Yocto Project and OpenEmbedded Training

Lab Book

Free Electrons
http://free-electrons.com

April 30, 2015

About this document

Updates to this document can be found on http://free-electrons.com/doc/training/yocto/.

This document was generated from LaTeX sources found on http://git.free-electrons.com/training-materials.

More details about our training sessions can be found on http://free-electrons.com/training.

Copying this document

© 2004-2015, Free Electrons, http://free-electrons.com.

This document is released under the terms of the Creative Commons CC BY-SA 3.0 license . This means that you are free to download, distribute and even modify it, under certain conditions.

Corrections, suggestions, contributions and translations are welcome!

Training setup

Download files and directories used in practical labs

Install lab data

For the different labs in this course, your instructor has prepared a set of data (kernel images, kernel configurations, root filesystems and more). Download and extract its tarball from a terminal:

```
cd
wget http://free-electrons.com/doc/training/yocto/yocto-labs.tar.xz
sudo tar Jvxf yocto-labs.tar.xz
sudo chown -R <user>.<user> yocto-labs
```

Note that `root` permissions are required to extract the character and block device files contained in this lab archive. This is an exception. For all the other archives that you will handle during the practical labs, you will never need `root` permissions to extract them. If there is another exception, we will let you know.

Lab data are now available in an `yocto-labs` directory in your home 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.

You are now ready to start the real practical labs!

Install extra packages

Ubuntu comes with a very limited version of the `vi` editor. Install `vim`, a improved version of this editor.

```
sudo apt-get install vim
```

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, configur-

ing 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`

 © 2004-2015 Free Electrons, CC BY-SA license

First Yocto Project build

Your first dive into Yocto Project and its build mechanism

During this lab, you will:

- Set up an OpenEmbedded environment
- Configure the project and choose a target
- Build your first Poky image

Setup

Before starting this lab, make sure your home directory is not encrypted. OpenEmbedded cannot be used on top of an eCryptFS file system due to limitations in file name lengths.

Go to the `$HOME/yocto-labs/` directory.

Install the required packages:

```
sudo apt-get install bc build-essential chrpath diffstat gawk git texinfo wget
```

Download Yocto

Download the latest stable release of the Yocto Project and extract it:

```
wget http://downloads.yoctoproject.org/releases/yocto/yocto-1.5.1/\
  poky-dora-10.0.1.tar.bz2
tar xf poky-dora-10.0.1.tar.bz2
```

Go to the Poky root directory: `cd $HOME/yocto-labs/poky-dora-10.0.1/`

Then download the OpenEmbedded TI layer:

```
git clone git://git.yoctoproject.org/meta-ti.git
cd $HOME/yocto-labs/poky-dora-10.0.1/meta-ti/
git checkout dora
```

To improve BeagleBone support, you need to apply a patch:

```
git am ~/yocto-labs/0001-beaglebone-use-the-am335x_boneblack-u-boot-configura.patch
```

Set up the build environment

Check you're using Bash. This is the default shell when using Ubuntu.

Export all needed variables and set up the build directory:

```
cd $HOME/yocto-labs/poky-dora-10.0.1/
source oe-init-build-env
```

In order to choose the target and to configure the generic build settings, edit the local configuration file (`$BUILDDIR/conf/local.conf`). Set the target machine to `beaglebone` and

update the parallelization variables (BB_NUMBER_THREADS and PARALLEL_MAKE) according to your computer capabilities.

Also, if you need to save disk space on your computer you can add INHERIT += "rm_work" in the previous configuration file. This will remove the package work directory once a package is built.

Don't forget to make the configuration aware of the TI layer. Edit the layer configuration file ($BUILDDIR/conf/bblayers.conf) and append the full path to the meta-ti directory to the BBLAYERS variable.

Finally, you must specify which machine is your target. By default it is quemu. We need to build an image for a beaglebone. Update the MACHINE configuration variable accordingly. Be careful, beaglebone is different from the beagleboard machine!

Build your first image

Now that you're ready to start the compilation, simply run:

```
bitbake core-image-minimal
```

Once the build finished, you will find the output images under $BUILDDIR/tmp/deploy/images/beaglebone.

Set up the SD card

In this first lab we will use an SD card to store the bootloader, kernel and root filesystem files. Before copying the images on it, we first need to set up the partition layout. You will find a bash script to do so, under $HOME/yocto-labs/script/.

Execute it:

```
umount /dev/mmcblk0*
sudo ./format_sdcard.sh /dev/mmcblk0
```

Once this is finished, remove the SD card, then insert it again, it should automatically be mounted.

