Pengwyn Documentation

Release A

Architech

Mar 16, 2017

Contents

Version 2.0.0A

Copyright (C)2016 Avnet Silica company

Date 13/02/2015

You can find the previous documentation: Here

Welcome to **Pengwyn** documentation!
Have you just received your Pengwyn board? Then you sure want to read the *Unboxing* Chapter first.

If you are a new user of the **Yocto based SDK** we suggest you to read the *Quick start guide* chapter, otherwise, if you want to have a better understanding of specific topics, just jump directly to the chapter that interests you the most.

Furthermore, we encourage you to read the official Yocto Project documentation.

CHAPTER 1

Notations

Throughout this guide, there are commands, file system paths, etc., that can either refer to the machine (real or virtual) you use to run the SDK or to the board.

Host

This box will be used to refer to the machine running the SDK

Board

This box will be used to refer to Pengwyn board

However, the previous notations can make you struggle with long lines. In such a case, the following notation is used.

If you click on *select* on the top right corner of these two last boxes, you will get the text inside the box selected. We have to warn you that your browser might select the line numbers as well, so, the first time you use such a feature, you are invited to check it.

Sometimes, when referring to file system paths, the path starts with **/path/to**. In such a case, the documentation is **NOT** referring to a physical file system path, it just means you need to read the path, understand what it means, and understand what is the proper path on your system. For example, when referring to the device file associated to your USB flash memory you could read something like this in the documentation:

Since things are different from one machine to another, you need to understand its meaning and corresponding value for your machine, like for example:

When referring to a specific partition of a device, you could read something like this in the documentation:

Even in this case, the things are different from one machine to another, like for example:

we are referring to the device /dev/sdb and in the specific to the partition 1. To know more details please refer to *device files* section of the *appendix*.

CHAPTER 2

Chapters

Unboxing

This powerful board comes with this beautiful box

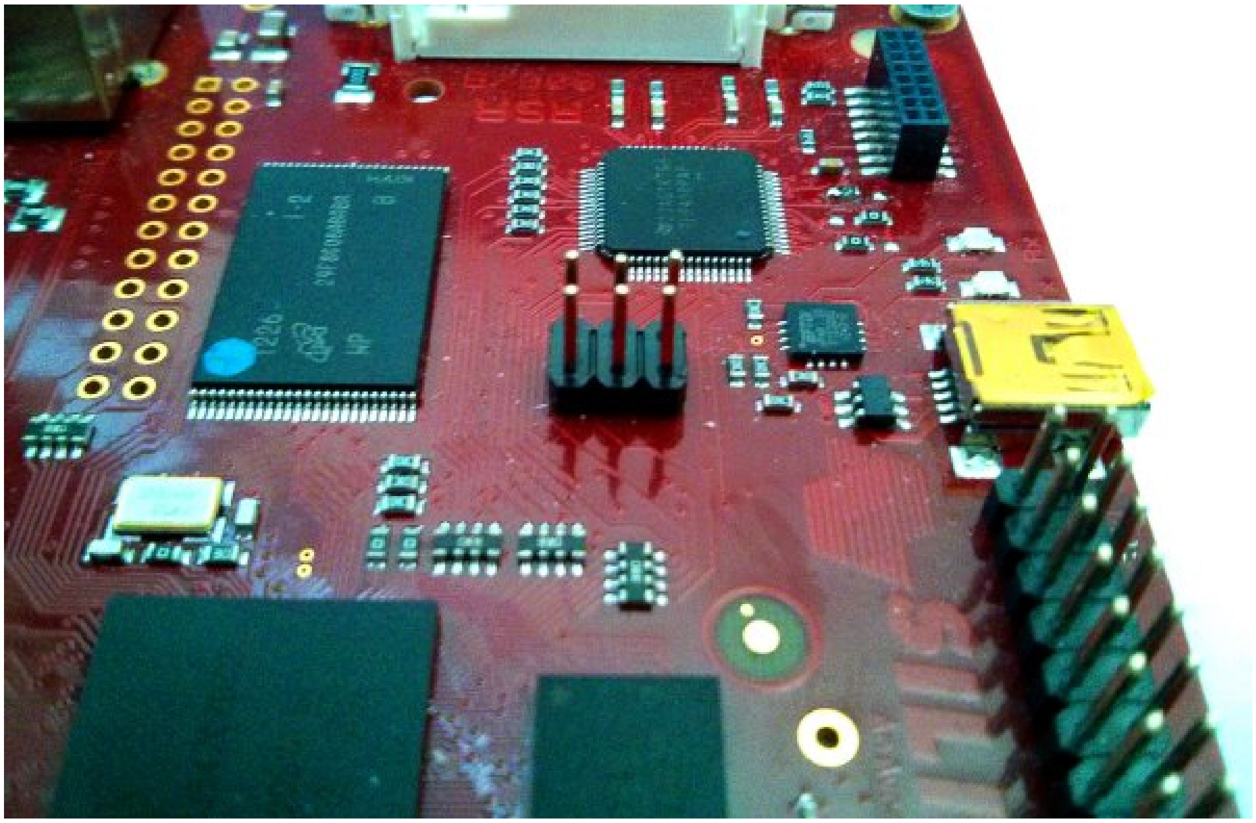
Pengwyn takes the power from the mini-USB connector **CN6** and/or connector **CN1**. The board is not shipped with an external power adapter.

SILICA
Silica PENGWYN Board
Quickstart guide
you for buying Silica PENGWYN development board.
kstart:
visit http://www.silica.com/pengwyn to download firmware, sources and docs

The NAND on the board has been programmed to let Pengwyn boot a *qt4e-demo-image-pengwyn* image generated with Yocto.

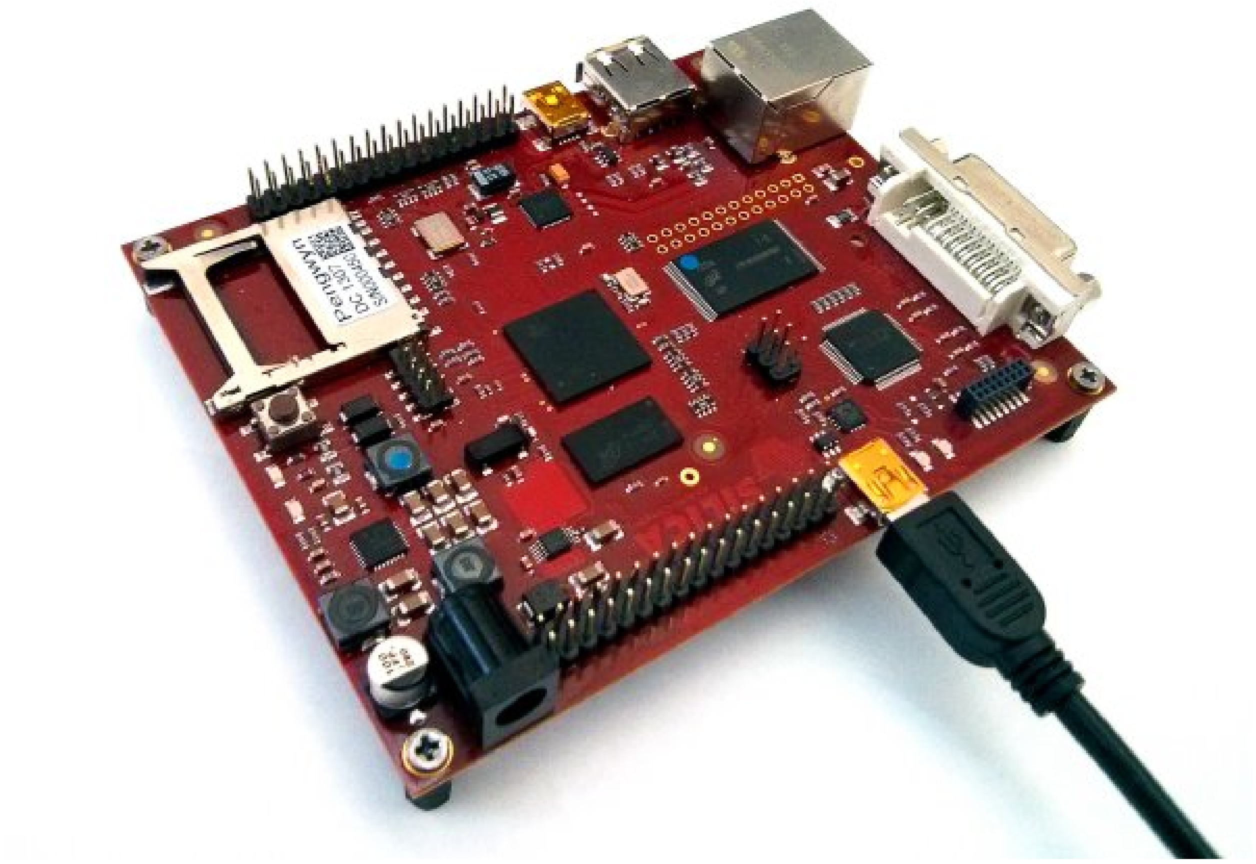
What are we waiting for? Lets boot the board!

1. First of all, make sure the board can boot entirely from the NAND by setting **J1**, **J2** and **J3** opened:



2. Connect the DVI-D connector **CN11** to your monitor/television by means of an DVI-D cable
3. Shall we power on the board for the first time? Of course!

On Pengwyn you can use the same USB cable used to power up the board to get access to the serial console.
The serial console connector **CN6**



which you can connect, by means of a mini-USB cable, to your personal computer.

Note: Every operating system has its own killer application to give you a serial terminal interface. In this guide, we are assuming your **host** operating system is **Ubuntu**.

On a Linux (Ubuntu) host machine, the console is seen as a *ttyUSB***X** device (where X is a number) and you can access to it by means of an application like *minicom*.

Minicom needs to know the name of the serial device. The simplest way for you to discover the name of the device is by looking to the kernel messages, so:

1. clean the kernel messages
2. connect the mini-USB cable to the board already powered-on
3. display the kernel messages
3. read the output

As you can see, here the device has been recognized as */dev/ttyUSB0*.

Now that you know the device name, run minicom:

If minicom is not installed, you can install it with:

then you can setup your port with these parameters:

If on your system the device has not been recognized as */dev/ttyUSB0*, just replace */dev/ttyUSB0* with the proper device.

Once you are done configuring the serial port, you are back to minicom main menu and you can select *exit*.

Give *root* to the login prompt:

Board

pengwyn login: root

and press *Enter*.

Note: Sometimes, the time you spend setting up minicom makes you miss all the output that leads to the login and you see just a black screen, press enter then to get the login prompt.

Enjoy!

Quick start guide

This document will guide you from importing the virtual machine to debugging an *Hello World!* example on a customized Linux distribution you will generate with **OpenEmbedded/Yocto** toolchain.

Install

The development environment is provided as a virtual disk (to be used by a VirtualBox virtual machine) which you can download from this page:

Important: http://downloads.architechboards.com/sdk/virtual_machine/download.html

Important: Compute the MD5SUM value of the zip file you downloaded and compare it to the golden one you find in the download page.

Uncompress the file, and you will get a *.vdi* file that is our virtual disk image. The environment contains the SDK for all the boards provided by Architech, Pengwyn included.

Download VirtualBox

For being able to use it, you first need to install **VirtualBox** (version 4.2.10 or higher). You can get VirtualBox installer from here:

https://www.virtualbox.org/wiki/Downloads

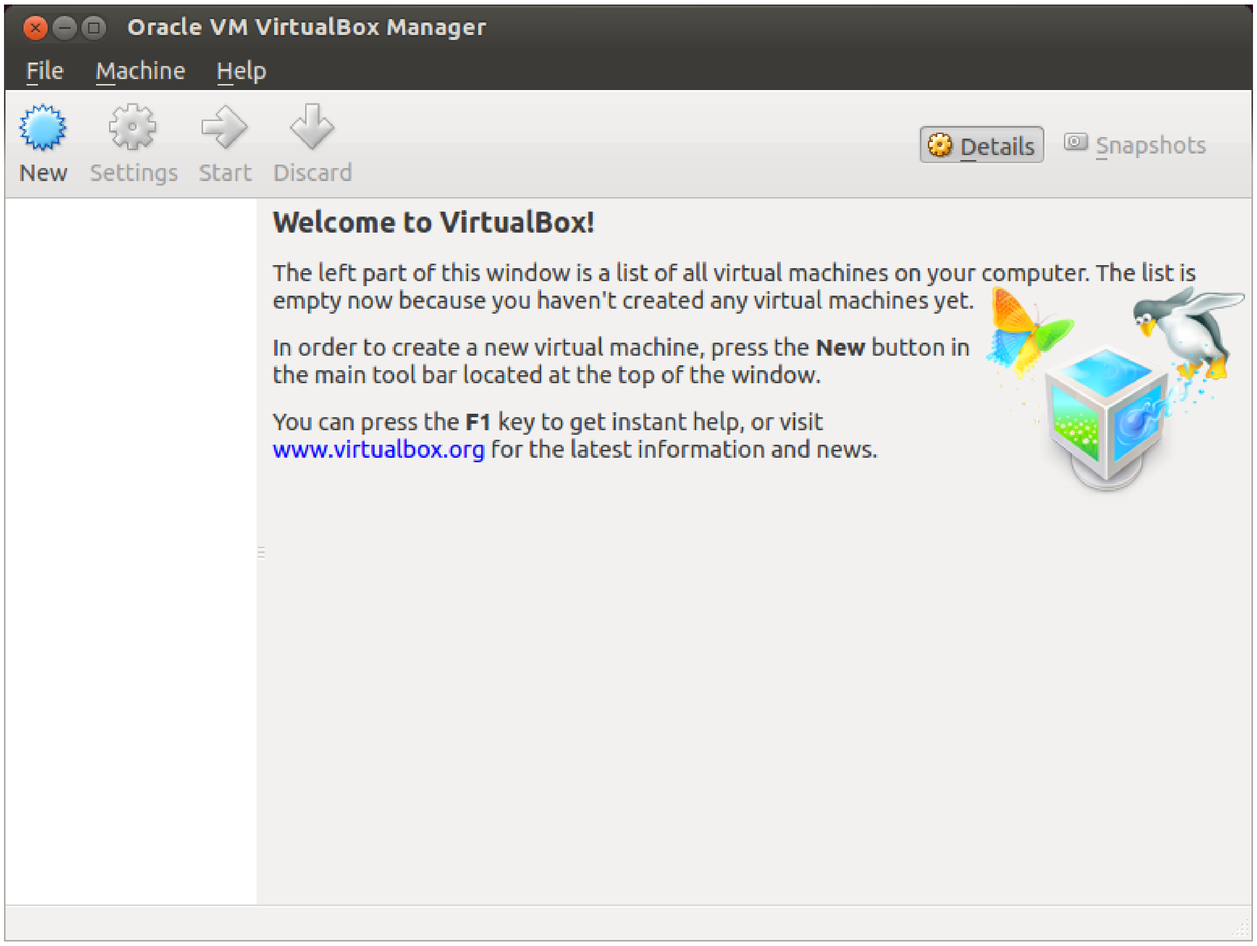
Download the version that suits your host operating system. You need to download and install the **Extension Pack** as well.

Important: Make sure that the extension pack has the same version of VirtualBox.

Install the software with all the default options.

Create a new Virtual Machine

1. Run VirtualBox



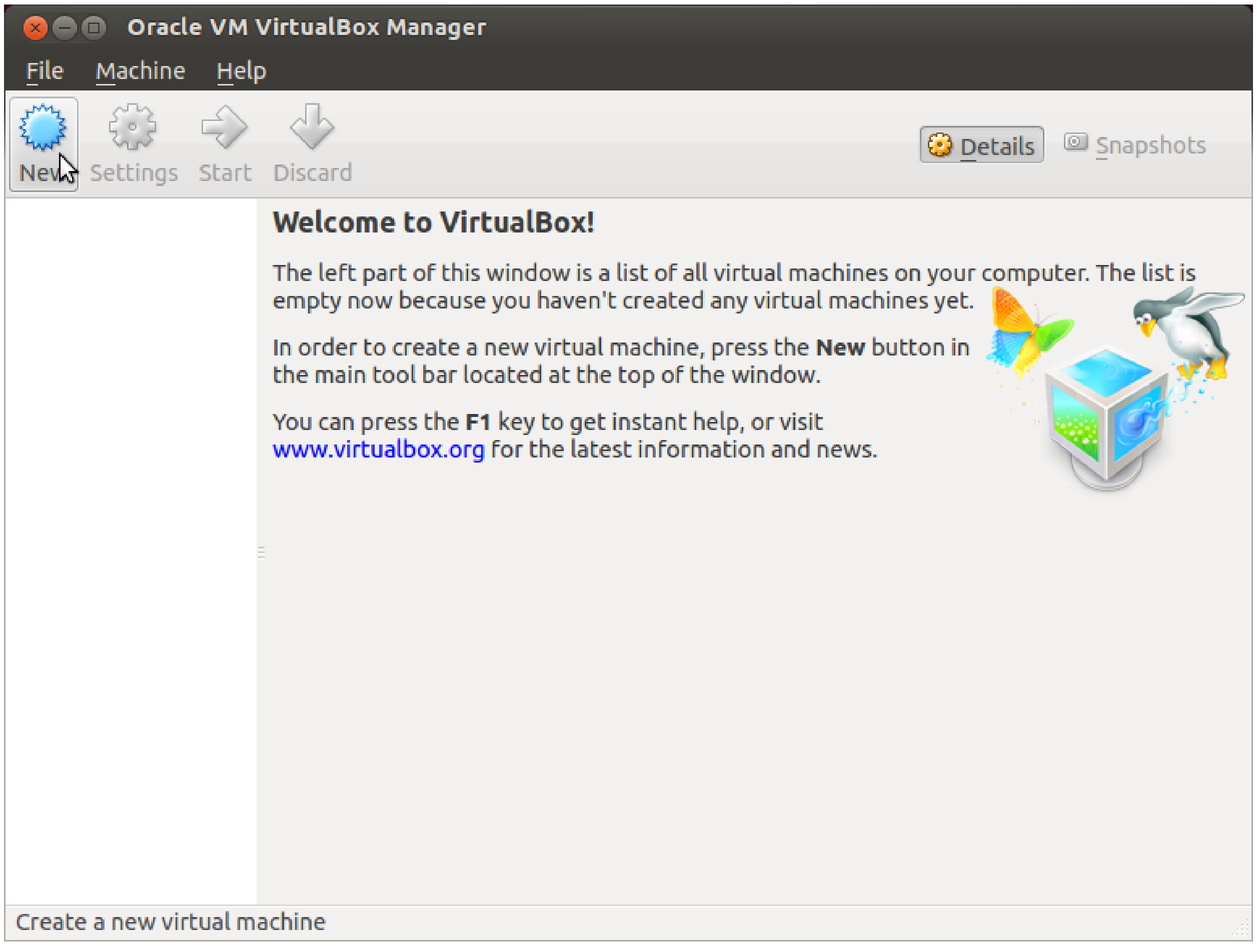
Welcome to VirtualBox!

The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for the latest information and news.

2. Click on *New* button



Welcome to VirtualBox!

