

The configuration file for `tftpd` is `/etc/xinetd.d/tftp`

An example `tftp` file is in `~/felabs/sysdev` /network

```
service tftp
{
        protocol        = udp
        port            = 69
        socket_type     = dgram
        wait            = yes
        user            = nobody
        server          = /usr/sbin/in.tftpd
        server_args     = /tftpboot
        disable         = no
}
```

Create the configuration file.

The default TFTP directory is `/tftpboot`

if it doesnt exist, create it and make it owned by 'nobody' and make sure that `tftpboot` is easily accessible

```
sudo    mkdir -p        /tftpboot
sudo    chown -R nobody /tftpboot
sudo    chmod -R 777    /tftpboot
```

Restart tftpd by:

```
sudo service xinetd restart
```


Testing the TFTPD server

Create a file in `tftpboot`:

```
cd /tftpboot
echo 111 > testfile.txt
```

Start tftp and get the file

```
cd
tftp localhost

tftp> get testfile.txt
Sent 5 bytes in 0.0 seconds

tftp> quit

cat testfile.txt
```


Setting up a Startech USB - Ethernet adapter

Objective: Get a dedicated network port for communication with the **Beaglebone Black**

After this lab, you have prepared for a second network port allowing you to nfs-mount the **Beaglebone Black** root filesystem without changing the network settings of the primary network port.

Prerequisites

You need a **Startech USB31000SW USB Ethernet** adapter to use the port, but the driver can be installed without the Adapter.

It is assumed, that your system is setup to build kernel modules.

This means that kernel source build directory must be present

```
ls /lib/modules/`uname -r`/build
```

Preparing to build kernel drivers

If you did not run `make prepare` in `~/felabs/sysdev` , you need to do this now.

Build and install the kernel driver

Go to the `~/felabs/sysdev` directory

The `network/startech` directory contains a tarball with the driver.

The tarball has been downloaded from the Startech website, and will be slightly modified using a patch to allow it to build.

From the `~/felabs/sysdev` directory, You can compile and install the driver by:

```
sudo make -C network startech-usb
```

Build and install the kernel driver outside the lab

Extract the tarball, and enter the source directory.

Apply the patches from the `../patches` directory

```
make
sudo make install
```


Setting up the NFS server

Objective: Prepare the host for NFS booting

Install the NFS Server

`make prepare` should have installed the NFS server. If you do not have an NFS server installed, install it by

```
sudo apt-get install nfs-kernel-server
```

Create the root file system directory

Create a `rootfs` directory in your $HOME directory. This directory will later be used to store the contents of our new root filesystem.

```
mkdir -p $(HOME)/rootfs
chmod 777 $(HOME)/rootfs
```

Configure the NFS server

The NFS configuration file (`/etc/exports`) needs to be modified. Edit the file as `root` and add the following line: (Replace `/home/ulf` with your own home directory)

```
/home/ulf/rootfs *(rw,sync,no_root_squash,no_subtree_check)
```

The format for the line is

```
<directory> <ip-adress>(<options>)
```

The IP address '*' means that the NFS server will allow any computer to connect. You can replace it with the IP address of the **Beaglebone Black** .

Make sure that the path and the options are on the same line. Also make sure that there is no space between the IP address and the NFS options, otherwise default options will be used for this IP address, causing your root filesystem to be read-only.

Then, restart the NFS server:

```
sudo service nfs-kernel-server restart
```

alternatively:

```
sudo exportfs -av
```

You can test your NFS setup by:

```
sudo mount -t nfs localhost:/home/ulf/rootfs
```

Replace `ulf` with your username.

Extra Lab: Building a cross-compiling toolchain using Yocto-1.6

Objective: Learn how build a well tested cross-compiling toolchain using the eglibc C library

This is an optional exercise for home

If you only want to download a the result of this lab, go to the next chapter.

On a real fast machine like a Dell T7500 with 2 Xeon X5670 (2 x 6 cores/24 threads @ 2.93 GHz/96GB RAM), `bitbake` will run for an hour to complete the build.

On a Core-i7 Quad-Core laptop, like the Dell E6520 with Core i7 2760m/16GB RAM, you should expect 3-4 hours.

Expect a long, long time on a Core 2 Duo with small amount of RAM.

Yocto shows a stack of current executing tasks, as well as to total number of tasks, so you quickly get an idea about the build time.