You can now copy the two U-Boot stages, the Linux kernel image and the compiled device tree in the boot partition.

```
cp $BUILDDIR/tmp/deploy/images/beaglebone/{MLO,u-boot.img,zImage} \
  /media/$USER/boot
cp $BUILDDIR/tmp/deploy/images/beaglebone/zImage-am335x-boneblack.dtb \
  /media/$USER/boot/dtb
```

Now uncompress the generated rootfs in the second partition with the following tar command:

```
sudo tar xpf $BUILDDIR/tmp/deploy/images/beaglebone/\
  core-image-minimal-beaglebone.tar.gz -C /media/$USER/rootfs
sync
```

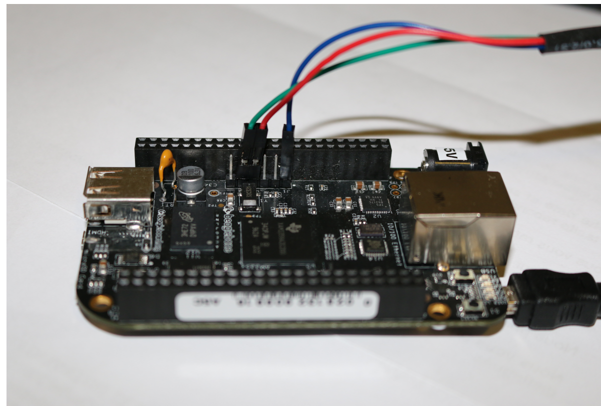
Setting up serial communication with the board

The Beaglebone serial connector is exported on the 6 pins close to one of the 48 pins headers. Using your special USB to Serial adapter provided by your instructor, connect the ground wire

 © 2004-2015 Free Electrons, CC BY-SA license

(blue) to the pin closest to the power supply connector (let's call it pin 1), and the TX (red) and RX (green) wires to the pins 4 (board RX) and 5 (board TX). [1]

You always should make sure that you connect the TX pin of the cable to the RX pin of the board, and vice-versa, whatever the board and cables that you use.



Once the USB to Serial connector is plugged in, a new serial port should appear: /dev/ttyUSB0. You can also see this device appear by looking at the output of dmesg.

To communicate with the board through the serial port, install a serial communication program, such as picocom:

```
sudo apt-get install picocom
```

If you run `ls -l /dev/ttyUSB0`, you can also see that only root and users belonging to the dialout group have read and write access to this file. Therefore, you need to add your user to the dialout group:

```
sudo adduser $USER dialout
```

You now need to log out and log in again to make the new group visible everywhere.

Now, you can 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].

There should be nothing on the serial line so far, as the board is not powered up yet.

Configure the U-Boot environment and boot

Insert the SD card in the dedicated slot on the BeagleBone Black. Press the S2 push button (located just above the previous slot), plug in the USB cable and release the push button. You should see boot messages on the console.

Stop the boot process when you see `Hit any key to stop autoboot` to access the U-Boot command line.

In order to boot properly, you need to set up some configuration variables to tell the bootloader how to load the Linux kernel. In the U-Boot command-line, set the following variables:

```
setenv bootcmd 'mmc rescan; fatload mmc 0 0x80200000 zImage; fatload mmc 0
0x82000000 dtb; bootz 0x80200000 - 0x82000000'
```

[1] See https://www.olimex.com/Products/Components/Cables/USB-Serial-Cable/USB-Serial-Cable-F/ for details about the USB to Serial adapter that we are using.

```
setenv bootargs 'console=ttyO0 earlyprintk root=/dev/mmcblk0p2 rw'
saveenv
```

Finally you can start the kernel using the U-Boot command `boot`. Wait until the login prompt, then enter `root` as user. Congratulations! The board has booted and you now have a shell.

 © 2004-2015 Free Electrons, CC BY-SA license

Advanced Yocto configuration

Configure the build, customize the output images and use NFS

During this lab, you will:

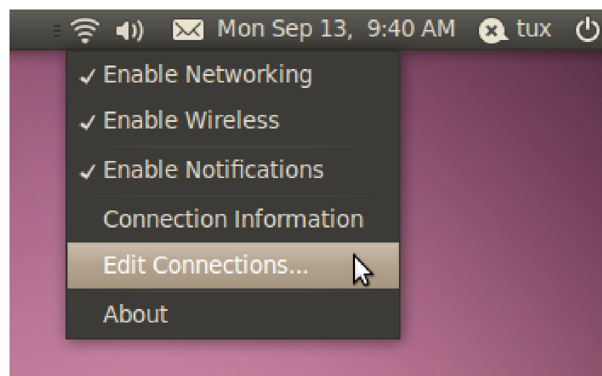
- Customize the package selection
- Configure the build system
- Use the rootfs over NFS

Set up the Ethernet communication

Later on, we will mount our root filesystem through the network using NFS. This works on top of an Ethernet connection.