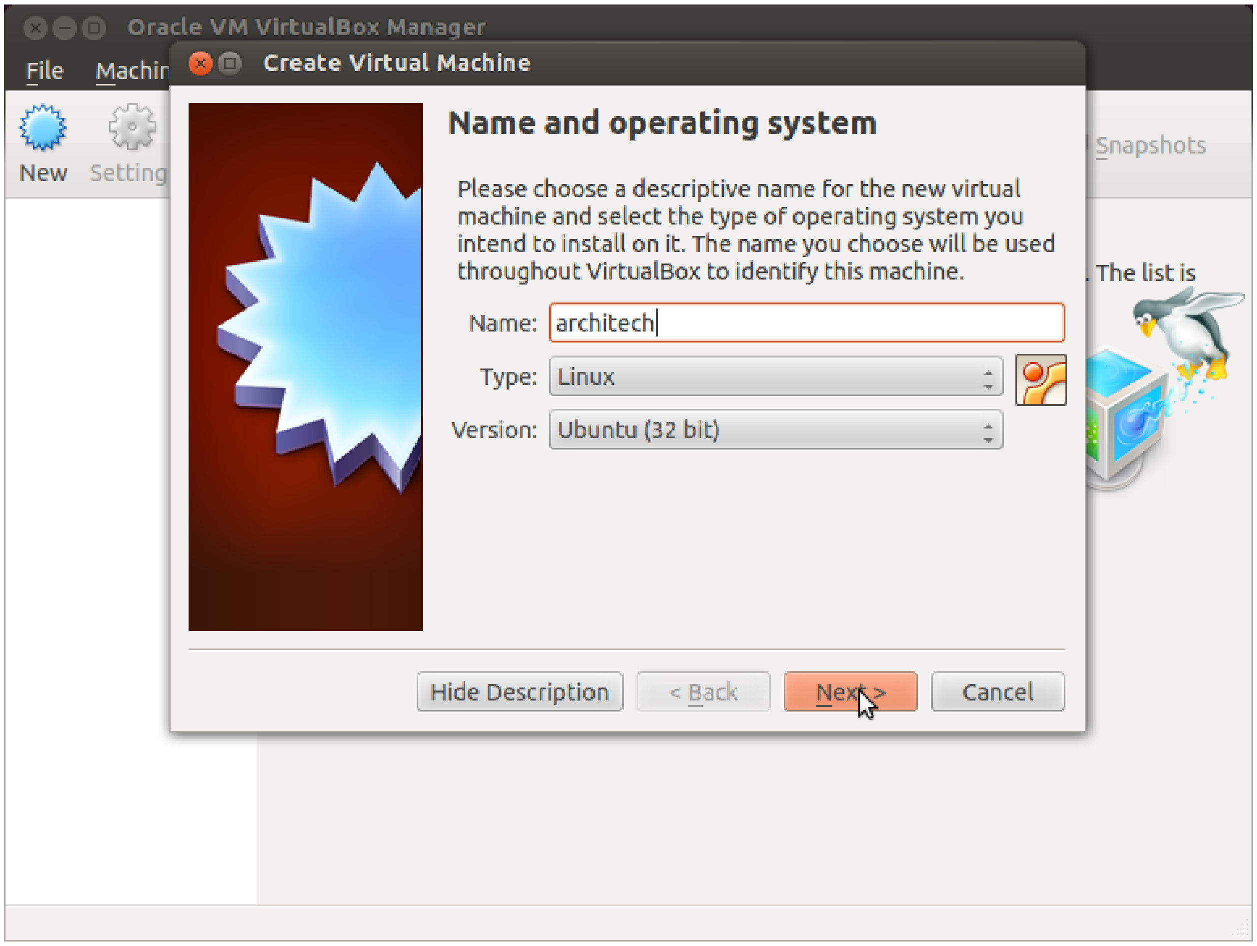
The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for the latest information and news.

Create a new virtual machine

3. Select the name of the virtual machine and the operating system type



Name and operating system

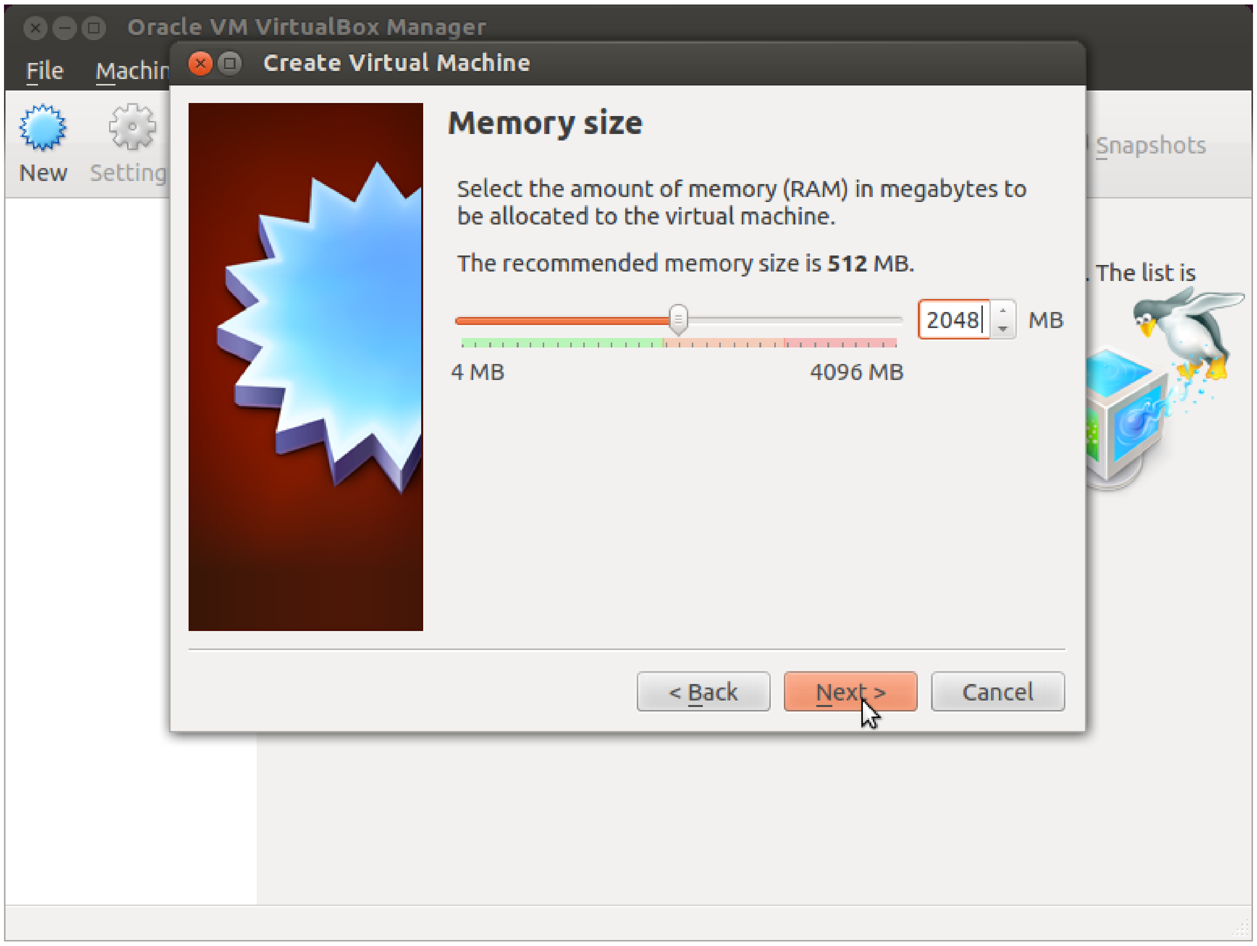
Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: architech

Type: Linux

Version: Ubuntu (32 bit)

4. Select the amount of memory you want to give to your new virtual machine



Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

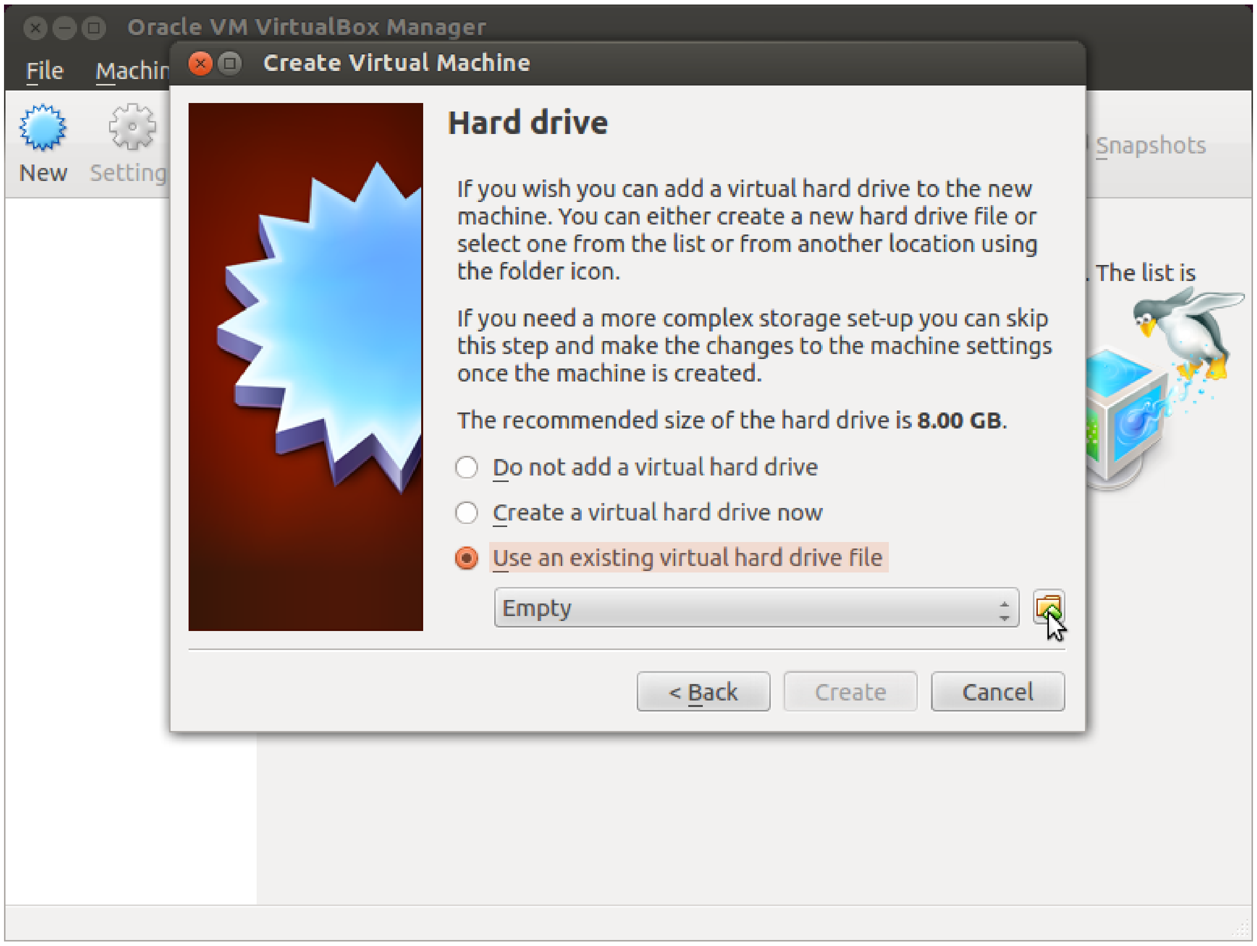
The recommended memory size is **512** MB.

2048 MB

4 MB 4096 MB

. The list is

5. Make the virtual machine use Architech's virtual disk by pointing to the downloaded file. Than click on *Create*.



Hard drive

If you wish you can add a virtual hard drive to the new machine. You can either create a new hard drive file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard drive is **8.00 GB**.

- Do not add a virtual hard drive
- Create a virtual hard drive now
- [✓ Use an existing virtual hard drive file]

Empty

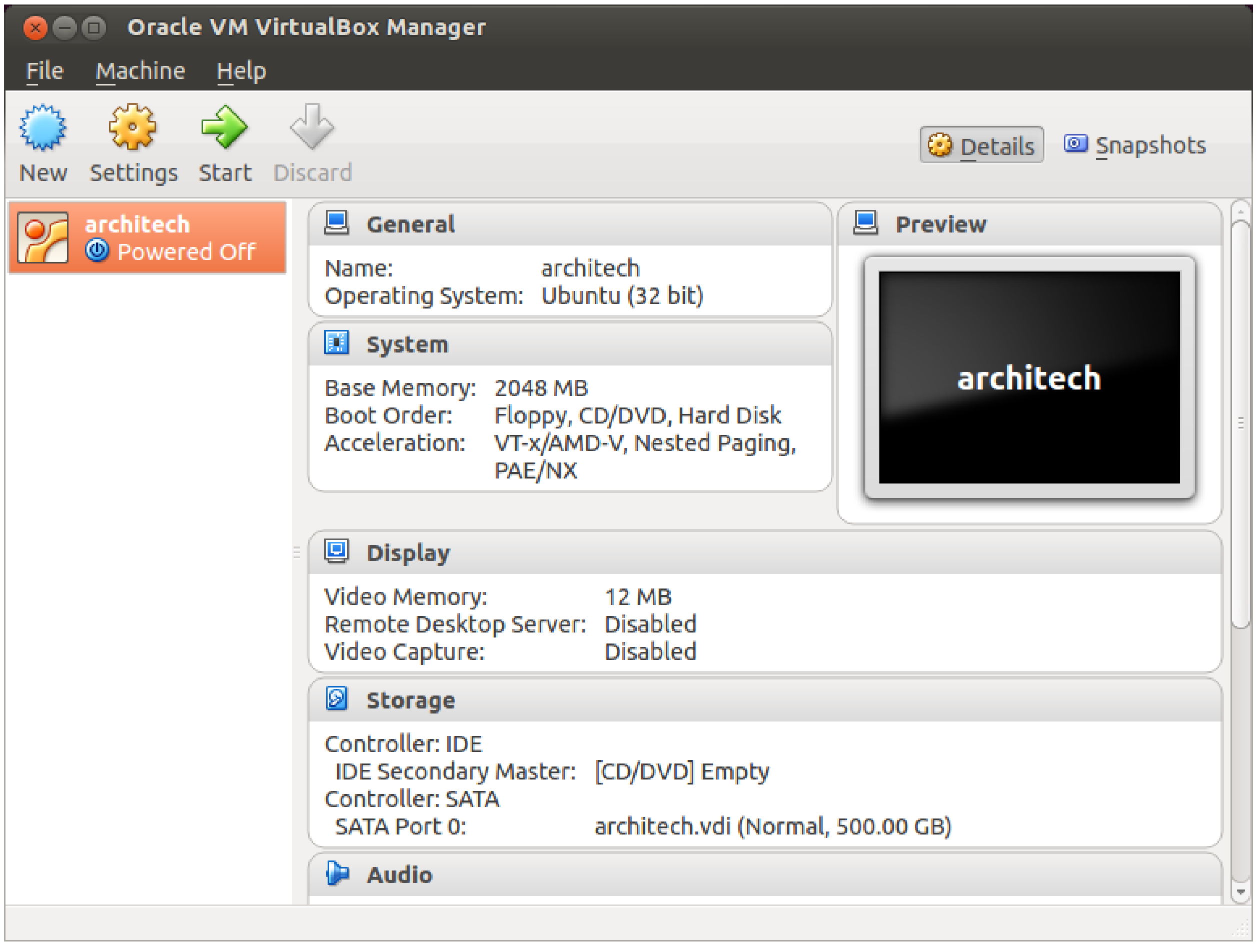
< Back | Create | Cancel

Setup the network

We need to setup a port forwarding rule to let you (later) use the virtual machine as a local repository of packages.

Note: The virtual machine must be off

1. Select Architech's virtual machine from the list of virtual machines



General

Name: architech
Operating System: Ubuntu (32 bit)

System

Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Display

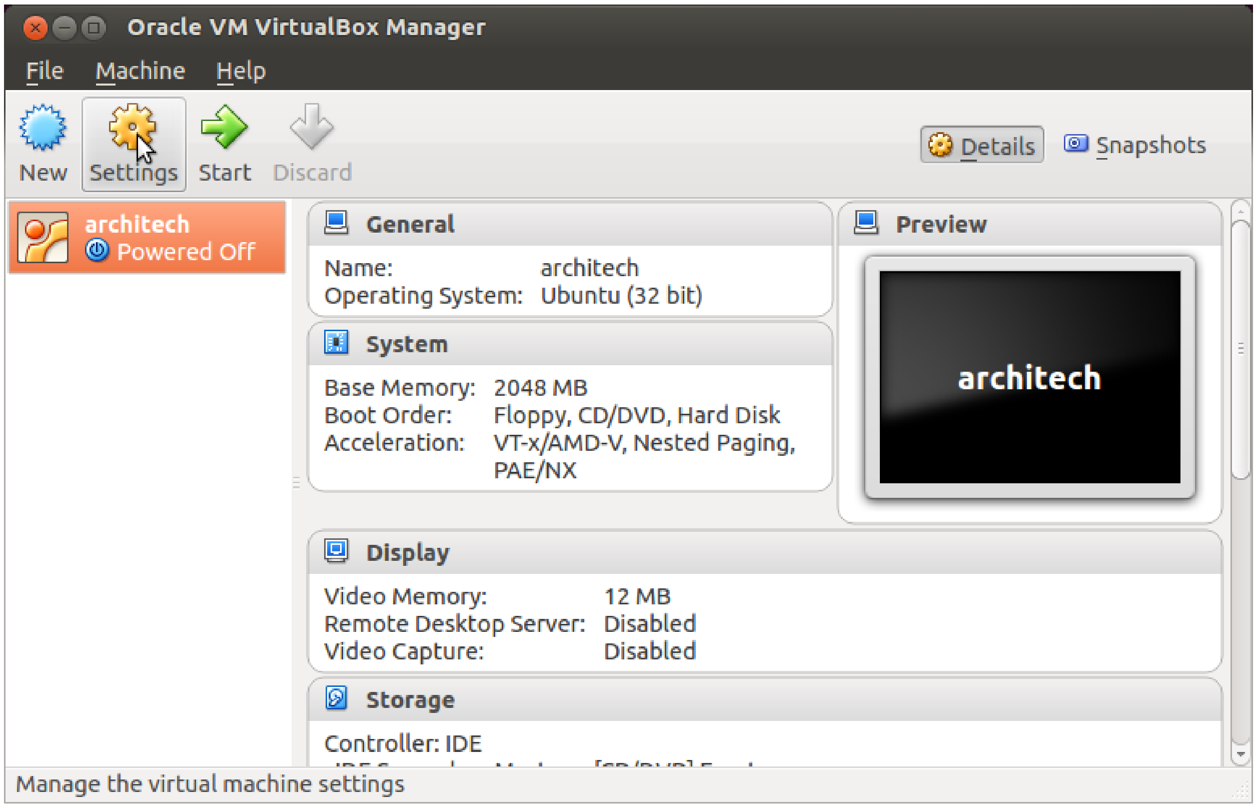
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: IDE
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: architech.vdi (Normal, 500.00 GB)

Audio

2. Click on *Settings*



General

Name: architech
Operating System: Ubuntu (32 bit)

System

Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Display

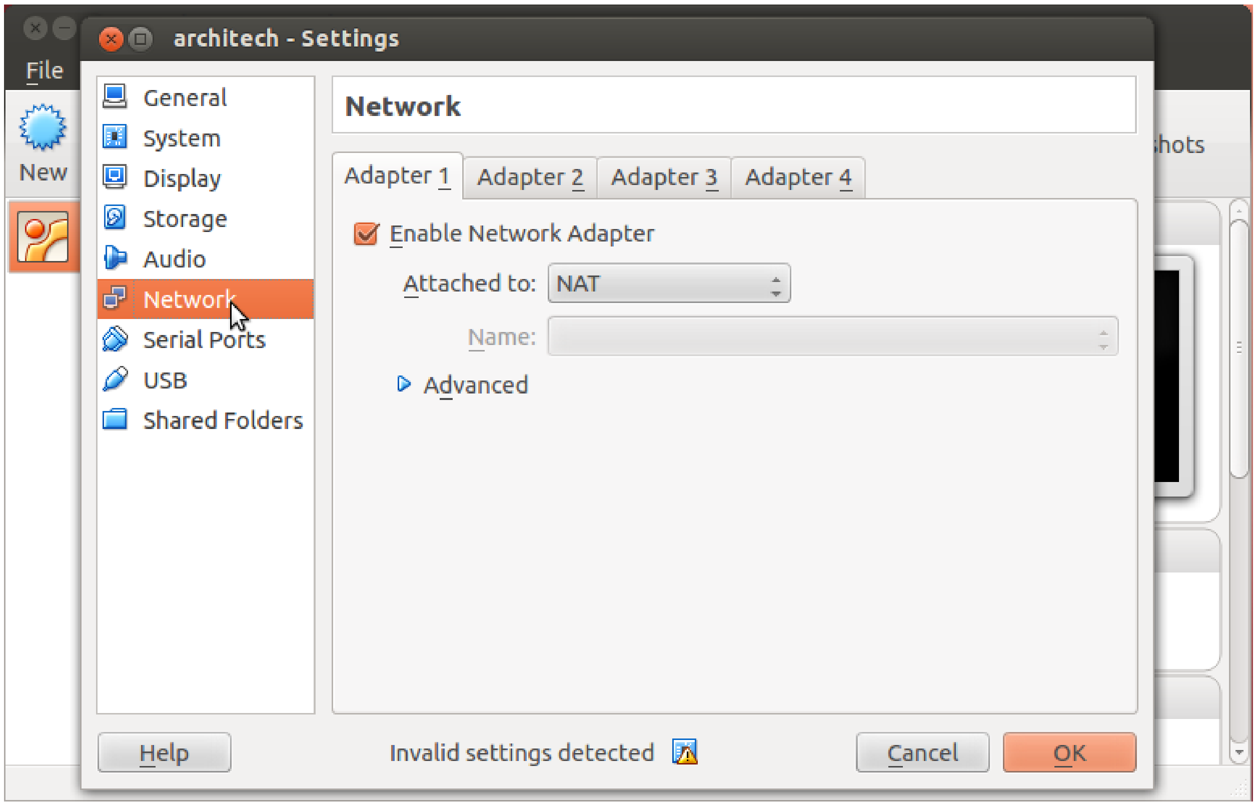
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: IDE