After this lab, you will be able to:

- Generate a modern toolchain for the Beaglebone.

Setup

Go to the `~/felabs/sysdev` directory.

Install needed packages

Install the packages needed for this lab, if you havent done this before:

```
make prepare
```

Getting Yocto

```
git clone git://git.yoctoproject.org/poky poky-daisy
```

Then checkout the daisy branch (Yocto-1.6).

```
cd poky-daisy
git checkout -b daisy origin/daisy
```


Configuring Yocto

Once you have Yocto installed, you should configure it for your board.

```
cd poky-daisy
. oe-init-build-env build-beaglebone
```

This will create the `build-beaglebone` directory

Check the `build-beaglebone/conf` configuration directory.

An important file is `local.conf`

Configuring Yocto in local.conf

You should edit the `local.conf` to optimize for your own machine.

Edit the MACHINE variable to set it to the ”beaglebone”.

This will ensure that the cross compiler will build an ARMv7 toolchain with `NEON` support.

```
# There are also the following hardware board target machines included for
# demonstration purposes:
#
MACHINE ?= "beaglebone"
```

The default is to build a toolchain without libraries for static linking.

We will use static linking, so we need to add support by:

```
# Add libraries for static linking
#
IMAGE_INSTALL_append = " eglibc-staticdev"
SDKIMAGE_FEATURES += "staticdev-pkgs dev-pkgs"
```

A good place is right after the definition of `EXTRA_IMAGE_FEATURES`.

If you use a common download directory, you might want to change the DL_DIR variable.

```
# The default is a downloads directory under TOPDIR which is the build directory.
#
#DL_DIR ?= "${TOPDIR}/downloads"
```

If you have access with a DVD/USB memory with the tarballs, then you may want to copy those to the `build-beaglebone/downloads` directory to speed up the build.

Change the package mechanism to ipk

```
# Package Management configuration
#
# This variable lists which packaging formats to enable. Multiple package backends
# can be enabled at once and the first item listed in the variable will be used
# to generate the root filesystems.
# Options are:
#  - 'package_deb' for debian style deb files
#  - 'package_ipk' for ipk files are used by opkg (a debian style embedded package r
#  - 'package_rpm' for rpm style packages
# E.g.: PACKAGE_CLASSES ?= "package_rpm package_deb package_ipk"
# We default to rpm:
PACKAGE_CLASSES ?= "package_ipk"
```

Building the Cross-Compiler

Yocto has the ability to generate a Software Development Kit (SDK) for an image.

By generating an SDK, you get a cross-compiler with everything needed to build further applications outside the Yocto build system.

The SDK contains all the header files for the applications and libraries included in the image. [2]

`time bitbake core-image-minimal`

followed by

`time bitbake core-image-minimal -c populate_sdk`

`core-image-minimal` is just that, so you may want to build a more complete image/toolchain.

`time bitbake core-image-sato`

followed by

`time bitbake core-image-sato -c populate_sdk`

The end result will be a script file in `~/felabs/sysdev/poky/build/tmp/deploy/sdk`

It is called something similar to:

`poky-eglibc-x86_64-core-image-minimal-armv7a-vfp-neon-toolchain-1.6.sh`

(The version number may differ)

[2] Some documentation recommends to do **bitbake meta-toolchain** or **bitbake meta-toolchain-sdk** but for some reason, the static libraries does not seem to be included when you do it this way

Installing a cross-compiling toolchain built by Yocto

Objective: Install a well tested cross-compiling toolchain using the eglibc C library

Getting the Yocto SDK

If you have done the Extra Lab: `Building a cross-compiling toolchain using Yocto`, you should have the file:

`poky-eglibc-x86_64-core-image-minimal-armv7a-vfp-neon-toolchain-1.6.sh`

or

`poky-eglibc-x86_64-core-image-sato-armv7a-vfp-neon-toolchain-1.6.sh`

or similar (the version number may differ)

If not, You can download the toolchain from `ftp://ftp.emagii.com/pub/training/tools` or copy if from the DVD/USB stick, if you got it from your teacher.

Install the latest version, and preferably `core-image-sato` over `core-image-minimal`.

Make sure this script is executable (Normally it should be). chnmod

Installing the Cross-Compiler

Typically the compiler will be installed in `/opt/poky/1.6`

You normally do not have write access to `/opt` so you should create the install directory before you install the cross-compiler.