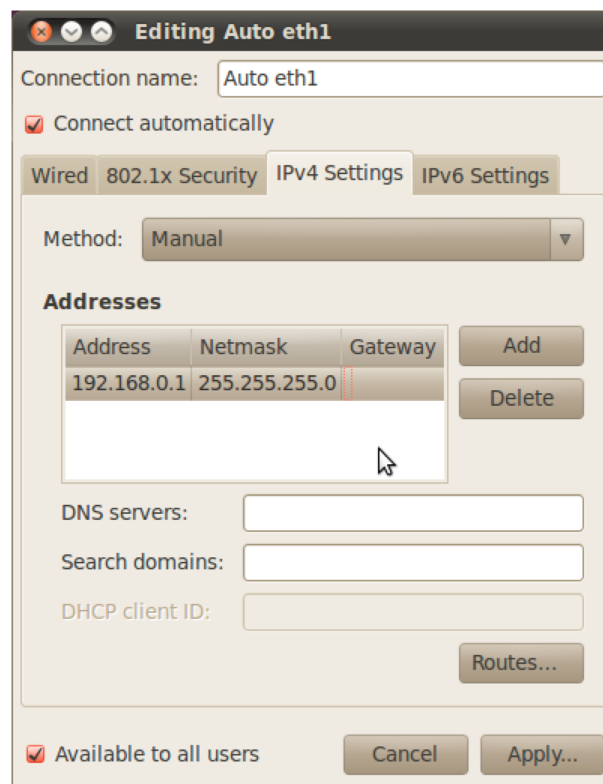
With a network cable, connect the Ethernet port of your board to the one of your computer. If your computer already has a wired connection to the network, your instructor will provide you with a USB Ethernet adapter. A new network interface, probably `eth1` or `eth2`, should appear on your Linux system.

To configure this network interface on the workstation side, click on the *Network Manager* tasklet on your desktop, and select *Edit Connections*.



Select the new *wired network connection*:

In the `IPv4 Settings` tab, press the `Add` button and make the interface use a static IP address, like `192.168.0.1` (of course, make sure that this address belongs to a separate network segment from the one of the main company network).



You can use `255.255.255.0` as `Netmask`, and leave the `Gateway` field untouched (if you click on the `Gateway` box, you will have to type a valid IP address, otherwise you won't be apply to click on the `Apply` button).

Now, configure the network on the board in U-Boot by setting the `ipaddr` and `serverip` environment variables:

```
setenv ipaddr 192.168.0.100
setenv serverip 192.168.0.1
```

The first time you use your board, you also need to send the MAC address in U-boot:

 © 2004-2015 Free Electrons, CC BY-SA license

```
setenv ethaddr 12:34:56:ab:cd:ef
```

In case the board was previously configured in a different way, we also turn off automatic booting after commands that can be used to copy a kernel to RAM:

```
setenv autostart no
```

To make these settings permanent, save the environment:

```
saveenv
```

Now switch your board off and on again[2].

Set up the NFS server

First install the NFS server on the training computer and create the root NFS directory:

```
sudo apt-get install nfs-kernel-server
sudo mkdir -m 777 /nfs
```

Then make sure this directory is used and exported by the NFS server by adding `/nfs *(rw,sync,no_root_squash,subtree_check)` to the `/etc/exports` file.

Finally restart the service:

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

Tell the kernel to use the rootfs over NFS

Then set the kernel boot arguments U-Boot will pass to the Linux kernel at boot time:

```
setenv bootargs 'console=ttyO0 root=/dev/nfs rw nfsroot=192.168.0.1:/nfs
  ip=192.168.0.100'
saveenv
```

If you later want to make changes to this setting, you can use:

```
editenv bootargs
```

Add a package to the rootfs image

You can add packages to be built by editing the local configuration file `$BUILDDIR/conf/local.conf`. The `IMAGE_INSTALL` variable controls the packages included into the output image.

To illustrate this, add the Dropbear SSH server to the list of enabled packages.

Tip: do not override the default enabled package list, but append the Dropbear package instead.

Choose a package variant

Dependencies of a given package are explicitly defined in its recipe. Some packages may need a specific library or piece of software but others only depend on a functionality. As an example, the kernel dependency is described by `virtual/kernel`.

[2] Power cycling your board is needed to make your `ethaddr` permanent, for obscure reasons. If you don't, U-boot will complain that `ethaddr` is not set.

To see which kernel is used, dry-run BitBake:

```
bitbake -vn virtual/kernel
```

In our case, we can see the `linux-ti-staging` provides the `virtual/kernel` functionality:

```
NOTE: selecting linux-ti-staging to satisfy virtual/kernel due to PREFERRED_PROVIDERS
```

We can force Yocto to select another `kernel` by explicitly defining which one to use in our local configuration. Try switching from `linux-ti-staging` to `linux-dummy` only using the local configuration.