Manage the virtual machine settings

3. Select *Network*



Network

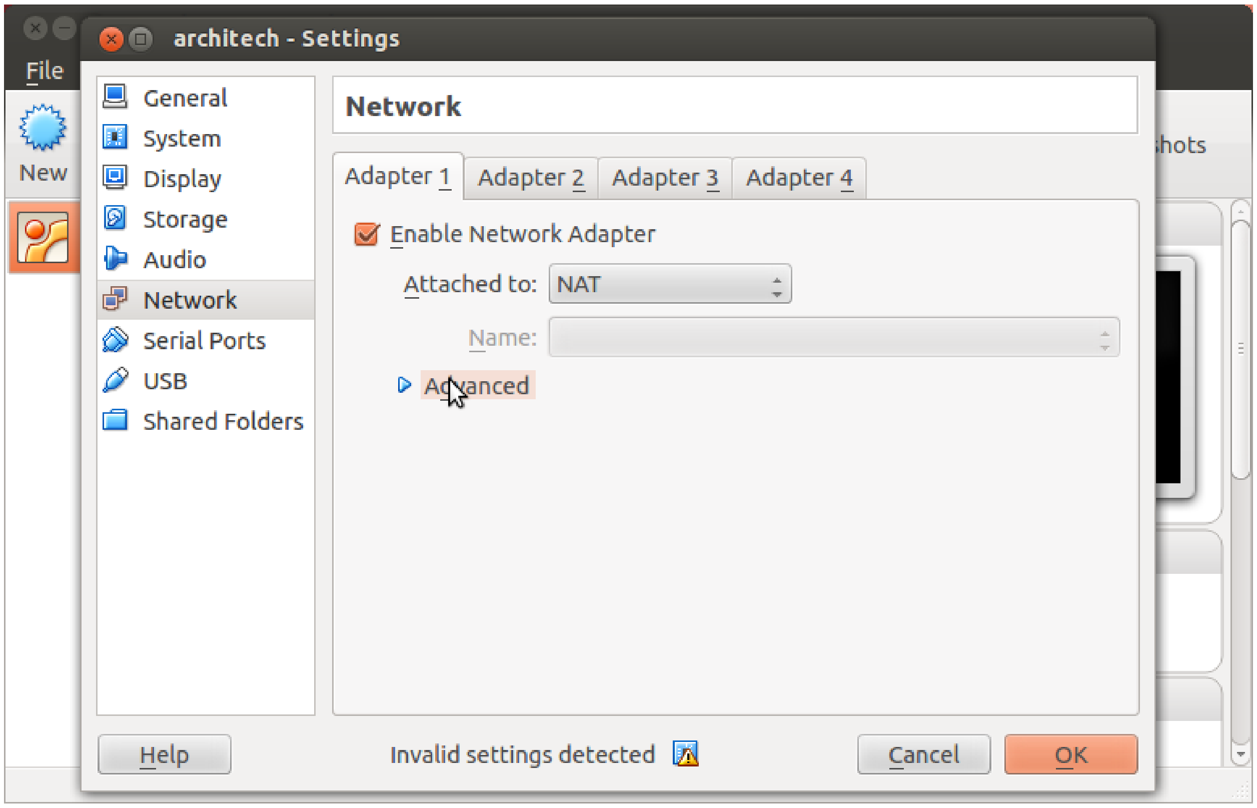
☑ Enable Network Adapter

Attached to: NAT

Name:

Invalid settings detected

4. Expand *Advanced* of *Adapter 1*



Network

Enable Network Adapter

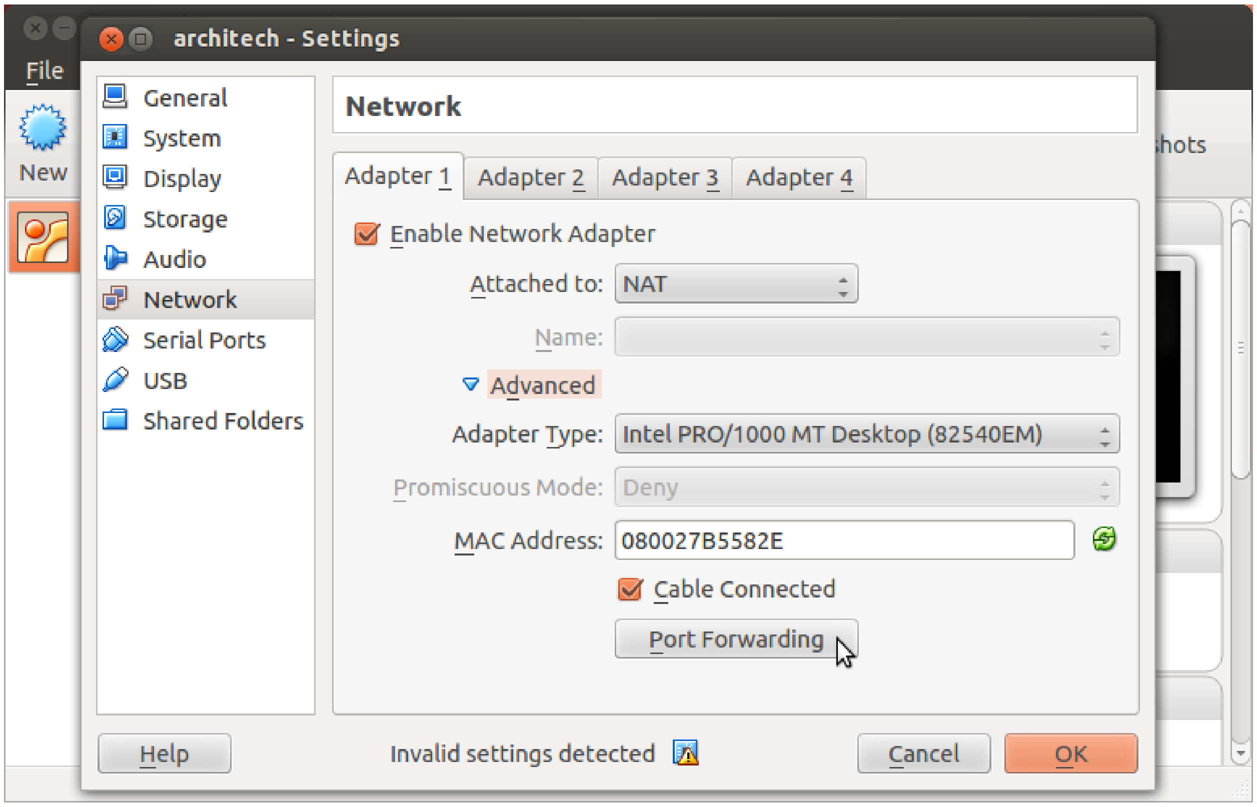
Attached to: NAT

Name:

Advanced

Invalid settings detected

5. Click on *Port Forwarding*



Network

Enable Network Adapter

Attached to: NAT

Name:

Advanced

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

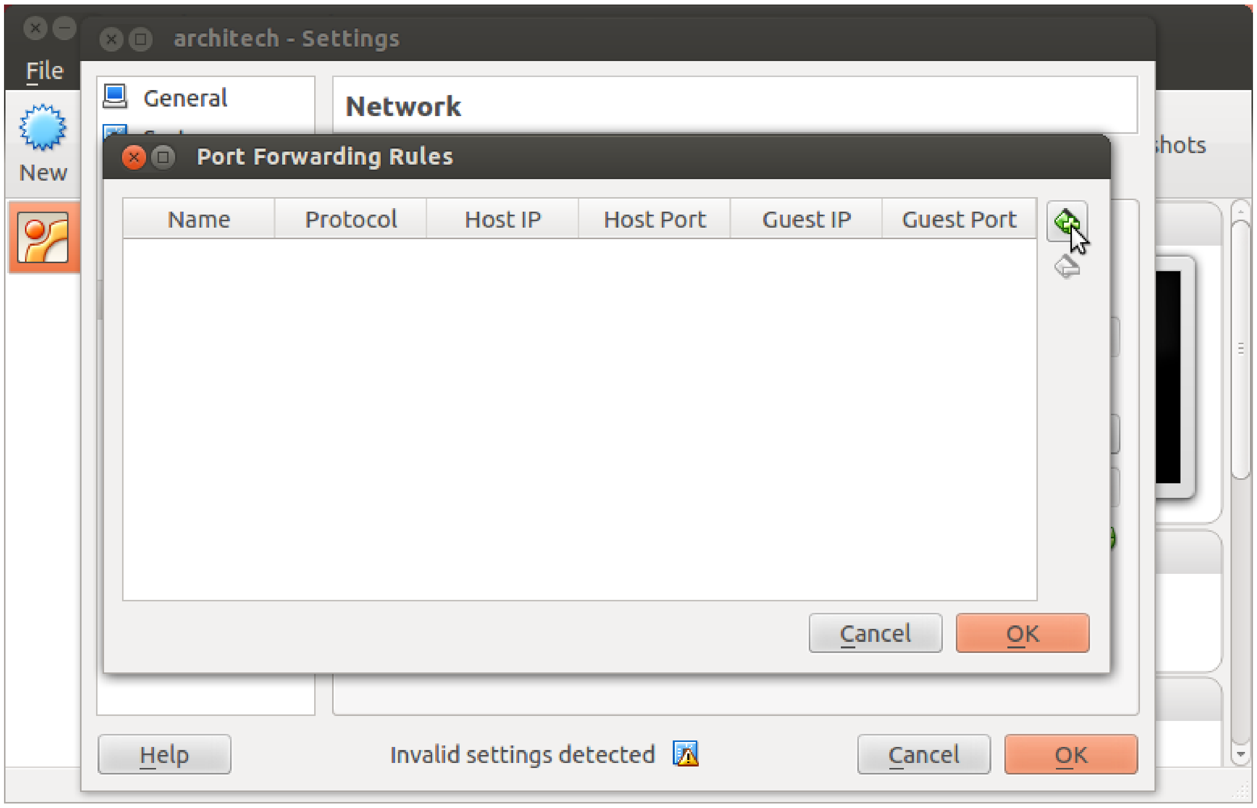
Promiscuous Mode: Deny

MAC Address: 080027B5582E

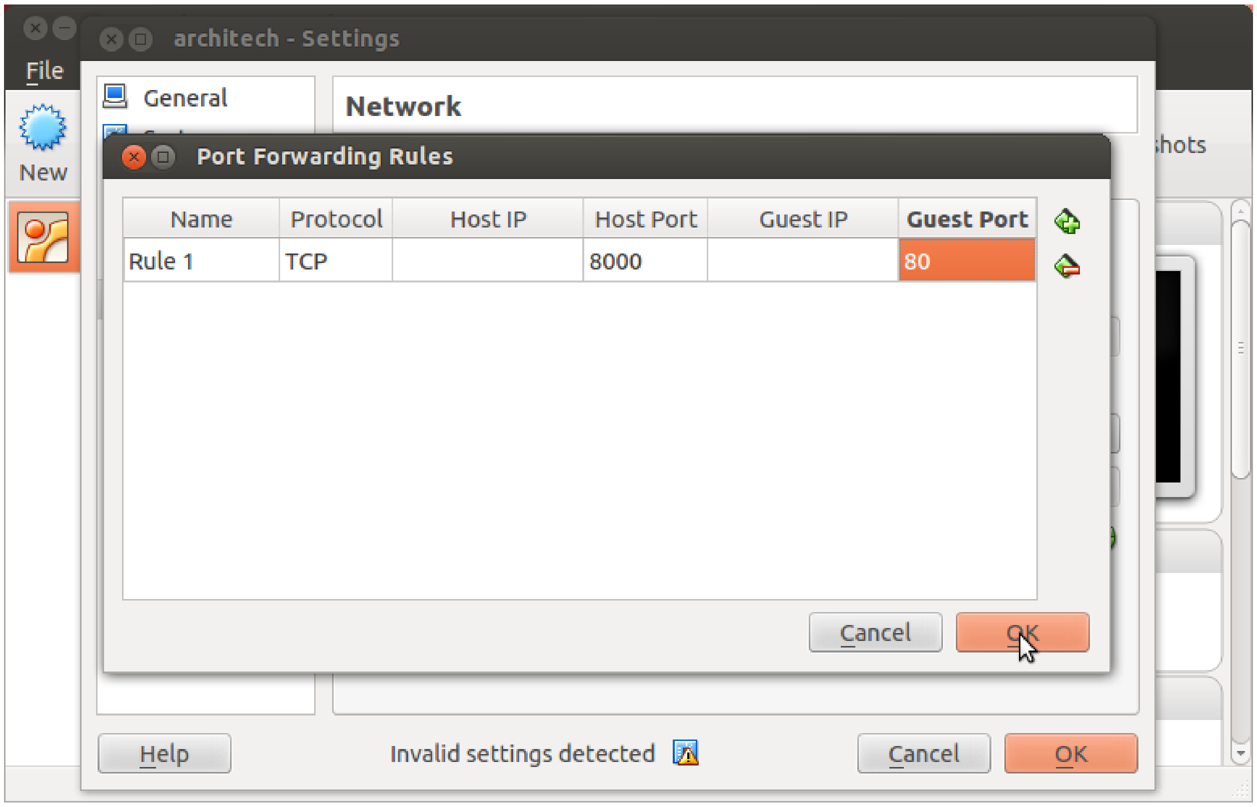
Cable Connected

Invalid settings detected

6. Add a new *rule*



7. Configure the *rule*



Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Rule 1	TCP		8000		80

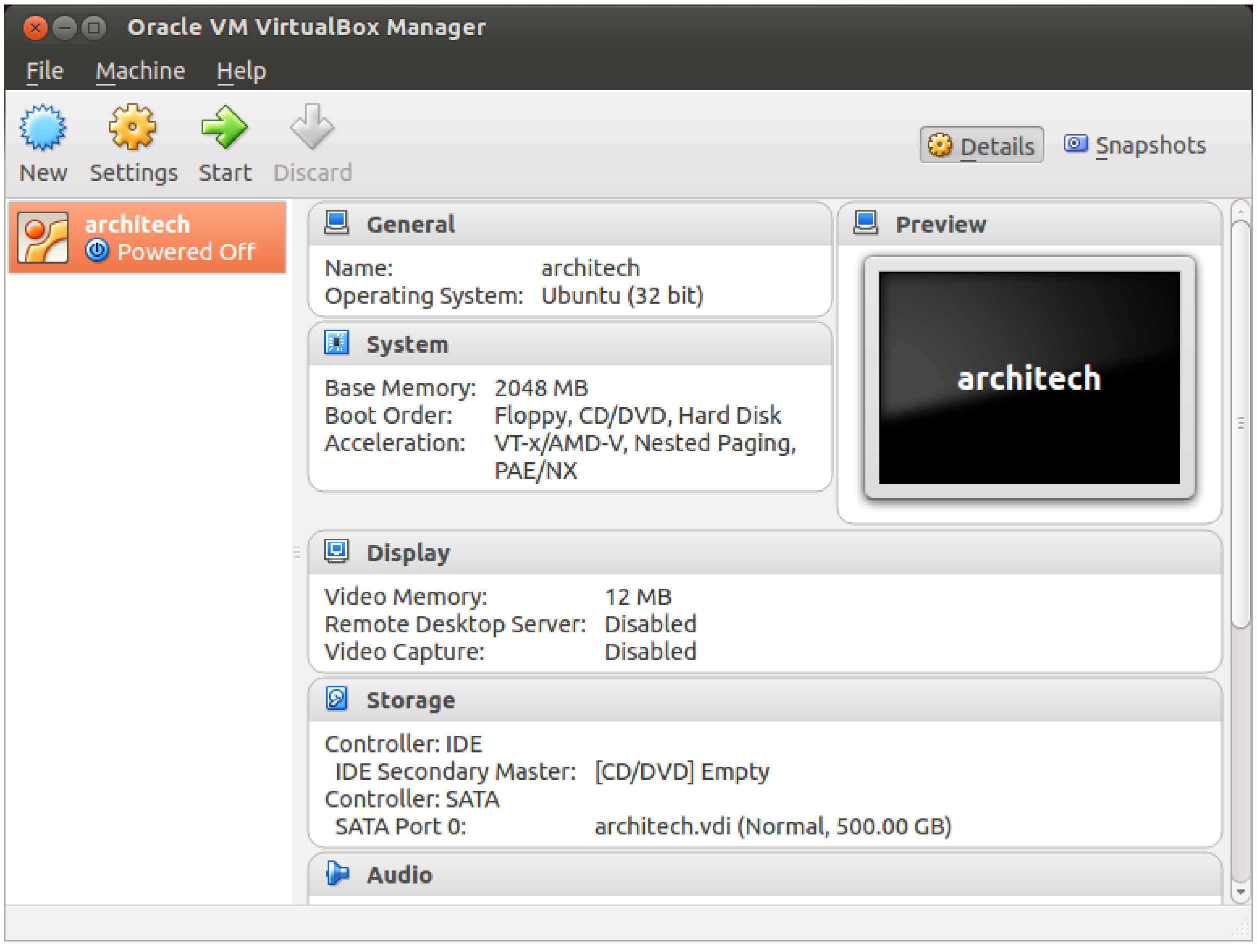
8. Click on *Ok*

Customize the number of processors

Building an entire system from the ground up is a business that can take up to several hours. To improve the performances of the overall build process, you can, if your computer has enough resources, assign more than one processor to the virtual machine.

Note: The virtual machine must be off

1. Select Architech's virtual machine from the list of virtual machines



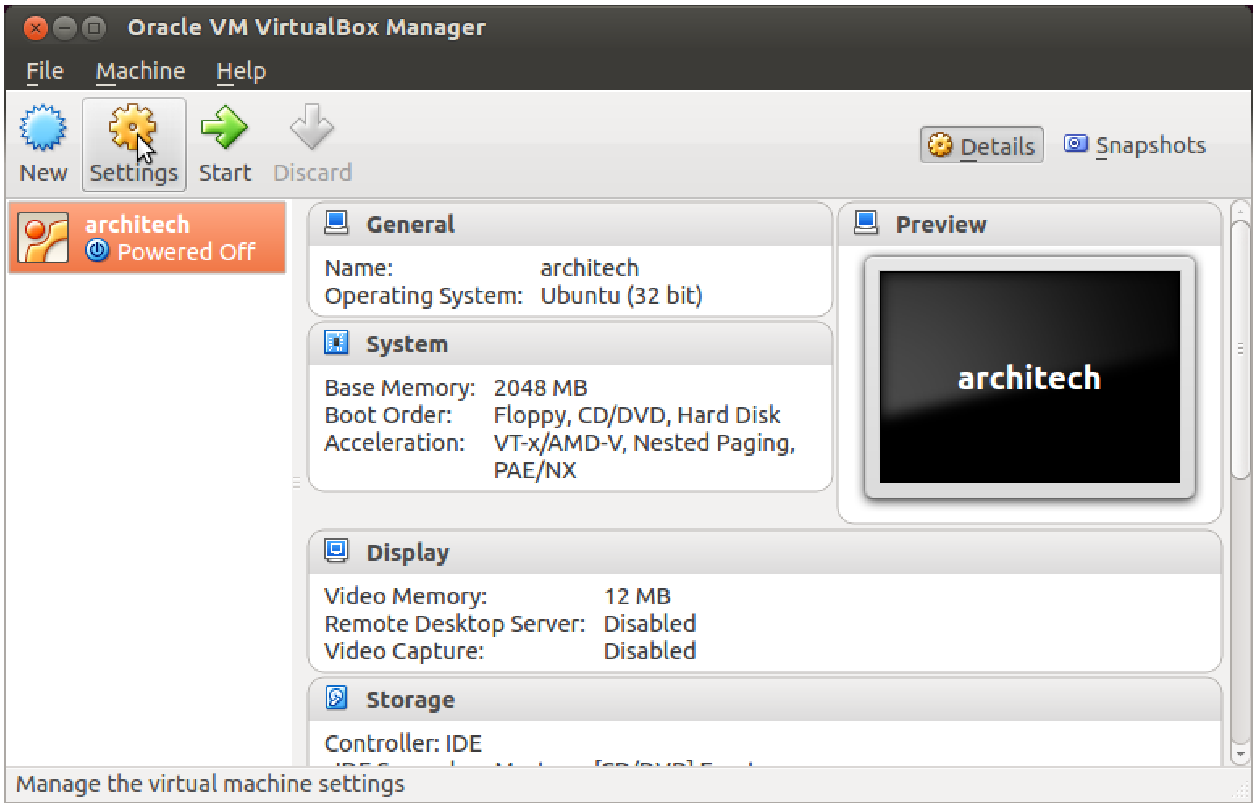
Name: architech
Operating System: Ubuntu (32 bit)

Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Controller: IDE
 IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
 SATA Port 0: architech.vdi (Normal, 500.00 GB)

2. Click on *Settings*



General

Name: architech
Operating System: Ubuntu (32 bit)