You will be asked where to install the cross-compiler, so an alternative is to change the installation directory to somewhere where you do have write access.

```
sudo mkdir -p /opt/poky
sudo chown ${USER} /opt/poky
```

Run the installation script.

Edit the `environment-setup-armv7a-vfp-neon-poky-linux-gnueabi` file, in the installation directory adding:

You will need to fix LDFLAGS, which will cause problems with building U-Boot later.

Comment away the existing definition of LDFLAGS and replace it with an empty defintion.

```
export LDFLAGS=""
```

From the install directory, copy the `environment-setup-armv7a-vfp-neon-poky-linux-gnueabi` to `toolchain.sh` in the `~/felabs/sysdev` directory, and make sure it is executable.

If you are running any of the extra labs for creating a toolchain they may overwrite this file, so beware that you are not running the wrong cross-compiler. You could name it something else like `yocto-toolchain.sh`

Now you can use the cross compiler by just sourcing this file.

```
source toolchain.sh
```


Extra Lab: Installing the Codesourcery Lite toolchain

Objective: Setup the Mentor Sourcery Codebench Lite cross-compiling toolchain

This is an optional exercise for home

Some people like to use this toolchain, so it is included for reference. It will not be used during the labs.

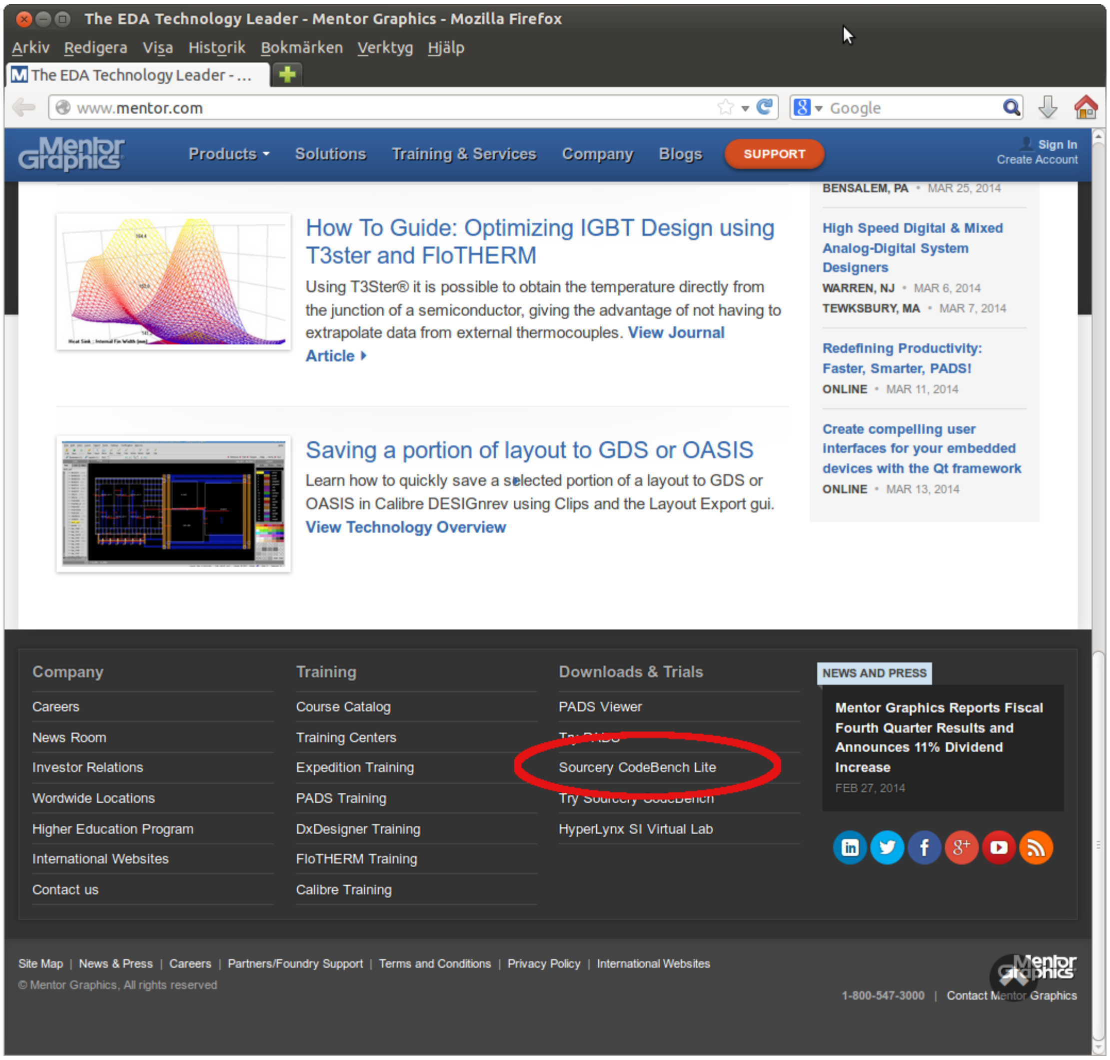
Sourcery Codebench Lite

The Sourcery Codebench Lite, is a free-of-charge high-quality toolchain, which is available with multiple library options.

There is a low cost commercial version, which contains an Eclipse Environment.