Then check the previous step worked by dry-running again BitBake.

```
bitbake -vn virtual/kernel
```

You can now rebuild the whole Yocto project, with `bitbake core-image-minimal`

Tip: you need to define the more specific information here to be sure it is the one used. The `MACHINE` variable can help here.

As this was only to show how to select a preferred provider for a given package, you can now use `linux-ti-staging` again.

Boot with the updated rootfs

First we need to put the rootfs under the NFS root directory so that it is accessible by NFS clients. Simply uncompress the archived output image in the previously created `/nfs` directory:

```
tar xpf $BUILDDIR/tmp/deploy/images/beaglebone/\
  core-image-minimal-beaglebone.tar.gz -C /nfs
```

Then boot the board.

The Dropbear SSH server was enabled a few steps before, and should now be running as a service on the BeagleBone Black. You can test it by accessing the board through SSH:

```
ssh root@192.168.0.100
```

You should see the BeagleBone Black command line!

Going further: BitBake tips

BitBake is a powerful tool which can be used to execute specific commands. Here is a list of some useful ones, used with the `virtual/kernel` package.

- The Yocto recipes are divided into numerous tasks, you can print them by using: `bitbake -c listtasks virtual/kernel`.
- BitBake allows to call a specific task only (and its dependencies) with: `bitbake -c <task> virtual/kernel`. (`<task>` can be `menuconfig` here).
- You can force to rebuild a package by calling: `bitbake -f virtual/kernel`
- `world` is a special keyword for all packages. `bitbake -c fetchall world` will download all packages sources (and their dependencies).
- You can get a list of locally available packages and their current version with: `bitbake -s`

 © 2004-2015 Free Electrons, CC BY-SA license

- You can also find detailed information on available packages, their current version, dependencies or the contact information of the maintainer by visiting:
  http://packages.yoctoproject.org/

For detailed information, please run `bitbake -h`

© 2004-2015 Free Electrons, CC BY-SA license

Add a custom application

Add a new recipe to support a required custom application

During this lab, you will:

- Write a recipe for a custom application
- Integrate this application into the build

This is the first step of adding an application to Yocto. The remaining part is covered in the next lab, "Create a Yocto layer".

Setup and organization

In this lab we will add a recipe handling the `nInvaders` application. Before starting the recipe itself, find the `recipes-extended` directory and add a subdirectory for your application.

A recipe for an application is usually divided into a version specific `bb` file and a common one. Try to follow this logic and separate the configuration variables accordingly.

Tip: it is possible to include a file into a recipe with the keyword `require`.

First hands on nInvaders

The nInvaders application is a terminal based game following the space invaders family. In order to deal with the text based user interface, nInvaders uses the ncurses library.

First try to find the project homepage, download the sources and have a first look: license, Makefile, requirements...

Write the common recipe

Create an appropriate common file, ending in `.inc`

In this file add the common configuration variables: source URI, package description...

Write the version specific recipe

Create a file that respects the Yocto nomenclature: `${PN}_${PV}.bb`

Add the required common configuration variables: archive checksum, license file checksum, package revision...

Testing and troubleshooting

You can check the whole packaging process is working fine by explicitly running the build task on the `nInvaders` recipe:

```
bitbake ninvaders
```

 © 2004-2015 Free Electrons, CC BY-SA license

Try to make the recipe on your own. Also eliminate the warnings related to your package: some configuration variables are not mandatory but it is a very good practice to define them all.

If you hang on a problem, check the following points:

- The common recipe is included in the version specific one
- The checksum and the URI are valid
- The dependencies are explicitly defined
- The internal state has changed, clean the working directory:
  `bitbake -c cleanall ninvaders`

Tip: BitBake has command line flags to increase its verbosity and activate debug outputs.

Update the rootfs and test

Now that you've compiled the `nInvaders` package, generate a new rootfs image with `bitbake core-image-minimal`. Then update the NFS root directory. You can confirm the `nInvaders` program is present by running:

```
find /nfs -iname ninvaders
```

Access the board command line through SSH. You should be able to launch the `nInvaders` program. It's time to play!

© 2004-2015 Free Electrons, CC BY-SA license

Create a Yocto layer

Add a custom layer to the Yocto project for your project needs

During this lab, you will:

- Create a new Yocto layer
- Interface this custom layer to the existing Yocto project
- Use applications from custom layers

This lab extends the previous one, in order to fully understand how to interface a custom project to the basic Yocto project.

Tools

You can access the configuration and state of layers with the `bitbake-layers` command. This command can also be used to retrieve useful information about available recipes. Try the following commands:

```
bitbake-layers show-layers
bitbake-layers show-recipes linux-ti-staging
bitbake-layers show-overlayed
```

Another helpful exported script is `yocto-layer`. You can read its dedicated help page by using the `help` argument. Also read the help page related to the `create` argument.