System

Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Display

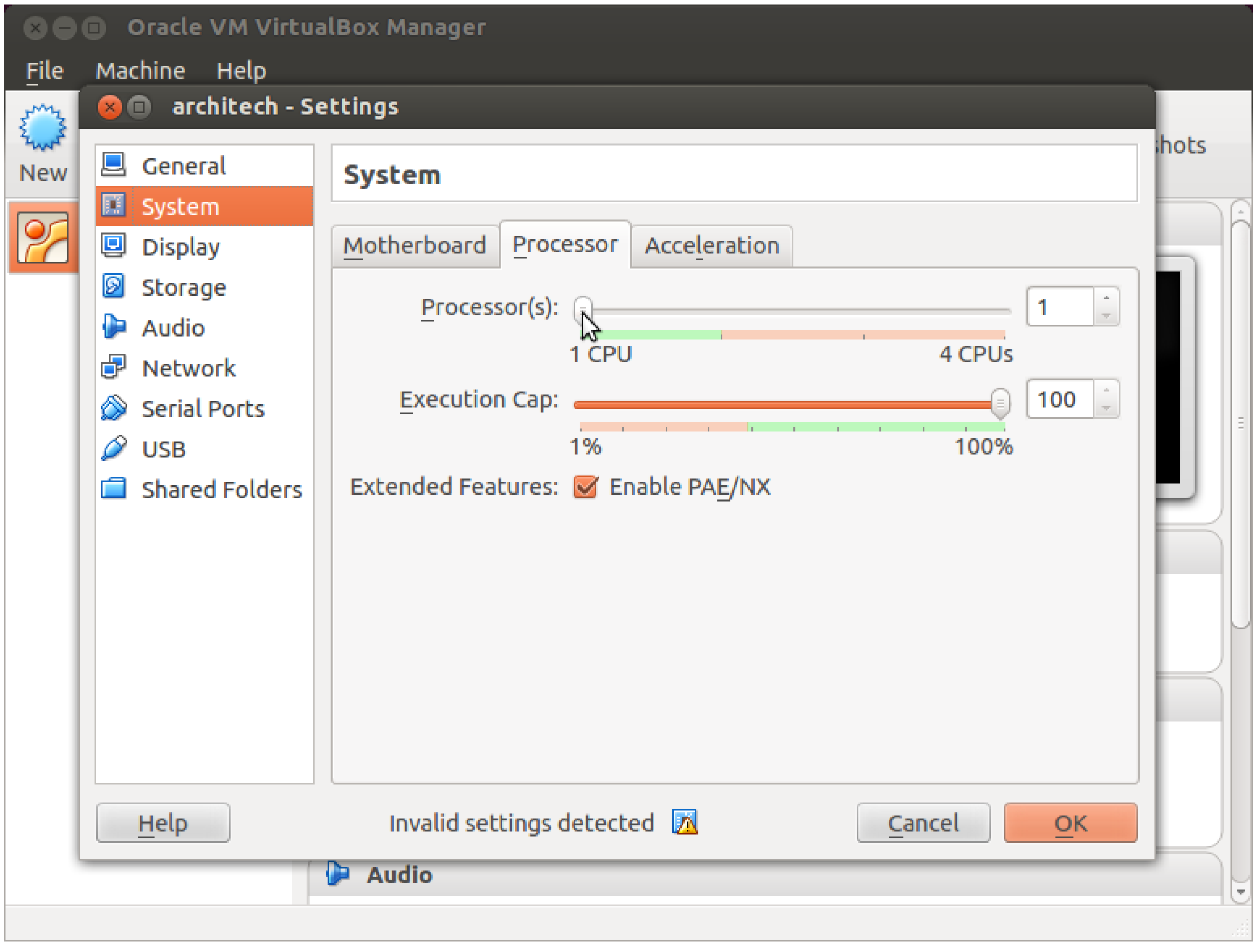
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: IDE

Manage the virtual machine settings

3. Select *System*
4. Select *Processor*
5. Assign the number of processors you wish to assign to the virtual machine



Processor(s): 1 (1 CPU – 4 CPUs)

Execution Cap: 100 (1% – 100%)

Extended Features: ✓ Enable PAE/NX

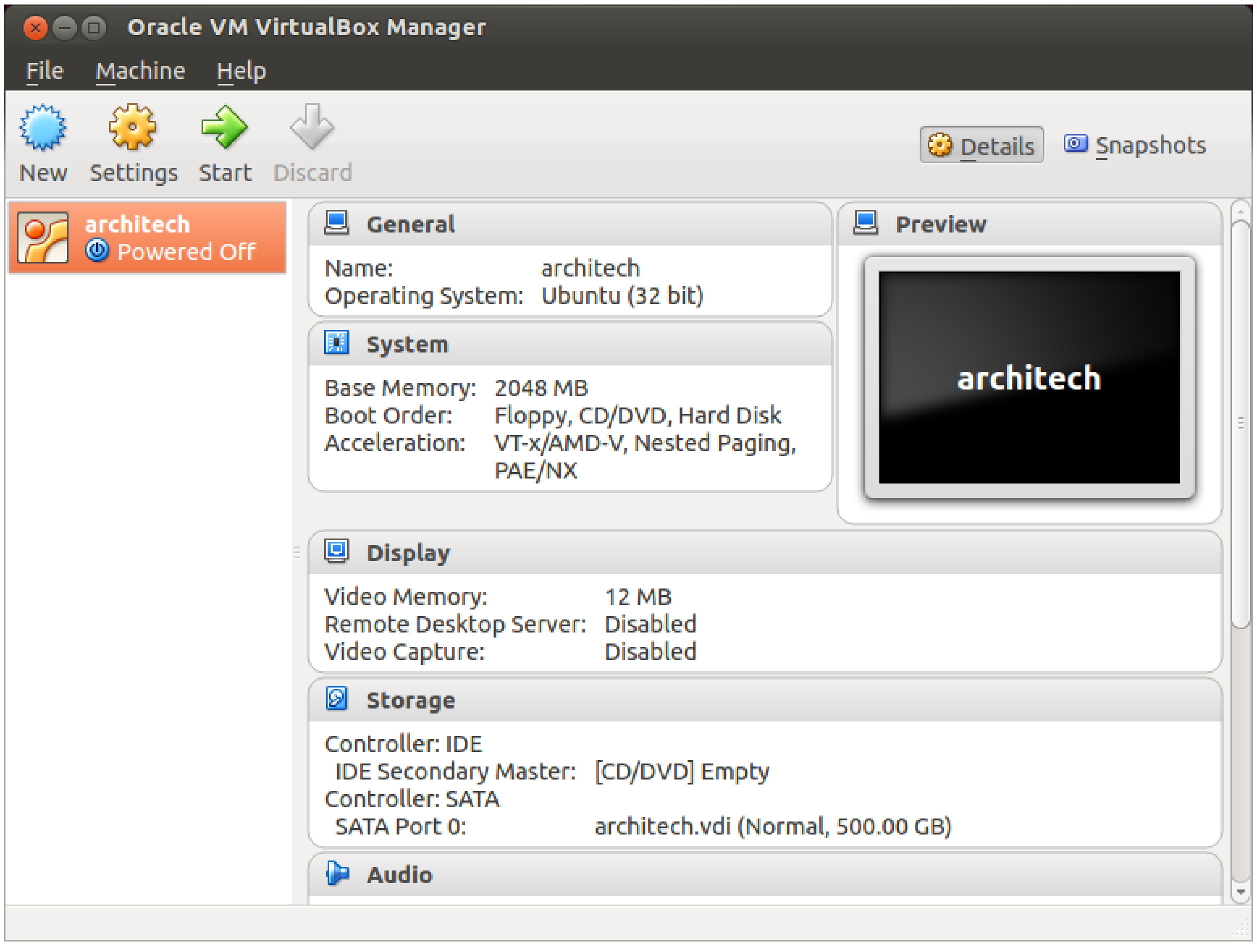
Invalid settings detected

Create a shared folder

A shared folder is way for host and guest operating systems to exchange files by means of the file system. You need to choose a directory on your host operating system to share with the guest operating system.

Note: The virtual machine must be off

1. Select Architech's virtual machine from the list of virtual machines



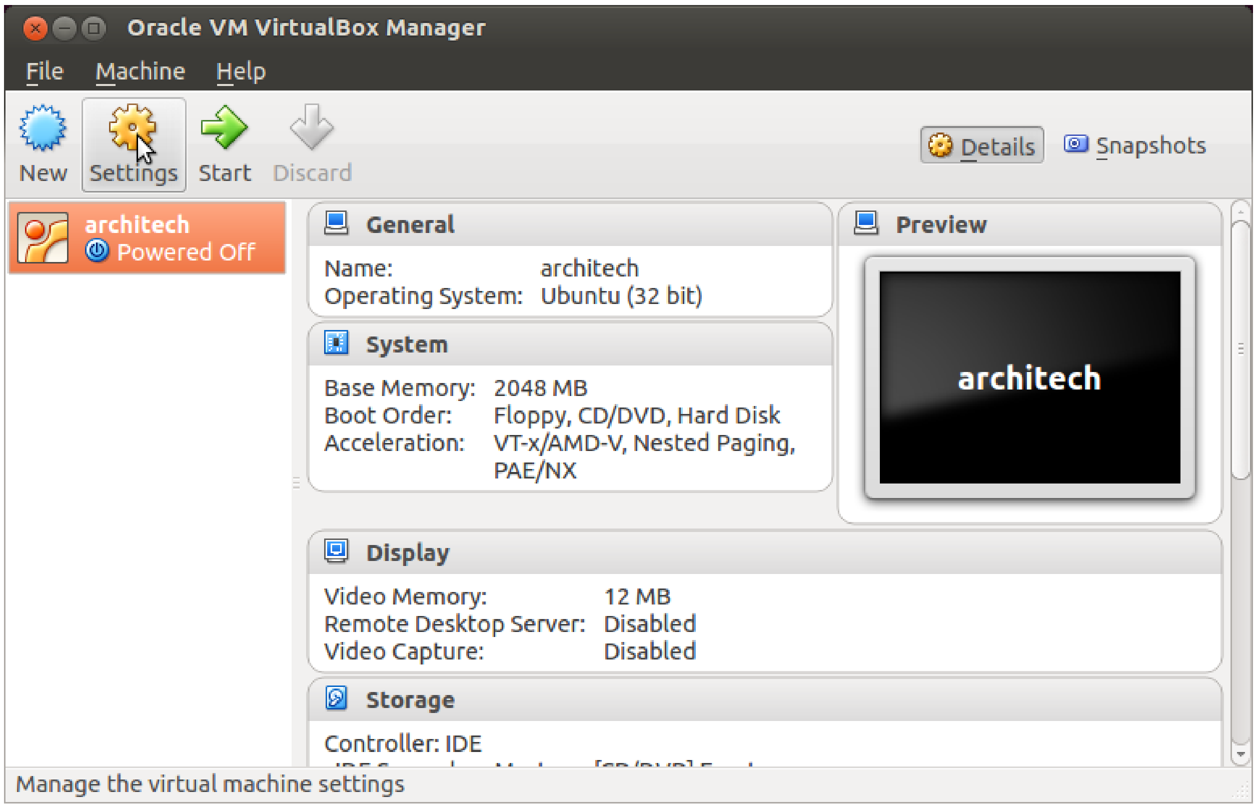
Name: architech
Operating System: Ubuntu (32 bit)

Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Controller: IDE
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: architech.vdi (Normal, 500.00 GB)

2. Click on *Settings*



architech
Powered Off

General

Name: architech
Operating System: Ubuntu (32 bit)

System

Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Display

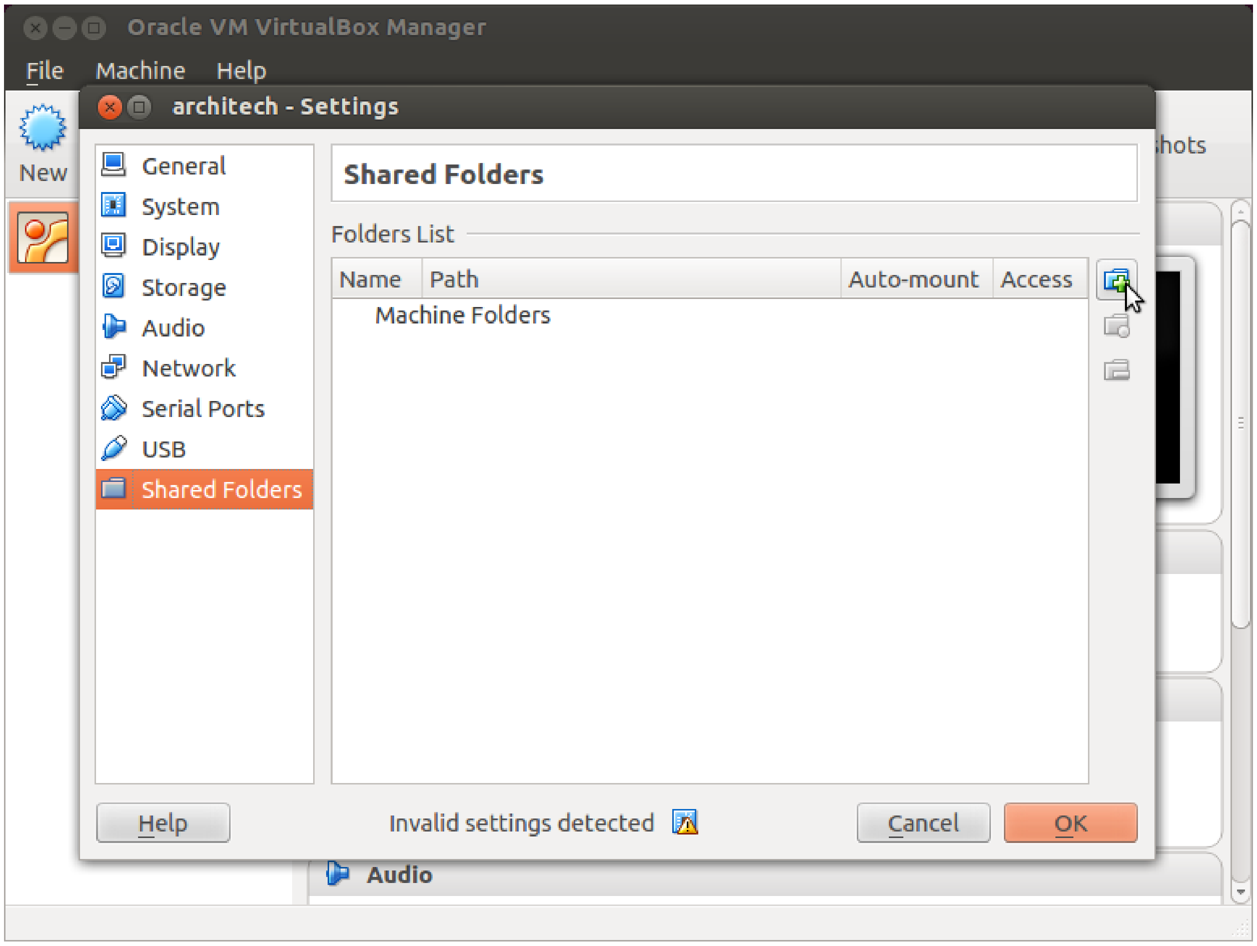
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: IDE

Manage the virtual machine settings

3. Select *Shared Folders*
4. Add a new shared folder



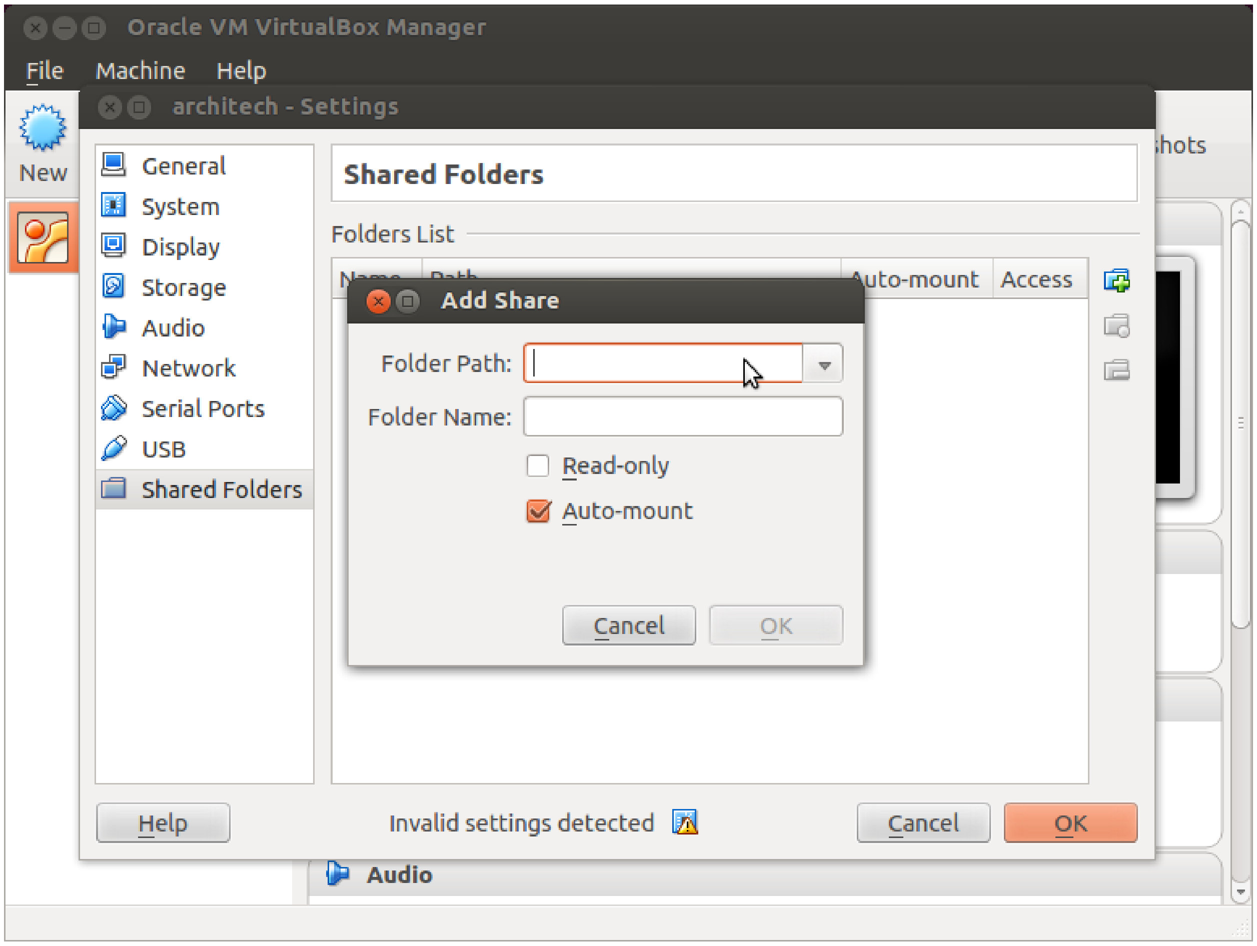
Shared Folders

Folders List

Name	Path	Auto-mount	Access
Machine Folders			

Invalid settings detected

5. Choose a directory to share on your host machine. Make sure *Auto-mount* is selected.

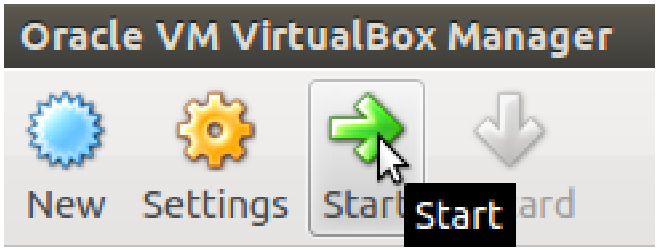


Once the virtual machine has been booted, the shared folder will be mounted under */media/* directory inside the virtual machine.

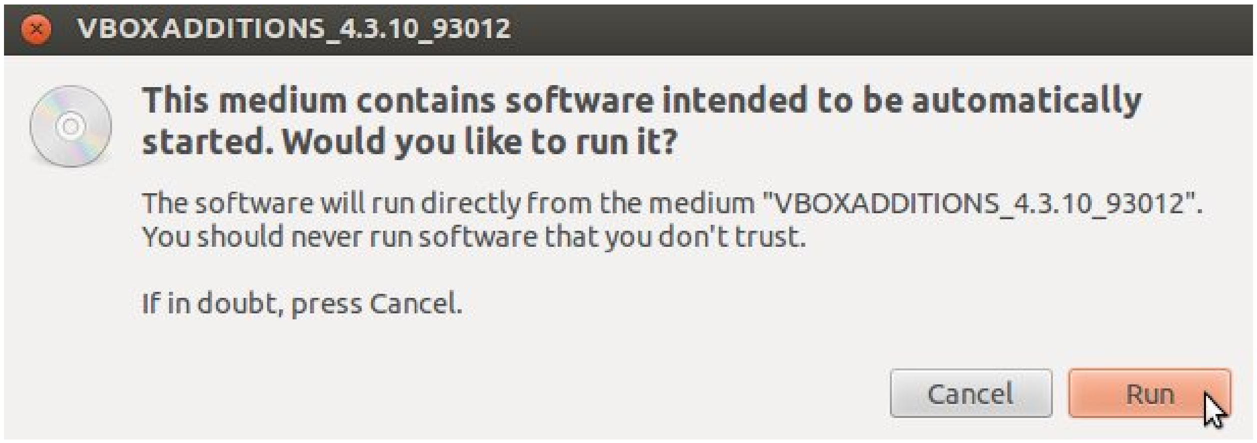
Install VBox Additions

The VBox addictions add functionalities to the virtual machine such as better graphic driver and more. It is already installed in the SDK but is important re-install it to configuring correctly the virtual machine with your operating system.