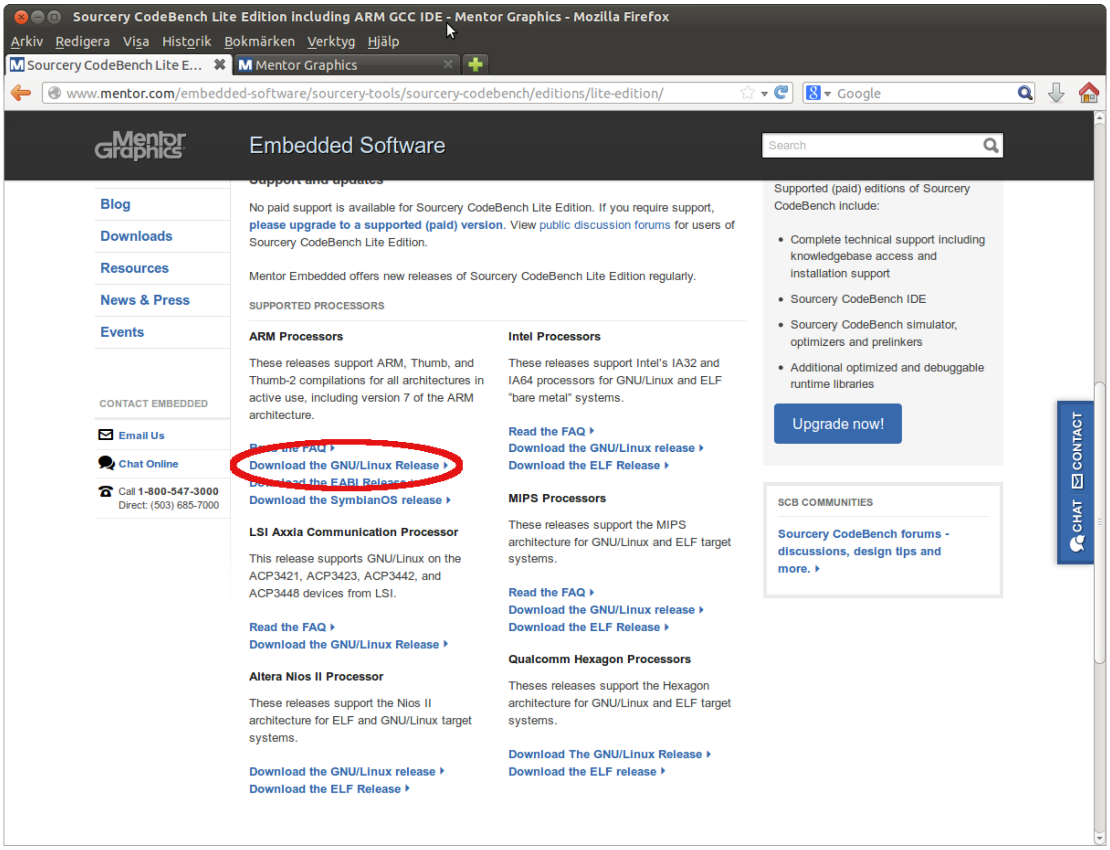
Getting the Toolchain

Open a Browser and go to the Mentor Graphics Homepage

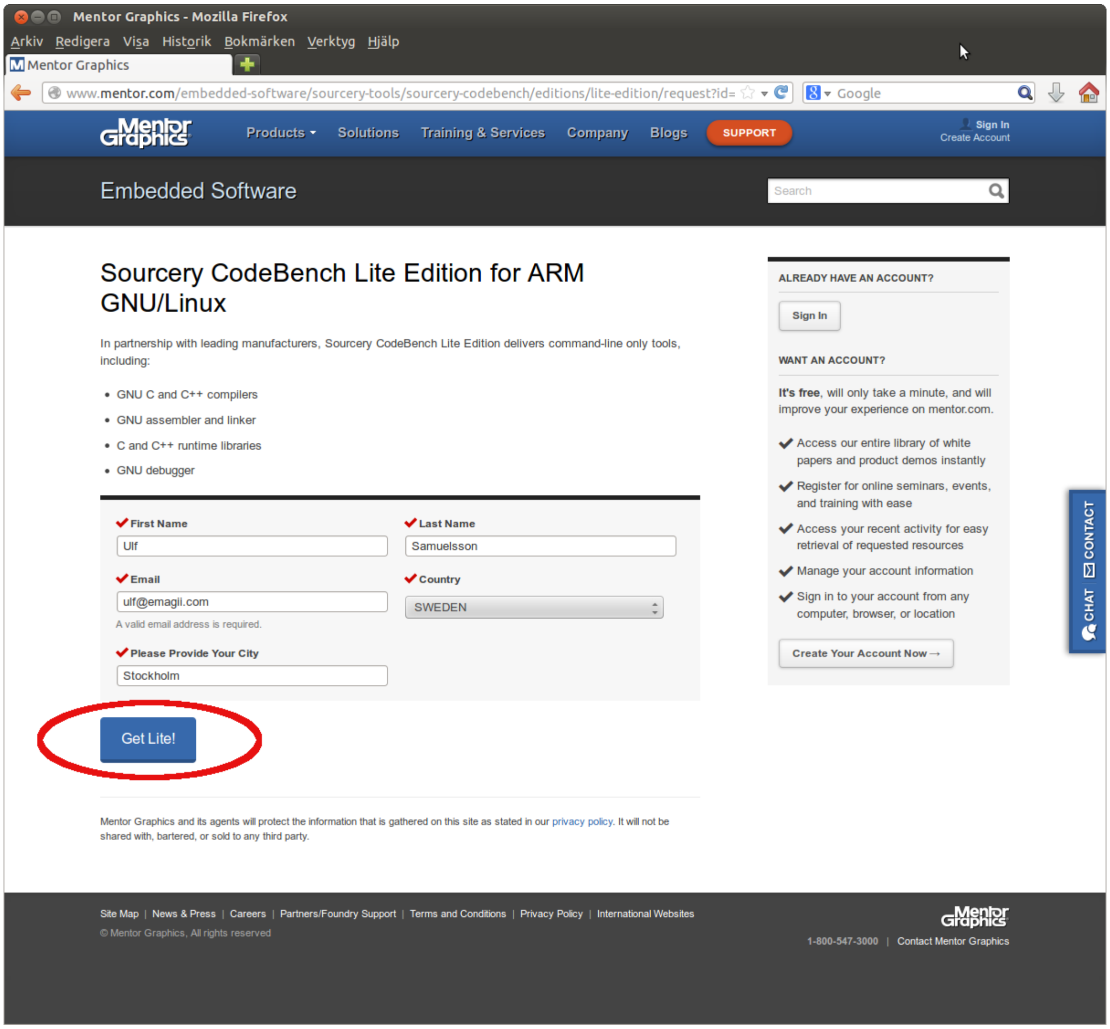


Go to the bottom and open the Sourcery Codebench Lite page

Once on the page, select the Download the Gnu/Linux Release for the ARM processor.