Create a new layer

With the above commands, create a new Yocto layer named `meta-felabs` with a priority of `7`.

Before using the new layer, we need to configure its generated configuration files. You can start with the `README` file which is not used in the build process but contains information related to layer maintenance. You can then check, and adapt if needed, the global layer configuration file located in the `conf` directory of your custom layer.

Tips: the `yocto-layer` command creates a layer in the current directory unless otherwise stated. Also be careful, the `meta-` keyword is mandatory.

Integrate a layer to the build

To be fair, we already used and integrated a layer in our build configuration during the first lab, with `meta-ti`. This layer was responsible for BeagleBone Black support in Yocto. We have to do the same for our `meta-felabs` now.

There is a file which contains all the paths of the layers we use. Try to find it without looking back to the first lab. Then add the full path to our newly created layer to the list of removable layers (i.e. those which aren't part of the true Yocto core).

Validate the integration of the `meta-felabs` layer with:

 © 2004-2015 Free Electrons, CC BY-SA license

```
bitbake-layers show-layers
```

Add a recipe to the layer

In the previous lab we introduced a recipe for the nInvaders game. We included it to the existing `meta` layer. While this approach give a working result, the Yocto logic is not respected. You should instead always use a custom layer to add recipes or to customize the existing ones. To illustrate this we will move our previously created nInvaders recipe into the `meta-felabs` layer.

You can check the nInvaders recipe is part of the `meta` layer first:

```
bitbake-layers show-recipes ninvaders
```

Then move the nInvaders recipe to the layer created below. You can check that the nInvaders recipe is now part of the `meta-felabs` layer with the `bitbake-layers` command.

Extend a recipe

Add your features to an existing recipe

During this lab, you will:

- Apply patches to an existing recipe
- Use a custom configuration file for an existing recipe
- Extend a recipe to fit your needs

Create a basic appended recipe

To avoid rewriting recipes when a modification is needed on an already existing package, BitBake allows to extend recipes and to overwrite, append or prepend configuration variables values through the so-called BitBake append files.

We will first create a basic BitBake append file, without any change made to the original recipe, to see how it is integrated into the build. We will then extend some configuration variables of the original recipe.

Try to create an appended recipe with the help of the online Yocto Project development documentation. You can find it at http://www.yoctoproject.org/docs/1.4.2/dev-manual/dev-manual.html. We here aim to extend the `linux-ti-staging` kernel recipe.

You can see available `bbappend` files and the recipe they apply to by using the `bitbake-layers` tool (again!):

```
bitbake-layers show-appends
```

If the BitBake append file you just created is recognized by your Yocto environment, you should see:

```
linux-ti-staging_3.12.bb:
  $POKY/meta-felabs/recipes-kernel/linux/linux-ti-staging_3.12.bbappend
```

Add patches to apply in the recipe

We want our extended `linux-ti-staging` kernel to support the Nunchuk as a joystick input. We can add this by applying a patch during the `do_configure` task. The needed patches are provided with this lab. You can find them under `~/yocto-labs/nunchuck/linux`. For more details about how to write the driver handling the Nunchuk, have a look on our embedded Linux kernel and driver development training course at http://free-electrons.com/training/kernel/.

Applying a patch is a common task in the daily Yocto process. Many recipes, appended or not, apply a specific patch on top of a mainline project. It's why patches do not have to be explicitly applied, if the recipe inherits from the patch class (directly or not), but only have to be present in the source files list.

Try adding the patches included in this lab to your BitBake append file. Do not forget to also add the `defconfig` file provided alongside the patches.

 © 2004-2015 Free Electrons, CC BY-SA license

You can now rebuild the `linux-ti-staging` kernel to take the new patches into account:

```
bitbake virtual/kernel
```

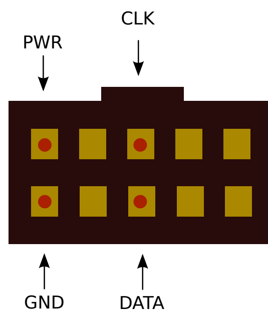
This method is the common one when surcharging recipes.

Connect the nunchuk

Take the nunchuk device provided by your instructor.

We will connect it to the second I2C port of the CPU (`i2c1`), with pins available on the `P9` connector.