1. Starts the virtual machine



2. Click on the virtual box menu to the voice *Devices* and select *Insert Guest Additions CD Images...*. A message box will appear at the start of the installation, click on *run* button

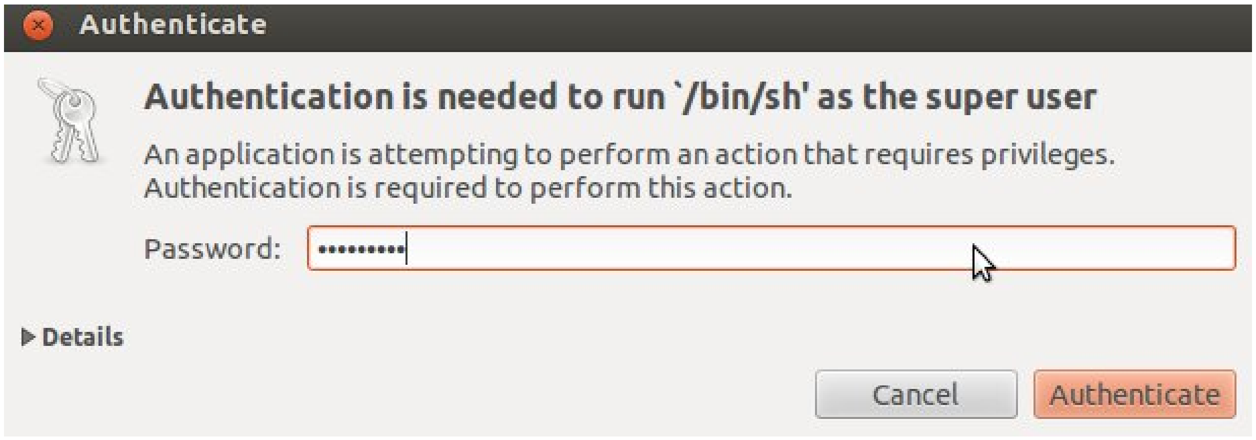

VBOXADDITIONS_4.3.10_93012

This medium contains software intended to be automatically started. Would you like to run it?

The software will run directly from the medium "VBOXADDITIONS_4.3.10_93012". You should never run software that you don't trust.

If in doubt, press Cancel.

4. To proceed are required admin privileges, so insert the password *architech* when asked

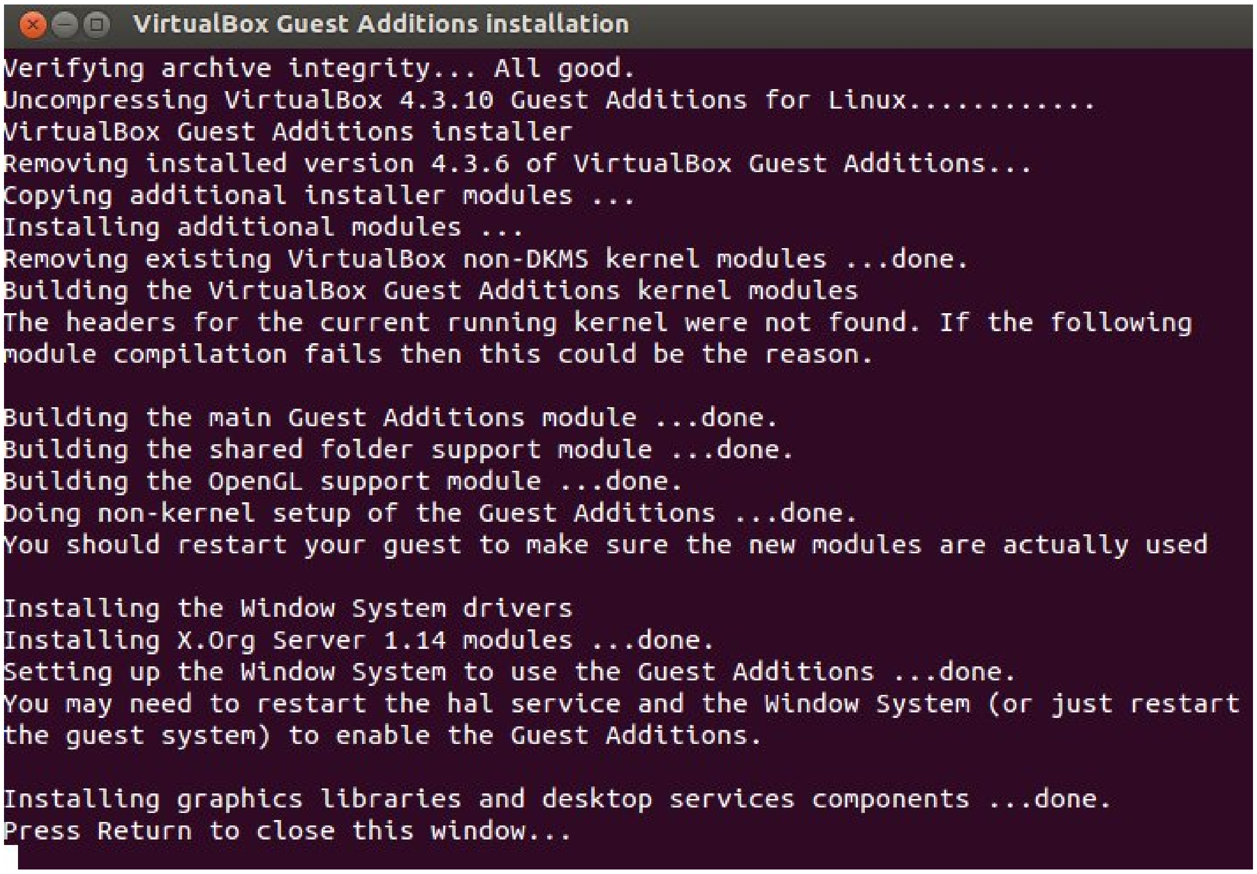

Authenticate

Authentication is needed to run `/bin/sh' as the super user

An application is attempting to perform an action that requires privileges. Authentication is required to perform this action.

Password: ••••••••••

5. Then a terminal will show the installation progress. When finished, press *Enter* key



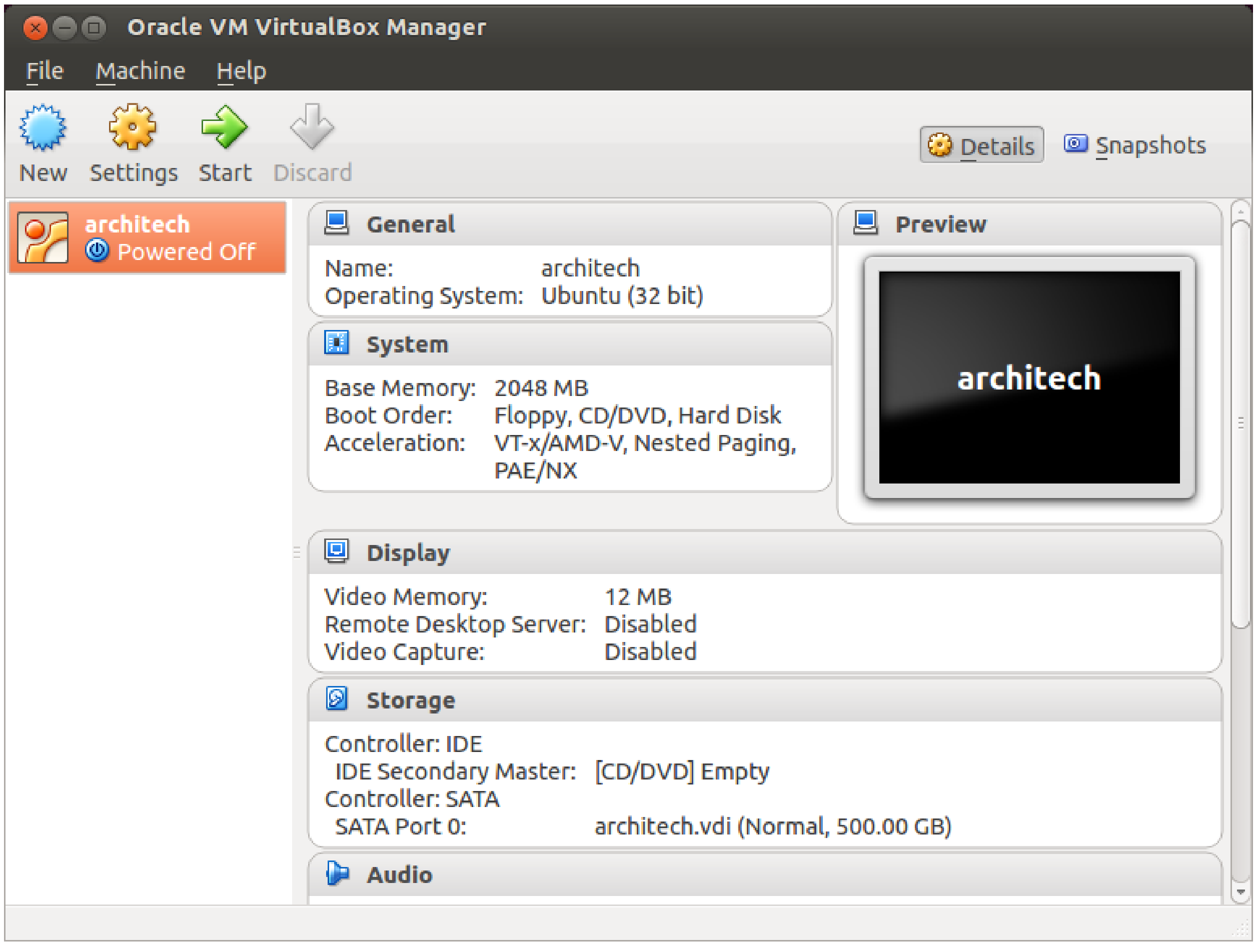
```
Verifying archive integrity... All good.
Uncompressing VirtualBox 4.3.10 Guest Additions for Linux...........
VirtualBox Guest Additions installer
Removing installed version 4.3.6 of VirtualBox Guest Additions...
Copying additional installer modules ...
Installing additional modules ...
Removing existing VirtualBox non-DKMS kernel modules ...done.
Building the VirtualBox Guest Additions kernel modules
The headers for the current running kernel were not found. If the following
module compilation fails then this could be the reason.

Building the main Guest Additions module ...done.
Building the shared folder support module ...done.
Building the OpenGL support module ...done.
Doing non-kernel setup of the Guest Additions ...done.
You should restart your guest to make sure the new modules are actually used

Installing the Window System drivers
Installing X.Org Server 1.14 modules ...done.
Setting up the Window System to use the Guest Additions ...done.
You may need to restart the hal service and the Window System (or just restart
the guest system) to enable the Guest Additions.

Installing graphics libraries and desktop services components ...done.
Press Return to close this window...
```

6. Before to use the SDK, it is required reboot the virtual machine

Build

Important: A working internet connection, several GB of free disk space and several hours are required by the build process

1. Select Architech's virtual machine from the list of virtual machines inside Virtual Box application



General
Name: architech
Operating System: Ubuntu (32 bit)

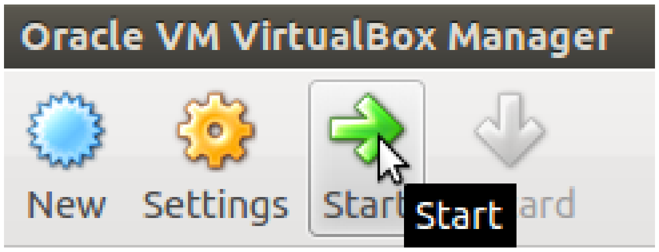
System
Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Display
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

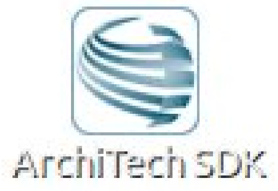
Storage
Controller: IDE
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: architech.vdi (Normal, 500.00 GB)

Audio

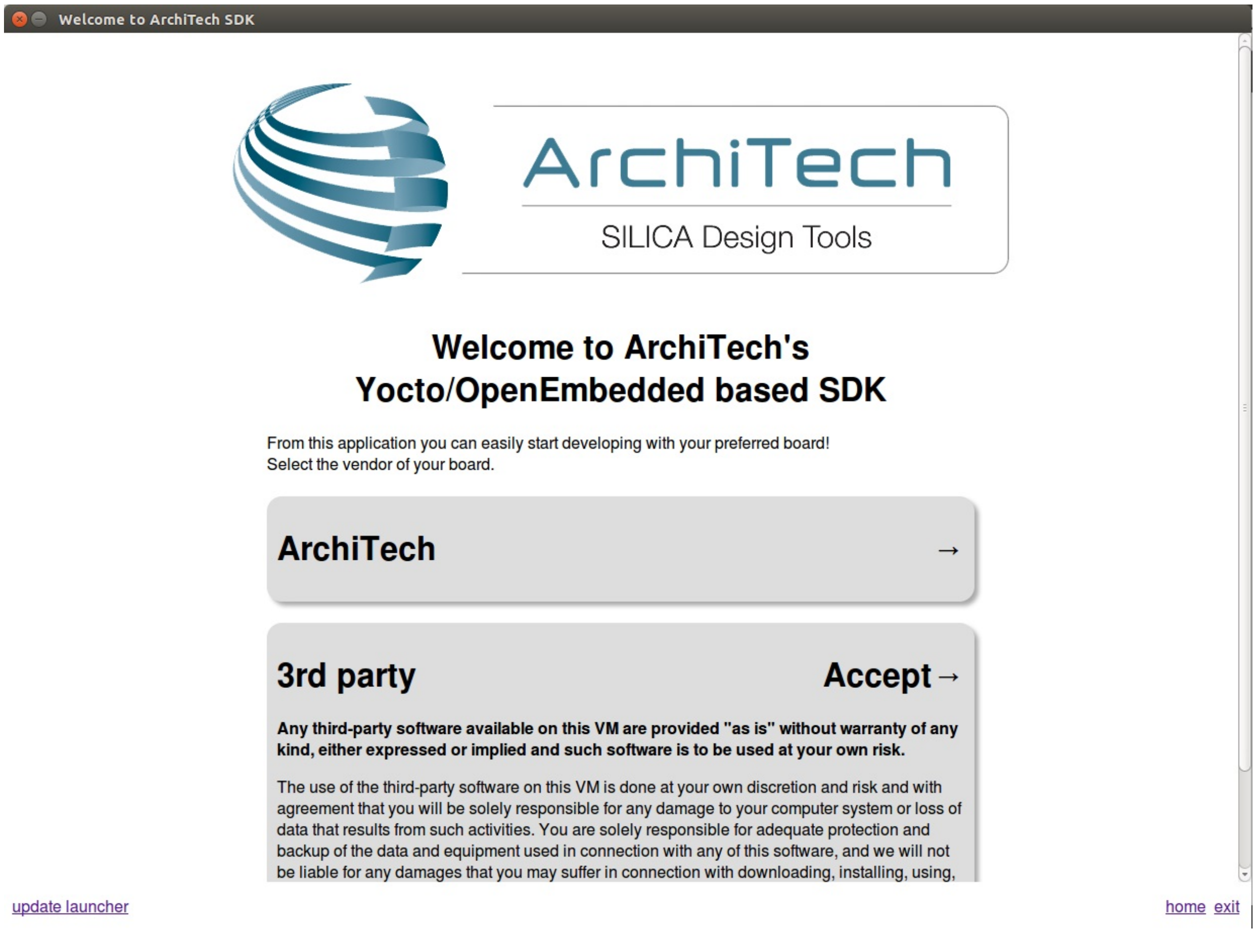
2. Click on the icon *Start* button in the toolbar and wait until the virtual machine is ready



3. Double click on *Architech SDK* icon you have on the virtual machine desktop.



4. The first screen gives you two choices: *ArchiTech* and *3rd Party*. Choose *ArchiTech*.



Welcome to ArchiTech SDK

ArchiTech
SILICA Design Tools

Welcome to ArchiTech's
Yocto/OpenEmbedded based SDK

From this application you can easily start developing with your preferred board!
Select the vendor of your board.

ArchiTech →

3rd party Accept→

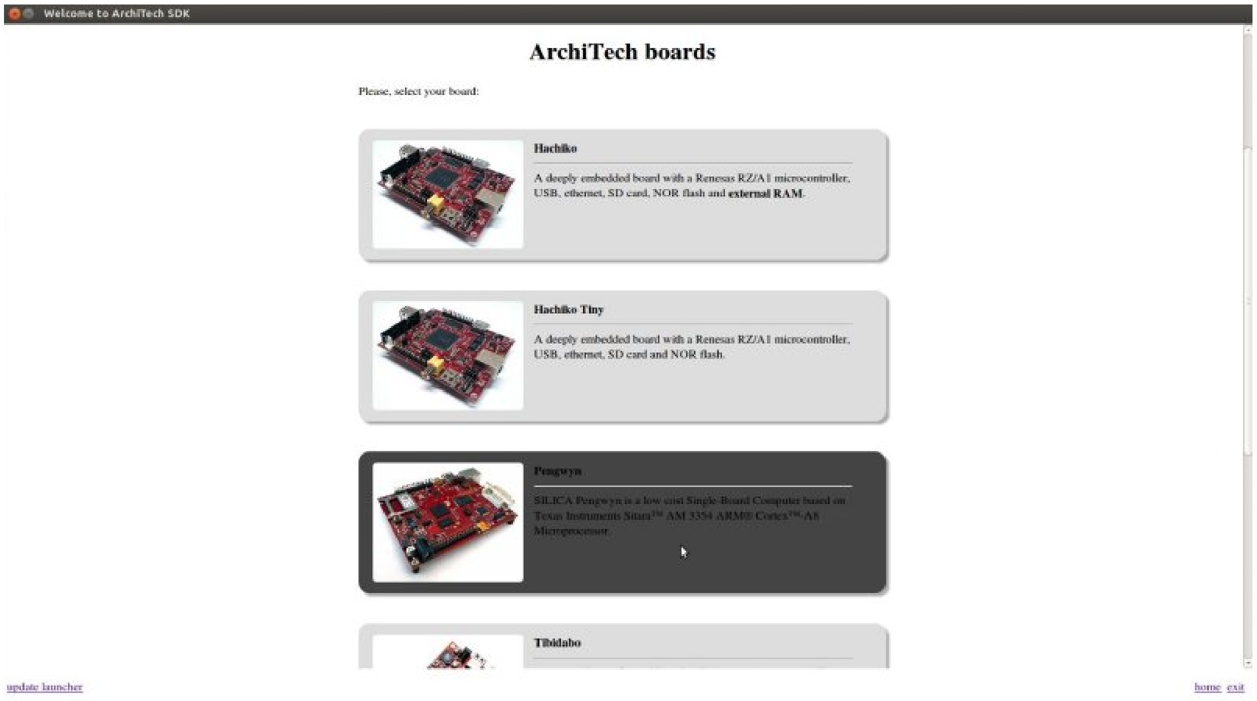
Any third-party software available on this VM are provided "as is" without warranty of any kind, either expressed or implied and such software is to be used at your own risk.

The use of the third-party software on this VM is done at your own discretion and risk and with agreement that you will be solely responsible for any damage to your computer system or loss of data that results from such activities. You are solely responsible for adequate protection and backup of the data and equipment used in connection with any of this software, and we will not be liable for any damages that you may suffer in connection with downloading, installing, using,

update launcher

home exit

5. Select Pengwyn as board you want develop on.



Welcome to ArchiTech SDK

ArchiTech boards

Please, select your board:

Hachiko

A deeply embedded board with a Renesas RZ/A1 microcontroller, USB, ethernet, SD card, NOR flash and **external RAM**.

Hachiko Tiny

A deeply embedded board with a Renesas RZ/A1 microcontroller, USB, ethernet, SD card and NOR flash.