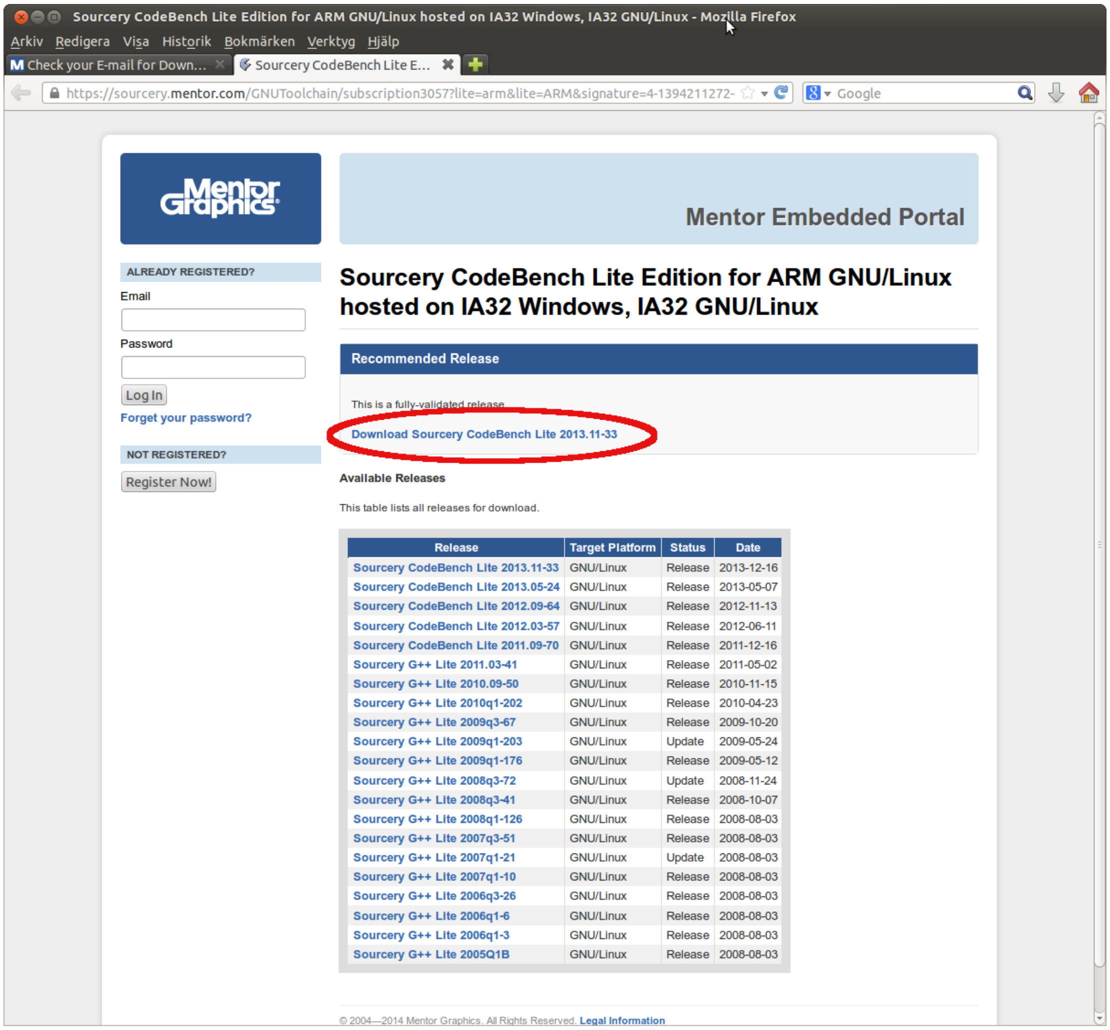


Fill in your personal details and click the **Get Lite!** button.