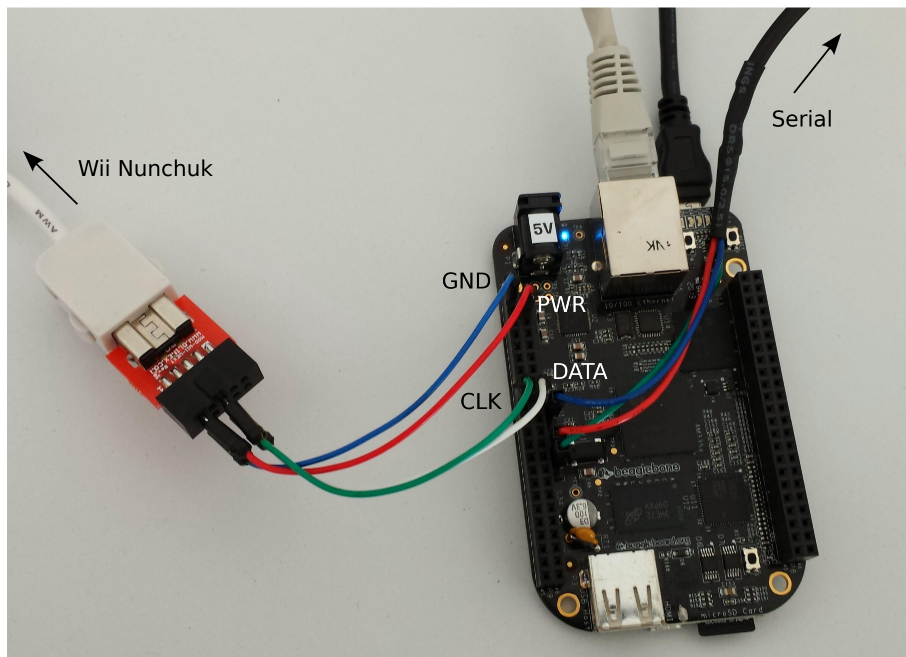
Identify the 4 pins of the nunchuk connector:



Nunchuk i2c pinout
(UEXT connector from Olimex)

Connect the nunchuk pins:

- The `GND` pin to P9 pins 1 or 2 (`GND`)
- The `PWR` pin to P9 pins 3 or 4 (`DC_3.3V`)
- The `CLK` pin to P9 pin 17 (`I2C1_SCL`)
- The `DATA` pin to P9 pin 18 (`I2C1_SDA`)



Test the Nunchuk

Copy the newly generated kernel and device tree images into the first SD card partition. Then boot the board and wait until you have access to the `busybox` command line.

You can then make sure that the Nunchuk is recognized and is working by checking the presence of the `js0` device file:

```
ls /dev/input/js0
```

Now display the raw events generated by the Nunchuk:

```
cat /dev/input/js0
```

You should see random characters appearing while playing with the Nunchuk. Be aware that the driver we integrated also handles accelerometer events. Therefore, moving the device will produce many events!

Patch nInvaders

The nInvaders game uses keyboard events for its controls. We first need to apply a patch introducing joystick support. The patch is located at `~/yocto-labs/nunchuck/ninvaders/`.

Add the patch to the nInvaders `SRC_URI` and do not forget you need to specify where it is located.

Then build a full `core-image-minimal` and update the NFS root directory.

Play nInvaders!

After booting the board you should be able to play nInvaders with the keyboard... and the Nunchuk! The `C` button is used to confirm and to fire, and `Z` to pause the game.

Access the board command line througth SSH, and launch the game:

 © 2004-2015 Free Electrons, CC BY-SA license

```
$ ninvanders
```

Going further: use the Yocto Kernel metadata

When dealing with some kernel recipes, it can be possible to use a more advanced way to add patches and to manage the configuration. This can be achieved with `linux-yocto` packages. These packages can be found under `meta/recipes-kernel/linux/` and provide the official generic Yocto kernel. They can be useful as a starting point to create a custom kernel without doing everything from scratch. You can also create a very modular recipe by doing so.

Read the Yocto Project's kernel documentation at http://www.yoctoproject.org/docs/current/kernel-dev/kernel-dev.html and find how to append a custom feature to an existing kernel recipe for an explicit architecture. Then write a BitBake append file extending the `linux-yocto_3.10` recipe.

© 2004-2015 Free Electrons, CC BY-SA license

Create a custom machine configuration

Let Poky know about your hardware!

During this lab, you will:

- Create a custom machine configuration
- Understand how the target architecture is dynamically chosen

Create a custom machine

The machine file configures various hardware related settings. As early as in lab1, we chose the `beaglebone` one. While it is not necessary to make our custom machine image here, we'll create a new one to demonstrate the process.

Add a new `felabs` machine to the previously created layer. In order to properly boot on the BeagleBone Black board. Since the `meta-felabs` layer is already created, do not use the `yocto-bsp` tool.