Pengwyn

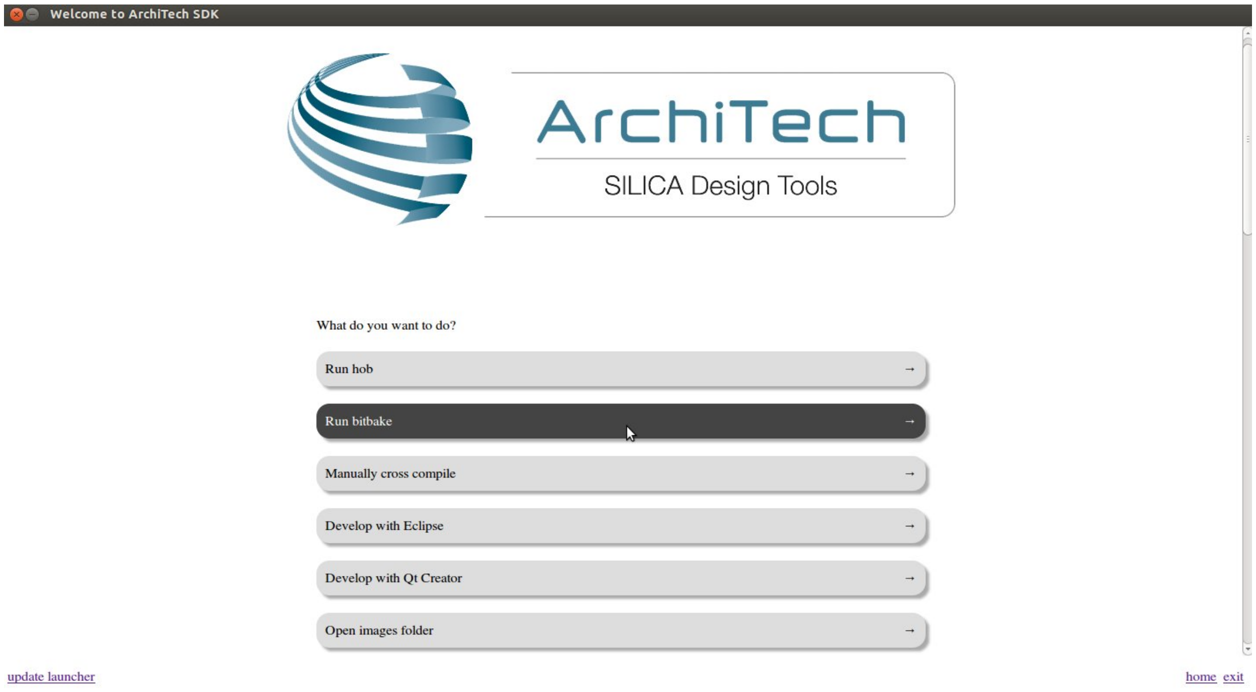
SILICA Pengwyn is a low cost Single-Board Computer based on Texas Instruments Sitara™ AM 3354 ARM® Cortex™-A8 Microprocessor.

Tibidabo

update launcher

home exit

6. A new screen opens up from where you can perform a set of actions. Click on *Run bitbake* to obtain a terminal ready to start to build an image.



7. Open *local.conf* file:
8. Go to the end of the file and add the following lines:

This will trigger the installation of a features set onto the final root file system, like *tcf-agent* and *gdbserver*.

9. Save the file and close gedit.
10. Build *core-image-minimal-dev* image by means of the following command:

At the end of the build process, the image will be saved inside directory:

11. Setup *sysroot* directory on your host machine:

Note: **sudo** password is: “**architech**“

Deploy

To deploy the root file system and the boot partition, first you need a SD card correctly formatted.

How to create the SD card

This section describes the steps to be followed to create a standalone bootable system on **SD card**.

Ensure that the following is available:

- An SD memory card reader/programmer to copy files from the Linux Host. The SD card reader must be accessible from VirtualBox.
- An USB device to read and write a SD card.

Warning: Not all computer built-in readers can be used, use USB connected devices instead.

- An SD card

Warning: Your sd-card will be formatted and all the data contained in it will be lost forever!

- The tool scripts (download link pengwyn-tools-dizzy.tar.bz2) and decompress in a directory. eg:
- A Linux host with fdisk, sfdisk, mkfs.ext3 and mkfs.vfat utilities. If you are running the VM with Ubuntu pre-installed and the environment already in place, all the packages are already available.
- The files *MLO*, *u-boot.img*, *zImage-pengwyn-dvi.dtb*, *zImage*, (optionally) the kernel modules (*modules-3.2.0-rX-pengwyn.tgz*, where X is the revision number for the kernel modules, without any modification to the kernel configuration it should be 0) and the *root file system* with name ending with **-pengwyn.tar.gz** are available inside the images deploy directory (*<build directory>/tmp/deploy/images/pengwyn/*, if your build directory is the default one, then the *deploy directory* is */home/architech/architech_sdk/architech/pengwyn/yocto/build/tmp/deploy/images/pengwyn*).

Note:

The scripts will get the latest files from folder
/home/architech/architech_sdk/architech/pengwyn/yocto/build/tmp/deploy/images/pengwyn.

If you want to specify a custom directory where there are the Yocto images, enter the path directly after the script name, e.g.:
sudo ./fast-create-sdcard.sh /home/architech/custom-dir

How to build the SD card

- Run the VM on VirtualBox
- Connect your SD card reader to your computer than to the virtual machine, from VirtualBox menu select *Devices → USB Devices → "your SDcard reader"*.
- Insert the SD card into the adapter (in this example we are inserting an SD card already partitioned with a FAT and an EXT3 partition, that is the basic configuration for the Pengwyn board).
- Run the following command (sudo password: **architech**)
- Find the device name from fdisk command output, in this example is */dev/sdb*.
- Run the script (in pengwyn-tools folder) that will prepare the SD card with all the needed files to run the system:

When the script starts asks for the *sudo password*, type *architech* followed by *enter*.
The list of available devices will be shown: the SD card should be the number **1** of the list with name sdb. Check the size shown on the table to be sure that the device is the correct one. Enter the device number **1** followed by enter-key.

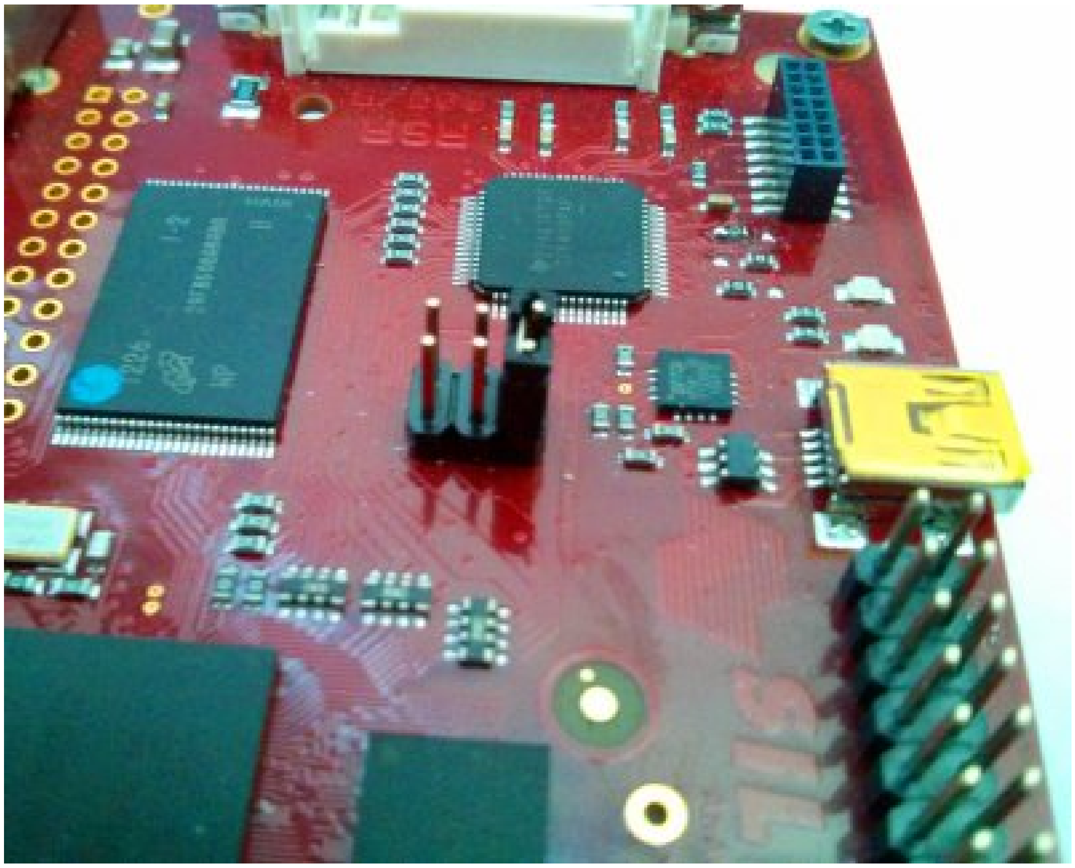
The script will create two partitions on the SD card: the first one is a **FAT32** with the **boot files** (*MLO*, *u-boot.img*, *zImage-pengwyn-dvi.dtb*, *zImage* will be renamed to *MLO*, *u-boot.img*, *pengwyn.dtb* and *zImage*), the second one is an *ext3* with the *target file system*.
The operations will take few minutes.

Make sure everything has been really written to the SD card:

Then unmount the SD card from your computer and plug the SD in the board socket.

Boot

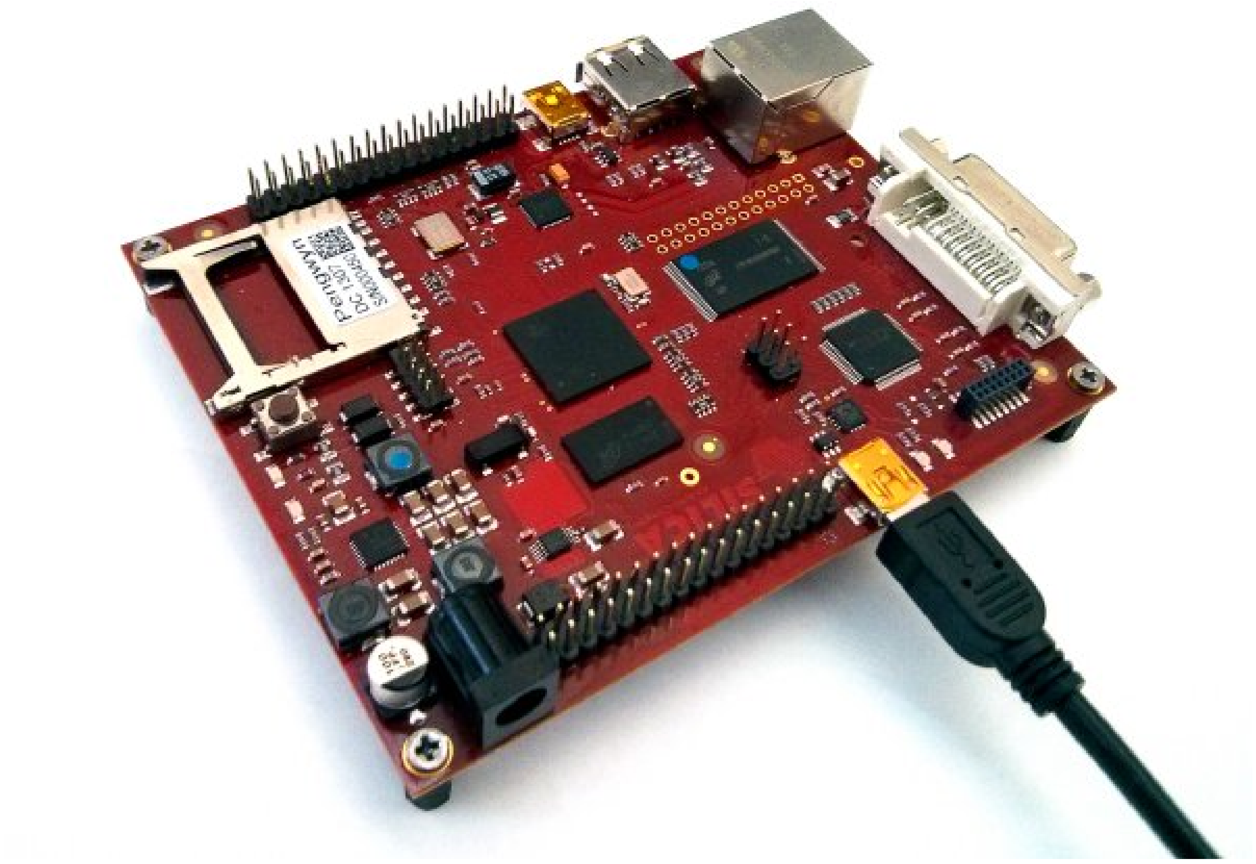
First of all, make sure the board can boot entirely from the SD-Card by setting **J1** closed and **J2** and **J3** opened:



Pengwyn takes the power from the mini-USB connector **CN6** and/or connector **CN1**.

Now it’s time to start the serial console.

On Pengwyn you can use the same USB cable used to power up the board to get access to the serial console.
The serial console connector **CN6**



which you can connect, by means of a mini-USB cable, to your personal computer.

Note: Every operating system has its own killer application to give you a serial terminal interface. In this guide, we are assuming your **host** operating system is **Ubuntu**.

On a Linux (Ubuntu) host machine, the console is seen as a *ttyUSB***X** device (where X is a number) and you can access to it by means of an application like *minicom*.

Minicom needs to know the name of the serial device. The simplest way for you to discover the name of the device is by looking to the kernel messages, so:

1. clean the kernel messages
2. connect the mini-USB cable to the board already powered-on
3. display the kernel messages
3. read the output

As you can see, here the device has been recognized as */dev/ttyUSB0*.

Now that you know the device name, run minicom:

If minicom is not installed, you can install it with:

then you can setup your port with these parameters:

If on your system the device has not been recognized as */dev/ttyUSB0*, just replace */dev/ttyUSB0* with the proper device.

Once you are done configuring the serial port, you are back to minicom main menu and you can select *exit*.

Give *root* to the login prompt:

Board

```
pengwyn login: root
```

and press *Enter*.

Note: Sometimes, the time you spend setting up minicom makes you miss all the output that leads to the login and you see just a black screen, press *Enter* then to get the login prompt.

Code

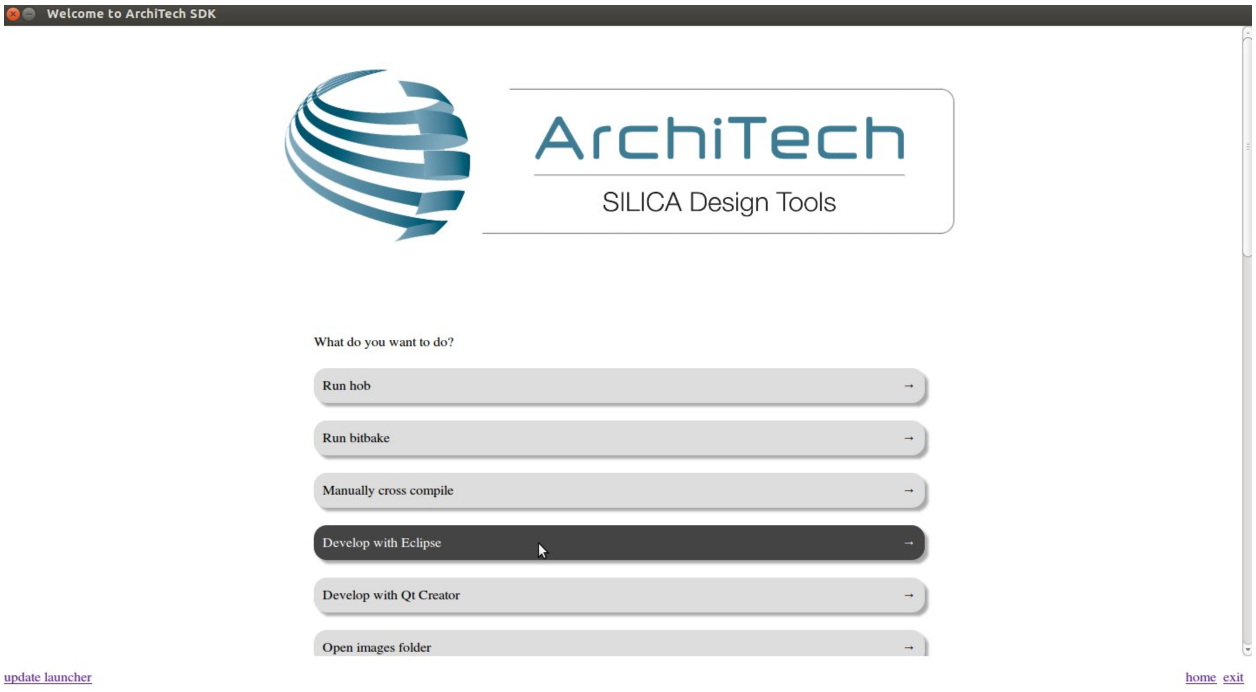
The time to create a simple *HelloWorld!* application using **Eclipse** has come.

Note:

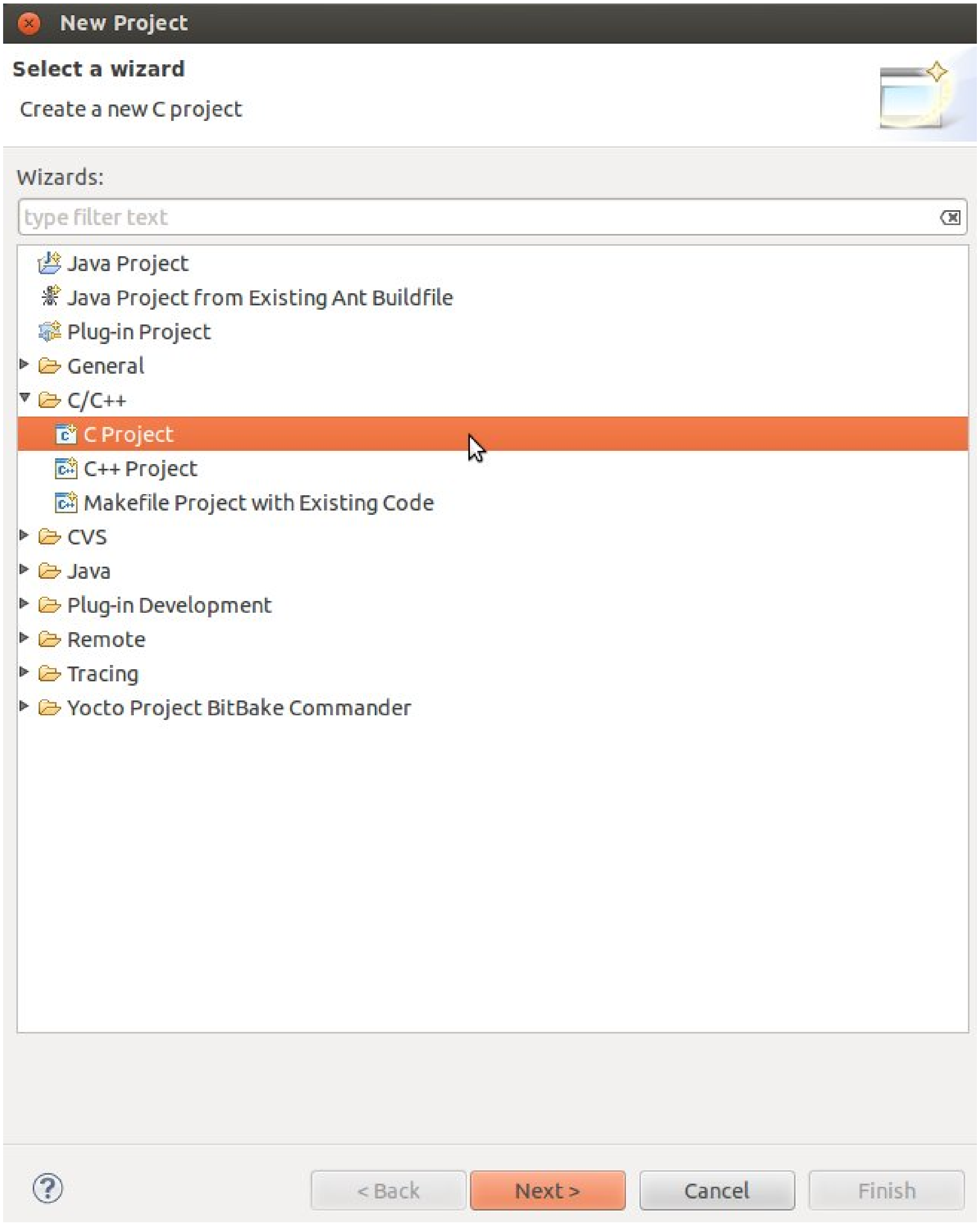
Before to start remember to copy the cross-toolchain libreries to sysroot

```
cp -r
/home/architech/architech_sdk/architech/pengwyn/toolchain/sysroots/cortexa8t2hf-vfp-neon-poky-linux-gnueabi/*
/home/architech/architech_sdk/architech/pengwyn/sysroot
```