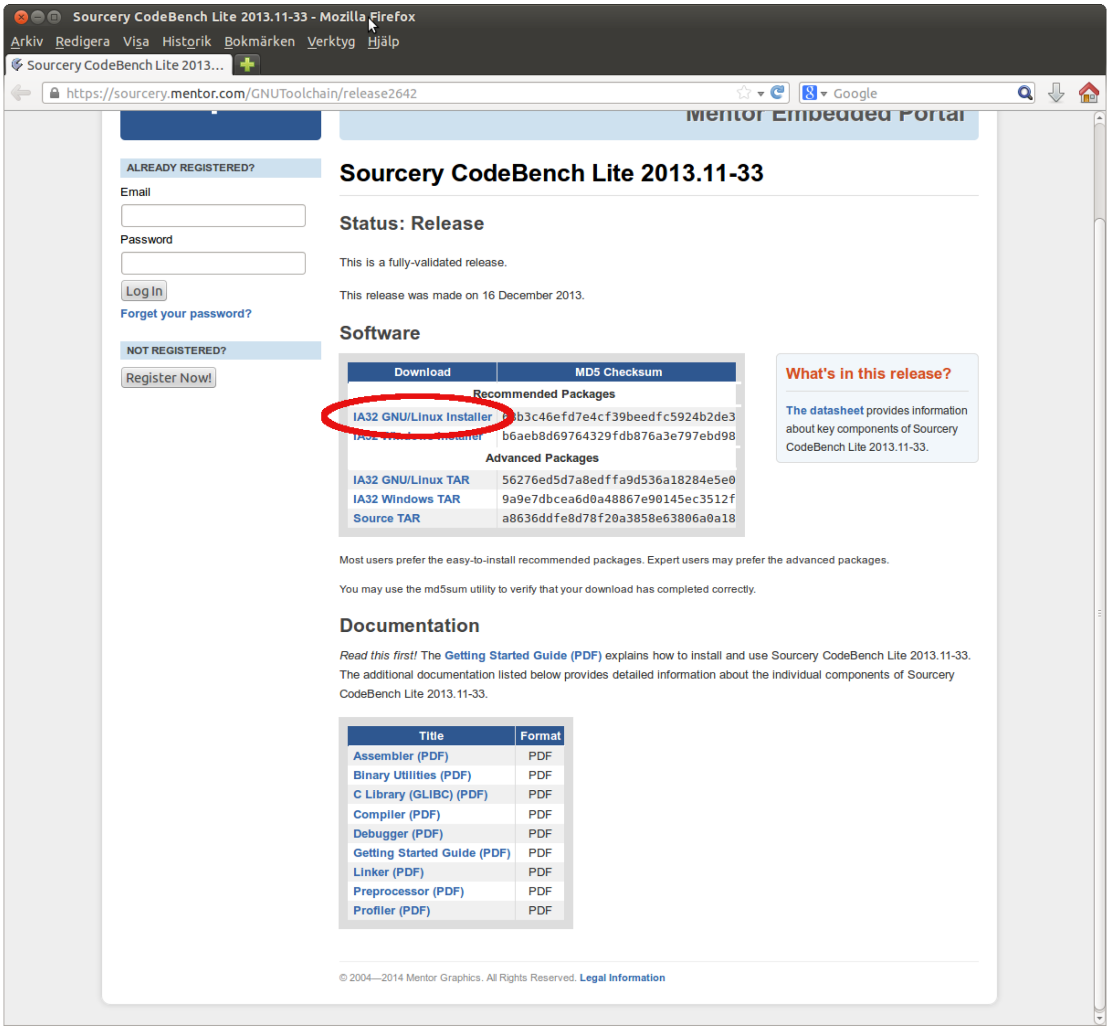


You will get a mail with the download location. Click on the link in the mail and you will open a page from where you can select your download.

Select the Sourcery Codebench Lite 2013.11-33 version



Right Click the **IA32 GNU/Linux Installer** and save at an appropriate place.



Getting-Started.pdf contains the installation guide and should also be downloads.

You may also want to download the rest of the documentation.

Installing the Sourcery Codebench Lite

Go to the directory where you downloaded the Installer and make it executable.

```
chmod a+x arm-2013.11-33-arm-none-linux-gnueabi.bin
```

Run the installer

```
./arm-2013.11-33-arm-none-linux-gnueabi.bin
```

The installer should be pretty obvious. If not, use the **Getting-Started.pdf** guide.

After the Installation, the path must be set up.

Edit "`~/.bashrc`" and add the path to the Installation.

```
export PATH=<install-dir>/bin:$PATH
```

You may also want to create a file `sourcery.sh` to source for setting up the toolchain.

```
#!/bin/sh
export ARCH=arm
export GCCROOT=<install-dir>
export PATH=$GCCROOT/bin:$PATH
export CROSS_COMPILE=arm-none-linux-gnueabi-
```

Make sure it is executable:

```
chmod a+x sourcery.sh
```

And then source it.

```
. ./sourcery.sh
```

Check the setup by checking the version of the C-Compiler.

```
arm-none-linux-gnueabi-gcc --version
```

Your output should be similar to:

```
arm-none-linux-gnueabi-gcc (Sourcery CodeBench Lite 2013.11-33) 4.8.1
Copyright (C) 2013 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.
```

Verify that the first line of the output contains: Sourcery CodeBench Lite 2013.11-33.

Whenever you need to run the Sourcery CodeBench Lite toolchain, you should source the `sourcery.sh` file to set up the environment.