This machine describes a board using the `cortexa8thf-neon` tune and is a part of the `ti33x` SoC family. Add the following lines to your machine configuration file:

```
SOC_FAMILY = "ti33x"
require conf/machine/include/soc-family.inc

DEFAULTTUNE ?= "cortexa8thf-neon"
require conf/machine/include/tune-cortexa8.inc

UBOOT_ARCH = "arm"
UBOOT_MACHINE = "am335x_evm_config"
UBOOT_ENTRYPOINT = "0x80008000"
UBOOT_LOADADDRESS = "0x80008000"
```

Populate the machine configuration

This `felabs` machine needs:

- To select `linux-ti-staging` as the preferred provider for the kernel.
- To use `am335x-boneblack.dtb` device tree.
- To select `u-boot-ti-staging` as the preferred provider for the bootloader.
- To use a `zImage` kernel image type.
- To configure the serial console to `115200 ttyO0`
- And to support some features:
  - kernel26

 © 2004-2015 Free Electrons, CC BY-SA license

- apm
- usbgadget
- usbhost
- vfat
- ext2
- ethernet

Build an image with the new machine

You can now update the MACHINE variable value in the local configuration and start a fresh build.

Have a look on the generated files

Once the generated images supporting the new felabs machine are generated, you can check all the needed images were generated correctly.

Have a look in the output directory, in $BUILDDIR/tmp/deploy/images/felabs/. Is there something missing?

Update the rootfs

You can now update your root filesystem, to use the newly generated image supporting our felabs machine!

© 2004-2015 Free Electrons, CC BY-SA license

Create a custom image

The highest level of customization in Poky

During this lab, you will:

- Write a full customized image recipe
- Choose the exact packages you want on your board

Add a basic image recipe

A build is configured by two top level recipes: the machine recipe and the image one. The image recipe is the top configuration file for the generated rootfs and the packages it includes. Our aim is this lab is to define a custom image from scratch to allow a precise selection of packages on the BeagleBone Black. To show how to deal with real world configuration and how the Yocto Project can be used in the industry we will, in addition to the production image recipe you will use in the final product, create a development one including debug tools and show how to link the two of them to avoid configuration duplication.

First add a custom image recipe in the `meta-felabs` layer. We will name it `felabs-image-minimal`. You can find information on how to create a custom image on the dedicated Yocto Project development manual at http://www.yoctoproject.org/docs/1.5.1/dev-manual/dev-manual.html. There are different ways to customize an image, we here want to create a full recipe, using a custom `.bb` file.

Do not forget to inherit from the `core-image` class.

Select the images capabilities and packages

You can control the packages built and included into the final image with the `IMAGE_INSTALL` configuration variable. It is a list of packages to be built. You can also use package groups to include a bunch of programs, generally enabling a functionality, such as `packagegroup-core-boot` which adds the minimal set of packages required to boot an image (i.e. a shell or a kernel).

You can find the package groups under the `packagegroups` directories. To have a list of the available one:

```
find -name packagegroups
```

Open some of them to read their description and have an idea about the capabilities they provide. Then update the installed packages of the image recipe and don't forget to add the nInvaders one!

Add a custom package group

We just saw it is possible to use package groups to organize and select the packages instead of having a big blob of configuration in the image recipe itself. We will here create a custom package for game related recipes.

 © 2004-2015 Free Electrons, CC BY-SA license

With the above documentation, create a `packagegroup-felabs-games` group which inherits from the `packagegroup` class. Add the nInvaders program into its runtime dependencies.

Now update the image recipe to include the package group instead of the nInvaders program directly.

Differentiate the production recipe from the debug one

You can enable the debugging capabilities of your image just by changing the BitBake target when building the whole system. We want here to have a common base for both the production and the debug images, but also take in account the possible differences. In our example only the built package list will change.

Create a debug version of the previous image recipe, and name it `felabs-image-minimal-dev`. Try to avoid duplicating code! Then add the `dev-pkgs` to the image features list. It is also recommended to update the recipe's description.

Build the new debug image with BitBake and check the previously included packages are present in the newly generated rootfs.

Develop your application in the Poky SDK

Generate and use the Poky SDK

During this lab, you will:

- Build the Poky SDK
- Install the SDK
- Compile an application for the BeagleBone in the SDK environment

Build the SDK

Two SDKs are available, one only embedding a toolchain and the other one allowing for application development. We will use the latter one here.

First, build an SDK for the `felabs-image-minimal` image, with the `populate_sdk` task.

Once the SDK is generated, a script will be available at `tmp/deploy/sdk`.

Install the SDK

Open a new console to be sure that no extra environment variable is set. We mean to show you how the SDK sets up a fully working environment.