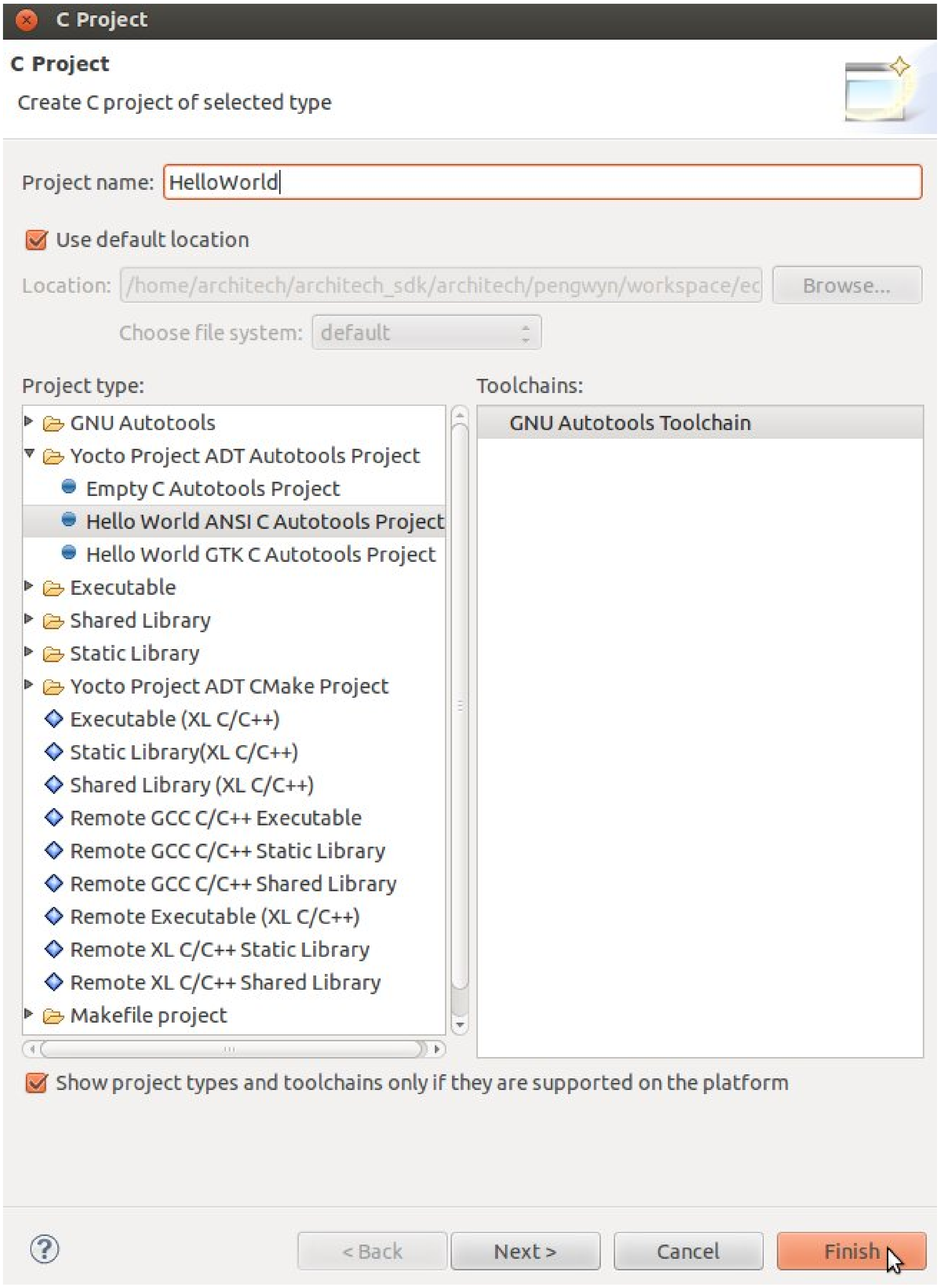
1. Return to the **Splashscreen**, which we left on Pengwyn board screen, and click on *Develop with Eclipse*.



2. Go to *File*→ *New*→ *Project...*, in the node "C/C++" select *C Project* and press *next* button.



3. Insert *HelloWorld* as project name, open the node *Yocto Project ADT Autotools Project* and select *Hello World ANSI C Autotools Project* and press *next* button.



C Project

C Project

Create C project of selected type

Project name: HelloWorld

[✓] Use default location

Location: /home/architech/architech_sdk/architech/pengwyn/workspace/ec

Choose file system: default

Project type:

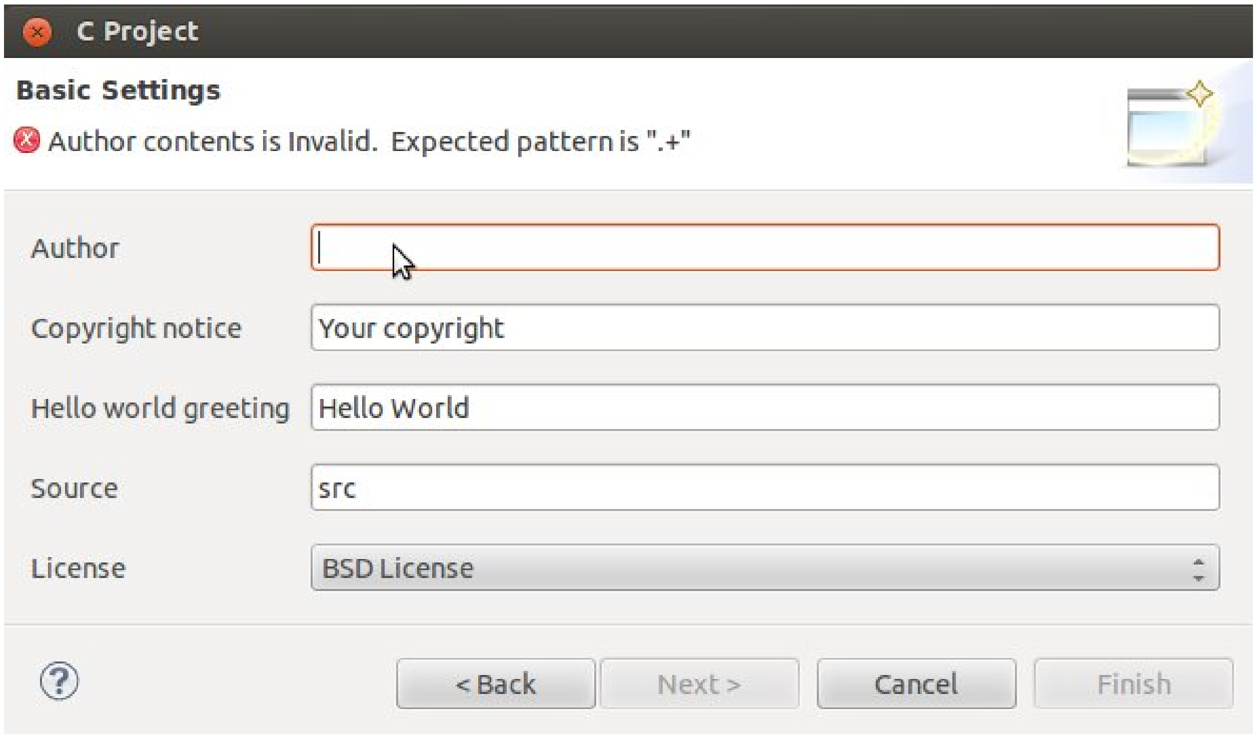
- GNU Autotools
- Yocto Project ADT Autotools Project
  - Empty C Autotools Project
  - Hello World ANSI C Autotools Project
  - Hello World GTK C Autotools Project
- Executable
- Shared Library
- Static Library
- Yocto Project ADT CMake Project
- Executable (XL C/C++)
- Static Library(XL C/C++)
- Shared Library (XL C/C++)
- Remote GCC C/C++ Executable
- Remote GCC C/C++ Static Library
- Remote GCC C/C++ Shared Library
- Remote Executable (XL C/C++)
- Remote XL C/C++ Static Library
- Remote XL C/C++ Shared Library
- Makefile project

Toolchains:

GNU Autotools Toolchain

[✓] Show project types and toolchains only if they are supported on the platform

4. Insert *Author* field and click on *Finish* button. Select *Yes* on the *Open Associated Perspective?* question.



C Project

Basic Settings

Author contents is Invalid. Expected pattern is ".+"

Author:

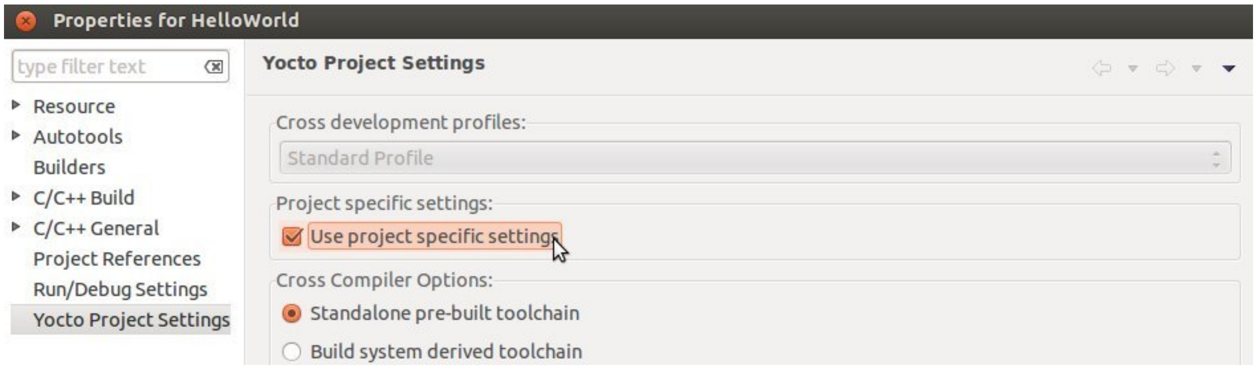
Copyright notice: Your copyright

Hello world greeting: Hello World

Source: src

License: BSD License

5. Open the windows properties clicking on *Project*→ *Properties* and select *Yocto Project Settings*. Check *Use project specific settings* in order to use the pengwyn cross-toolchain.



Properties for HelloWorld

Yocto Project Settings

Cross development profiles: Standard Profile

Project specific settings: ✓ Use project specific settings

Cross Compiler Options: ● Standalone pre-built toolchain ○ Build system derived toolchain

5. Click on *OK* button and build the project by selecting *Project*→ *Build All*.

Debug

Use an ethernet cable to connect the board (connector XF1) to your PC. Configure your workstation ip address as 192.168.0.100. Make sure the board can be seen by your host machine:

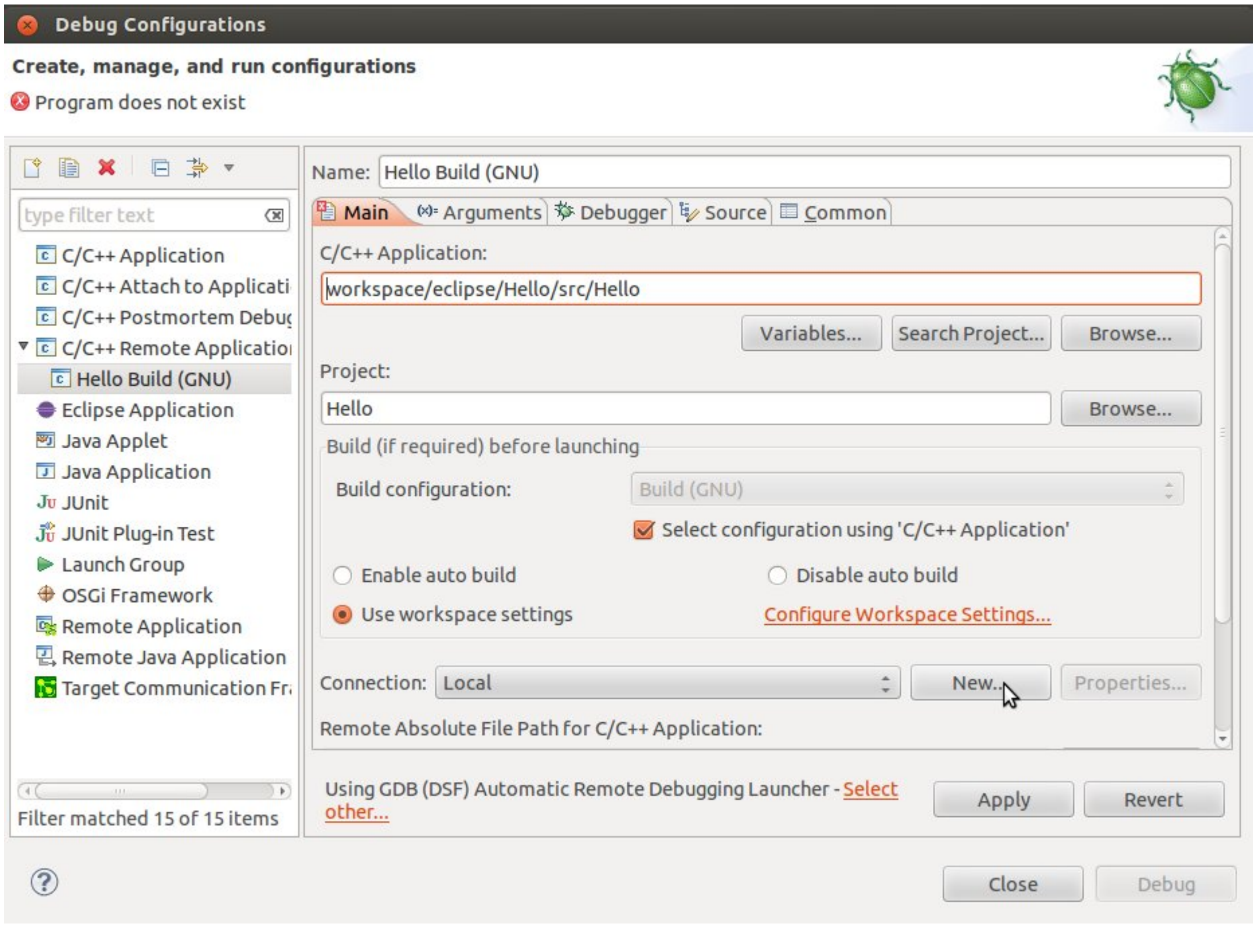
If the output is similar to this one:

then the ethernet connection is ok. Enable the remote debug with Yocto by typing this command on Pengwyn console:

On the Host machine, follow these steps to let **Eclipse** deploy and debug your application:

- Select *Run*→ *Debug Configurations...*
- In the left area, expand *C/C++Remote Application*.

- Locate your project and select it to bring up a new tabbed view in the *Debug Configurations* Dialog.



Name: Hello Build (GNU)

C/C++ Application: workspace/eclipse/Hello/src/Hello

Project: Hello

Build (if required) before launching

Build configuration: Build (GNU)

[✓] Select configuration using 'C/C++ Application'

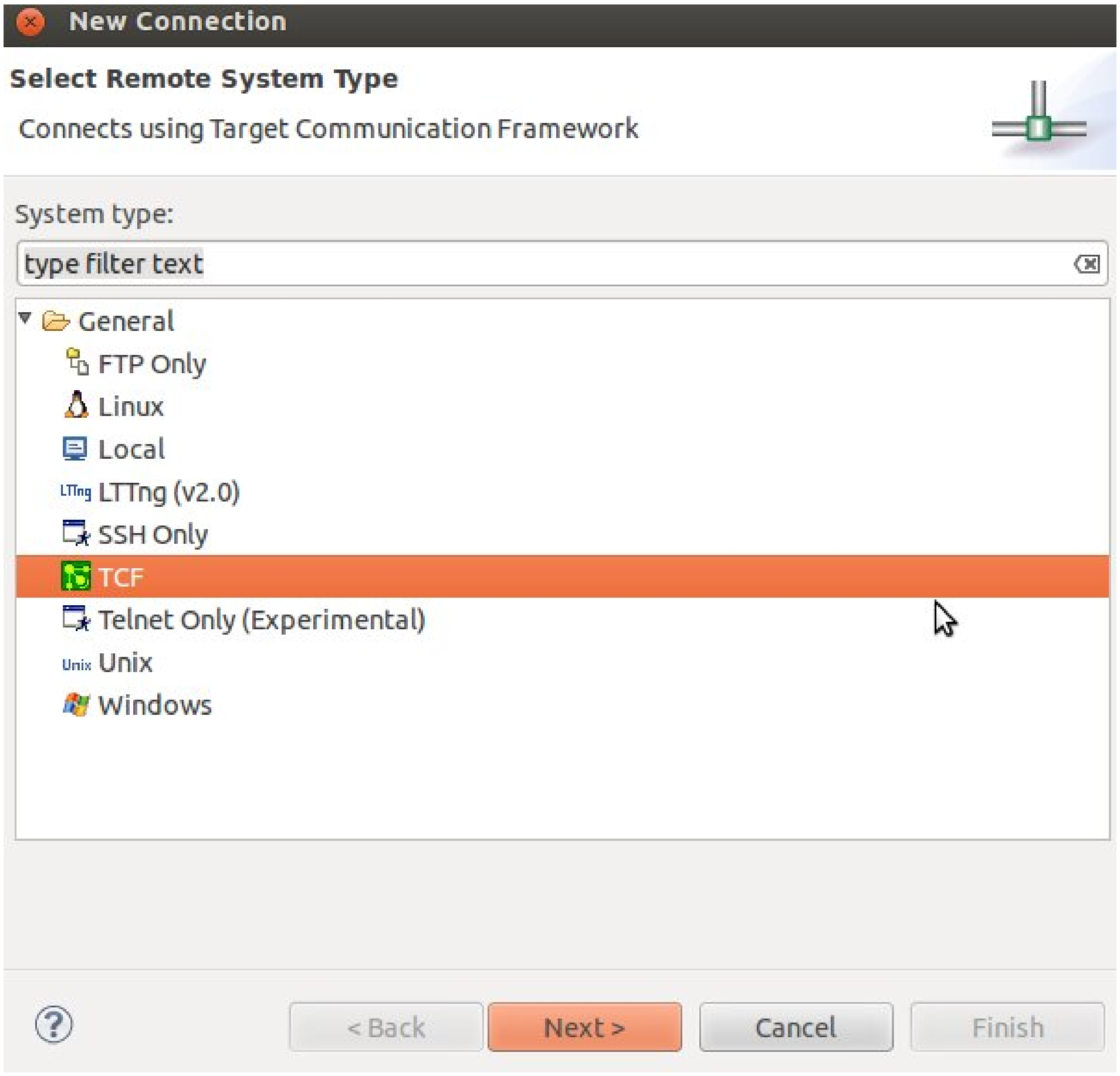
Enable auto build ( ) Disable auto build ( ) [✓ Use workspace settings] Configure Workspace Settings...

Connection: Local

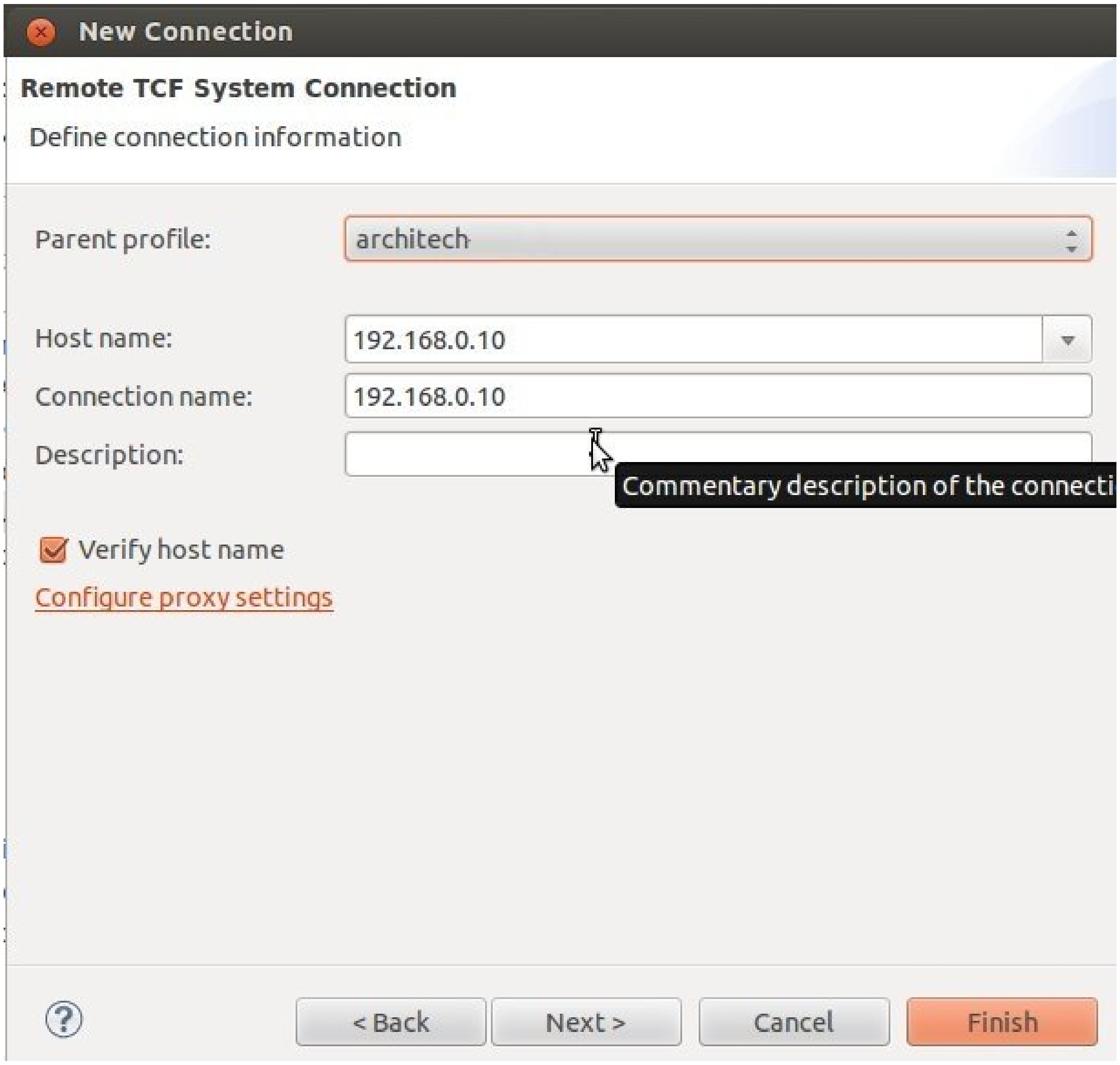
Remote Absolute File Path for C/C++ Application:

Using GDB (DSF) Automatic Remote Debugging Launcher - Select other...