Install the SDK in `$HOME/yocto-labs/sdk` by executing the script generated at the previous step.

```
$BUILDDIR/tmp/deploy/sdk/poky-eglibc-x86_64-felabs-image-minimal-cortexa8hf-vfp-neon-toolchain-1.5.1.sh
```

Set up the environment

Go into the directory where you installed the SDK (`$HOME/yocto-labs/sdk`). Source the environment script:

```
source environment-setup-cortexa8hf-vfp-neon-poky-linux-gnueabi
```

Have a look on the exported environment variables:

```
env
```

Compile an application in the SDK

Download the essential `Ctris` sources at http://www.hackl.dhs.org/data/download/download.php?file=ctris-0.42.tar.bz2

Extract the source in the SDK:

 © 2004-2015 Free Electrons, CC BY-SA license

```
tar xf ctris-0.42.tar.bz2
cd ctris-0.42
```

Then modify the Makefile, to make sure that the environment variables exported by the SDK script are not overridden.

Compile the application. You can check the application was successfully compiled for the right target by using the file command. The `ctris` binary should be an ELF 32-bit LSB executable compiled for ARM.

Finally, you can copy the binary to the board, by using the `scp` command. Then run it and play a bit to ensure it is working fine!

Use the Yocto Project SDK through Eclipse

Build and use the Yocto Project SDK with Eclipse

During this lab, you will:

- Integrate the Eclipse Yocto Project plugin
- Configure the plugin to work with the previously used Yocto project
- Develop and modify Poky from Eclipse

Set up the environment

First we need to set up the environment in order to be able to develop our applications. We need Poky to build support for Eclipse (an IDE). Run:

```
bitbake meta-ide-support
```

Donwload Eclipse

Download the Kepler version of Eclipse on the official website: `http://www.eclipse.org/downloads/packages/eclipse-standard-432/keplersr2`. Then uncompress the tarball and launch Eclipse. Ubuntu has a known bug, and you need to run the following command from the extracted Eclipse directory to be able to use it properly:

```
UBUNTU_MENUPROXY=0 ./eclipse
```

Install the Eclipse plugin

First, you need to download a few packages to fulfill the Yocto Eclipse plugin requirements. Open `Install New Software` in the `Help` menu. Select the `Kepler - http://download.eclipse.org/releases/kepler` repository and install:

- LTTng - Linux Tracing Toolkit
- C/C++ Remote Launch
- Remote System Explorer End-user Runtime
- Remote System Explorer User Actions
- Target Management Terminal
- TCF Remote System Explorer add-in
- TCF Target Explorer

You can then either choose to download the already built Eclipse Yocto Project plugin or to build your own by first downloading its source repository. We will here download the latest plugin available on the Yocto Project website directly from Eclipse.

 © 2004-2015 Free Electrons, CC BY-SA license

First add the Yocto Project Eclipse Update site to the available software sites: http://downloads.yoctoproject.org/releases/eclipse-plugin/1.5.1/kepler. Then download:

- Yocto Project ADT Plug-in
- Yocto Project BitBake Commander Plug-in
- Yocto Project Documentation plug-in

Finally, you need to set up the plugin itself. Open `Preferences` from the "Window" menu. Then click on `Yocto Project ADT` in the preference dialog. Choose `Build System Derived Toolchain`, set the `Toolchain Root Location` to your build directory and the `Sysroot Location` to the `tmp/sysroot/beaglebone` directory of your build directory. Then select the `Target Architecture`. Apply and close, the plugin is set up!

Build a Poky image from Eclipse

The Eclipse Yocto Project plugin allows, in addition to compiling an application to the right target architecture, to build a Poky image. We will here configure Eclipse to build our exact previous BeagleBone image, but it is also possible to create a build environment from scratch.

Select `New` in the `File` menu. Then double click on `New Yocto Project` in the `Yocto Project Bitbake Commander` category. In the new window, choose `Local` as the `Remote service provider` and `local` as the connection name. Uncheck the `Clone from Yocto Project Git Repository into new location`, this option is only used when starting a vanilla project. The directory used by Eclipse will be `Location/Project name`, you need to fill the form with this in mind: use the path of the directory containing your previously used `poky` root directory and put the name of this directory as the project name. Then click on `Finish`. The new project is now visible in the `Project Explorer` panel.

In order to build a target image, you need to launch Hob, the graphical interface for BitBake. You can find this tool under the `Project` menu. In the Hob window, select the right target machine. You will then be able to select our previously created image recipe. Have a look at the `Advanced configuration` menu, and then start a build.

Once the build completed, you will see useful information about the image built in the Hob window.

Create a recipe

The Yocto Project BitBake Commander Plug-in allows to fully manage the Yocto Project from Eclipse, including creating a new recipe with a user friendly wizard. To demonstrate this ability, we will create a new recipe for the wonderful Steam Locomotive command (`sl`). The home page is located at: `https://github.com/mtoyoda/sl`.

Select the `File New Other` menu. Double click on the `BitBake Recipe` under the `Yocto Project BitBake Commander` category. Now you can create the recipe by filling the form according to the previous labs we followed.

© 2004-2015 Free Electrons, CC BY-SA license