- Insert in *C/C++ Application* the filepath (on your host machine) of the compiled binary.
- Click on *New* button near the drop-down menu in the *Connection* field.
- Select *TCF* icon.



- Insert in *Host Name* and *Connection Name* fields the IP address of the target board. (e.g. 192.168.0.10)



New Connection

Remote TCF System Connection

Define connection information

Parent profile: architech

Host name: 192.168.0.10

Connection name: 192.168.0.10

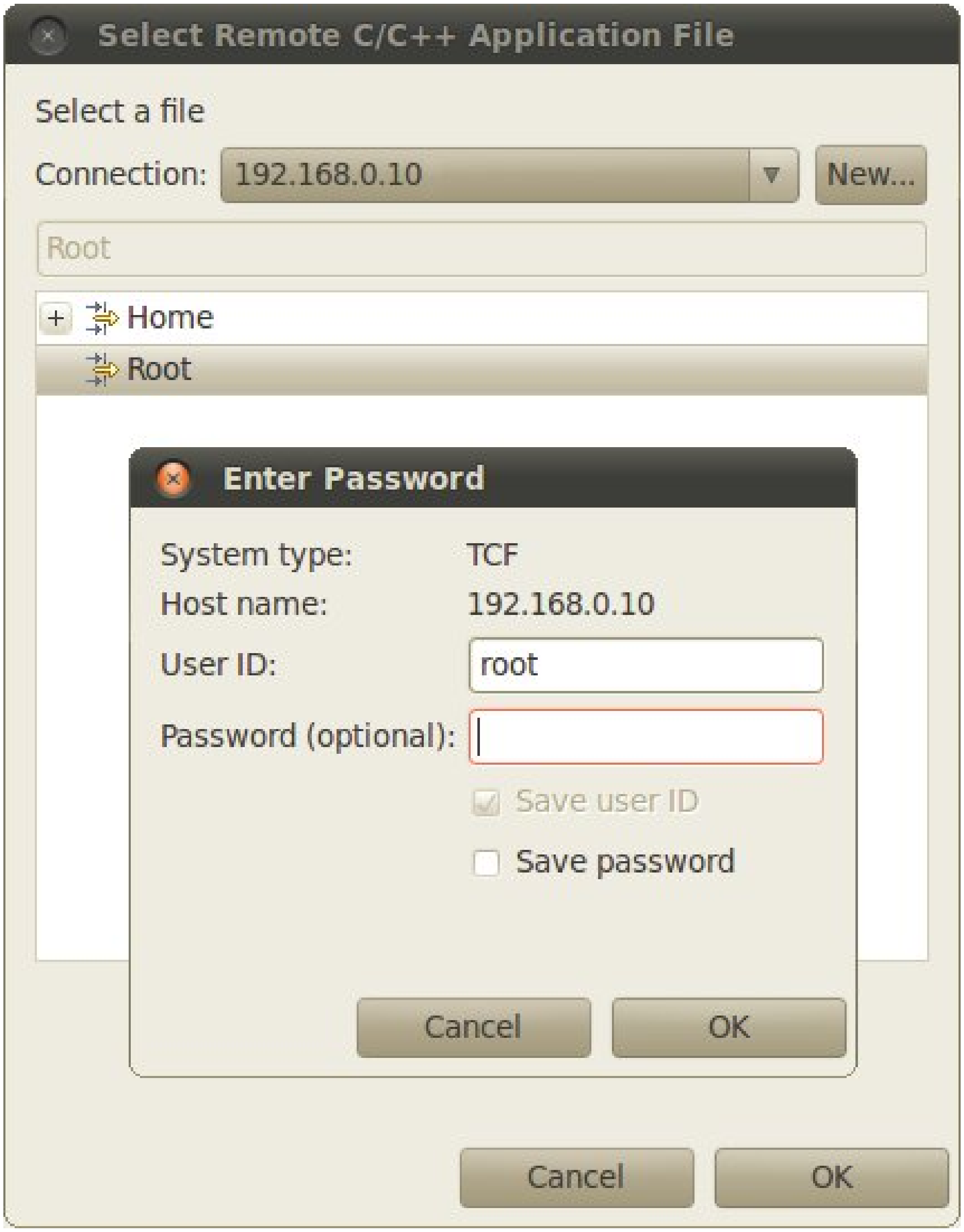
Description:

Commentary description of the connecti

[✓] Verify host name

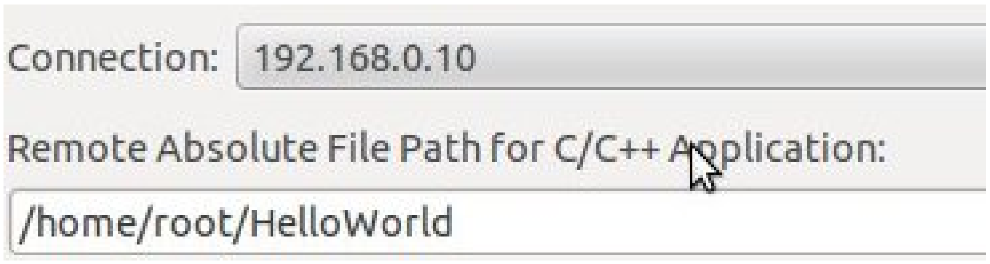
Configure proxy settings

- Then press *Finish*.
- Use the drop-down menu now in the *Connection* field and pick up the IP Address you entered earlier.
- Enter the absolute path on the target into which you want to deploy the cross-compiled application. Use the *Browse* button near *Remote Absolute File Path for C/C++Application:* field. No password is needed.



System type: TCF
Host name: 192.168.0.10
User ID: root
Password (optional):
[✓ Save user ID]
Save password

• Enter also in the path the name of the application you want to debug. (e.g. HelloWorld)

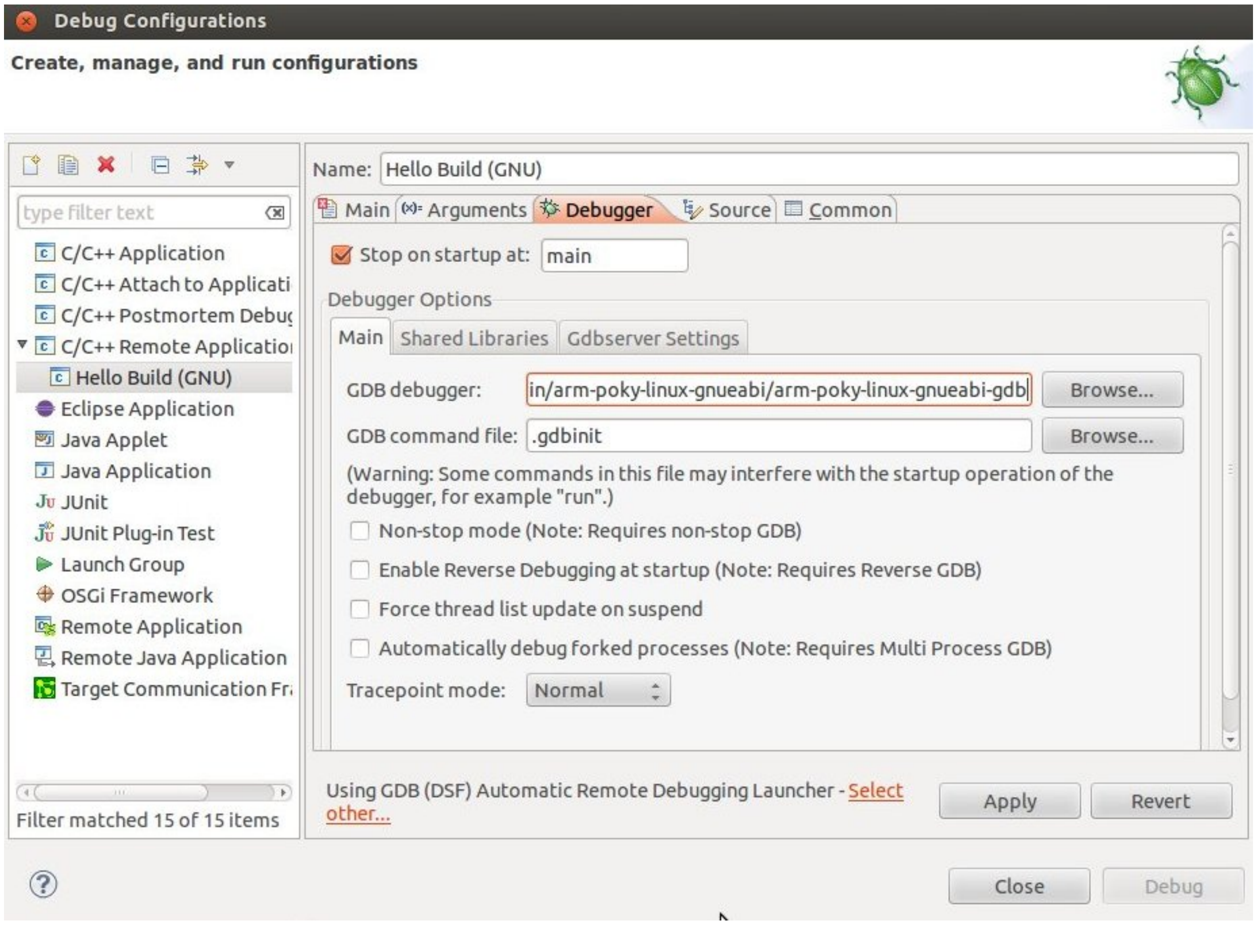


Connection: 192.168.0.10

Remote Absolute File Path for C/C++ Application:

/home/root/HelloWorld

- Select *Debugger* tab



Debug Configurations

Create, manage, and run configurations

Name: Hello Build (GNU)

Stop on startup at: main

Debugger Options

GDB debugger: in/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb

GDB command file: .gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

Non-stop mode (Note: Requires non-stop GDB)

Enable Reverse Debugging at startup (Note: Requires Reverse GDB)

Force thread list update on suspend

Automatically debug forked processes (Note: Requires Multi Process GDB)

Tracepoint mode: Normal

Using GDB (DSF) Automatic Remote Debugging Launcher - Select other...

Filter matched 15 of 15 items

- In GDB Debugger field, insert the filepath of gdb for your toolchain
- In *Debugger* window there is a tab named *Shared Library*, click on it.
- Add the libraries paths *lib* and *usr/lib* of the rootfs (which must be the same used in the target board)
- Click *Debug* to login.
- Accept the debug perspective.

Important: If debug does not work, check on the board if *tcf-agent* is running and *gdbserver* has been installed. You can ignore the message "Cannot access memory at address 0x0".

SDK Architecture

This chapter gives an overview on how the SDK has been composed and where to find the tools on the virtual machine.

SDK

The SDK provided by *Architech* to support Pengwyn is composed by several components, the most important of which are:

- **Yocto**,
- **Eclipse**, and
- **Qt Creator**

Regarding the installation and configuration of these tools, you have many options:

1. get a virtual machine with everything already setup,
2. download a script to setup your Ubuntu machine, or
3. just get the meta-layer and compose your SDK by hand

The method you choose depends on your level of expertise and the results you want to achieve.

If you are new to **Yocto** and/or **Linux**, or simply you don't want to read tons of documentation right now, we suggest you to download and *install the virtual machine* because it is the simplest solution (have a look at *VM content*), everything inside the virtual machine has been thought to work out of the box, plus you will get support.

If performances are your greatest concerns, consider reading Chapter *Create SDK*.

Virtual Machine

The development environment is provided as a virtual disk (to be used by a VirtualBox virtual machine) which you can download from this page:

Important: http://downloads.architechboards.com/sdk/virtual_machine/download.html

Important: Compute the MD5SUM value of the zip file you downloaded and compare it to the golden one you find in the download page.

Uncompress the file, and you will get a *.vdi* file that is our virtual disk image. The environment contains the SDK for all the boards provided by Architech, Pengwyn included.

Download VirtualBox

For being able to use it, you first need to install **VirtualBox** (version 4.2.10 or higher). You can get VirtualBox installer from here:

https://www.virtualbox.org/wiki/Downloads

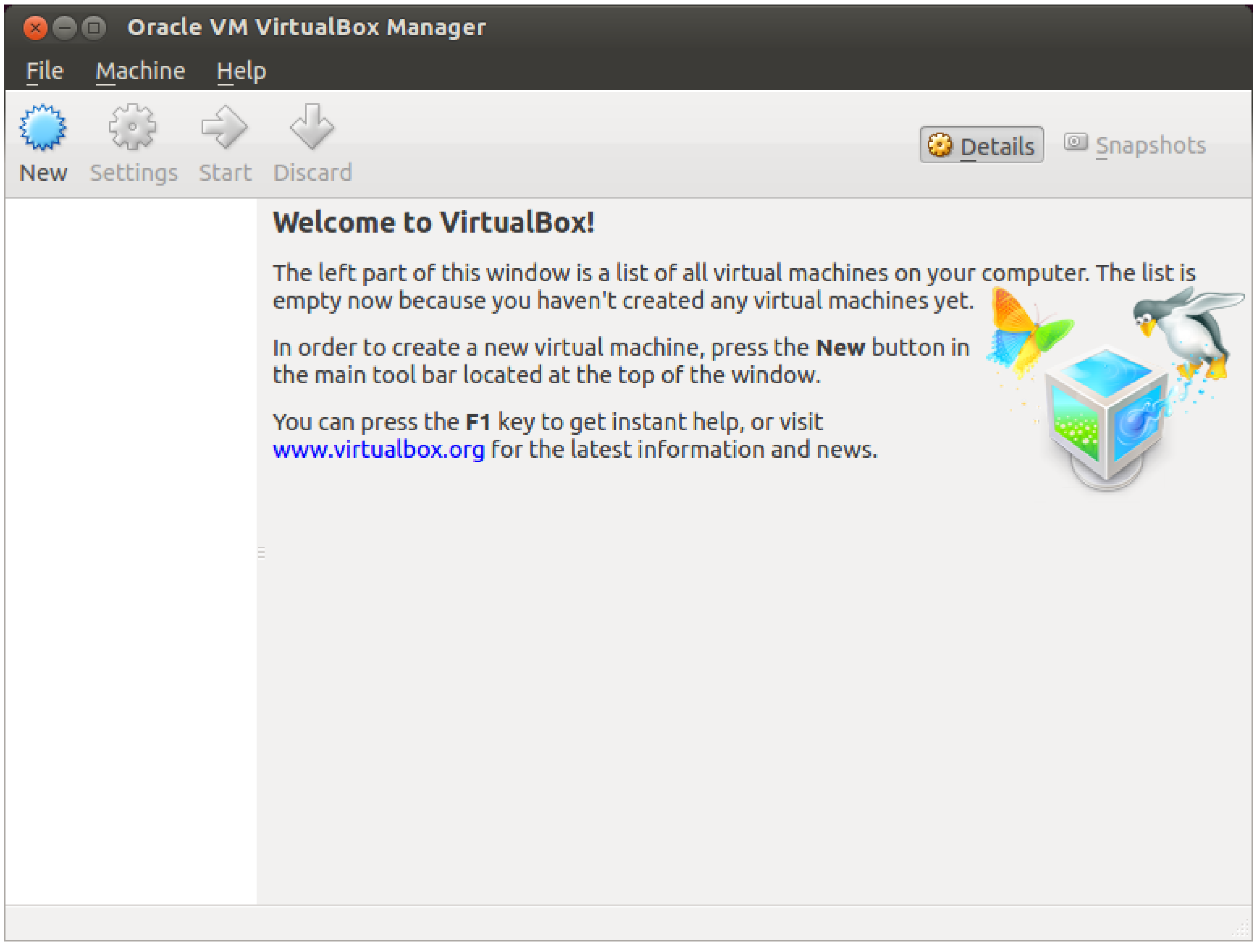
Download the version that suits your host operating system. You need to download and install the **Extension Pack** as well.

Important: Make sure that the extension pack has the same version of VirtualBox.

Install the software with all the default options.

Create a new Virtual Machine

1. Run VirtualBox



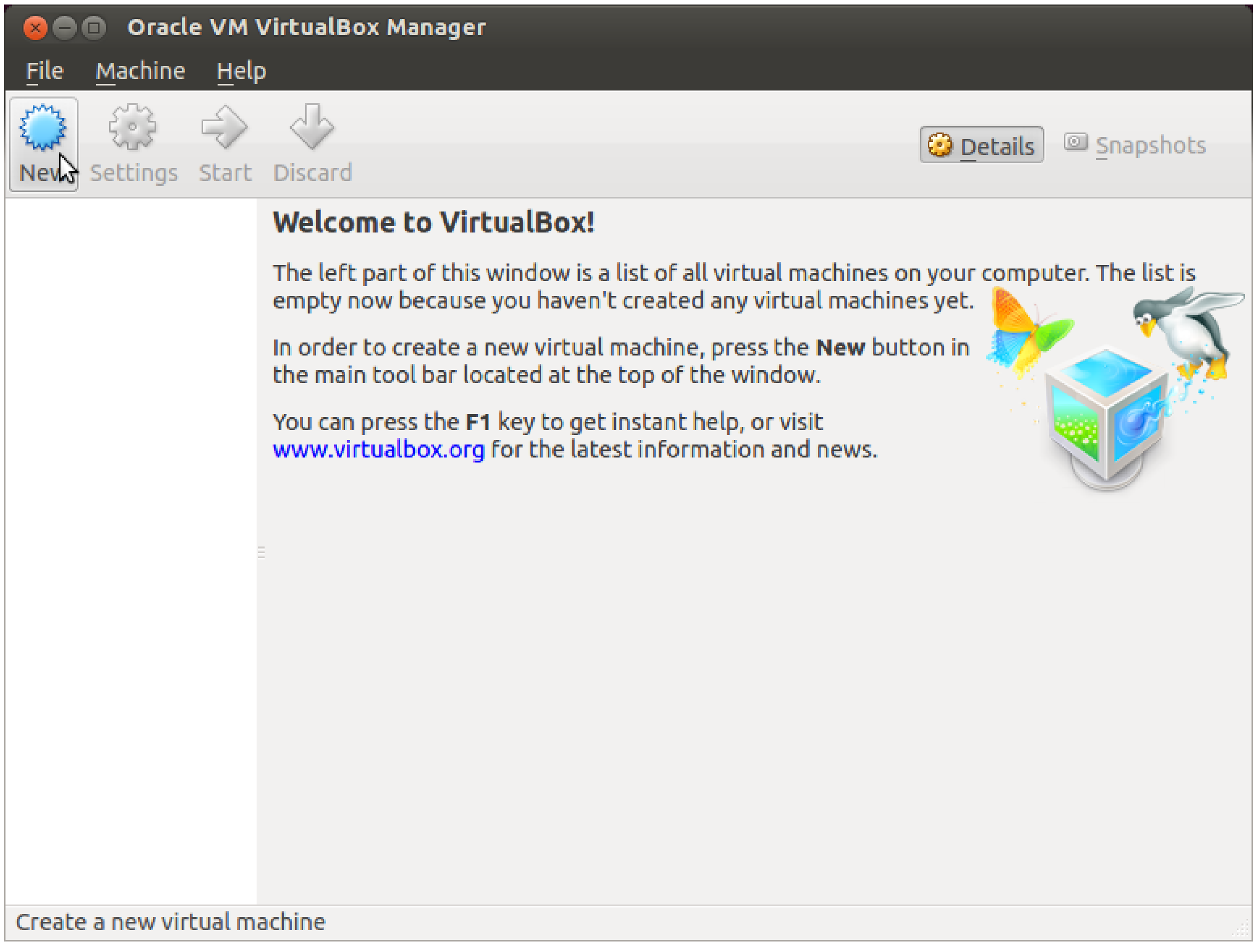
Welcome to VirtualBox!

The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for the latest information and news.

2. Click on *New* button



Welcome to VirtualBox!

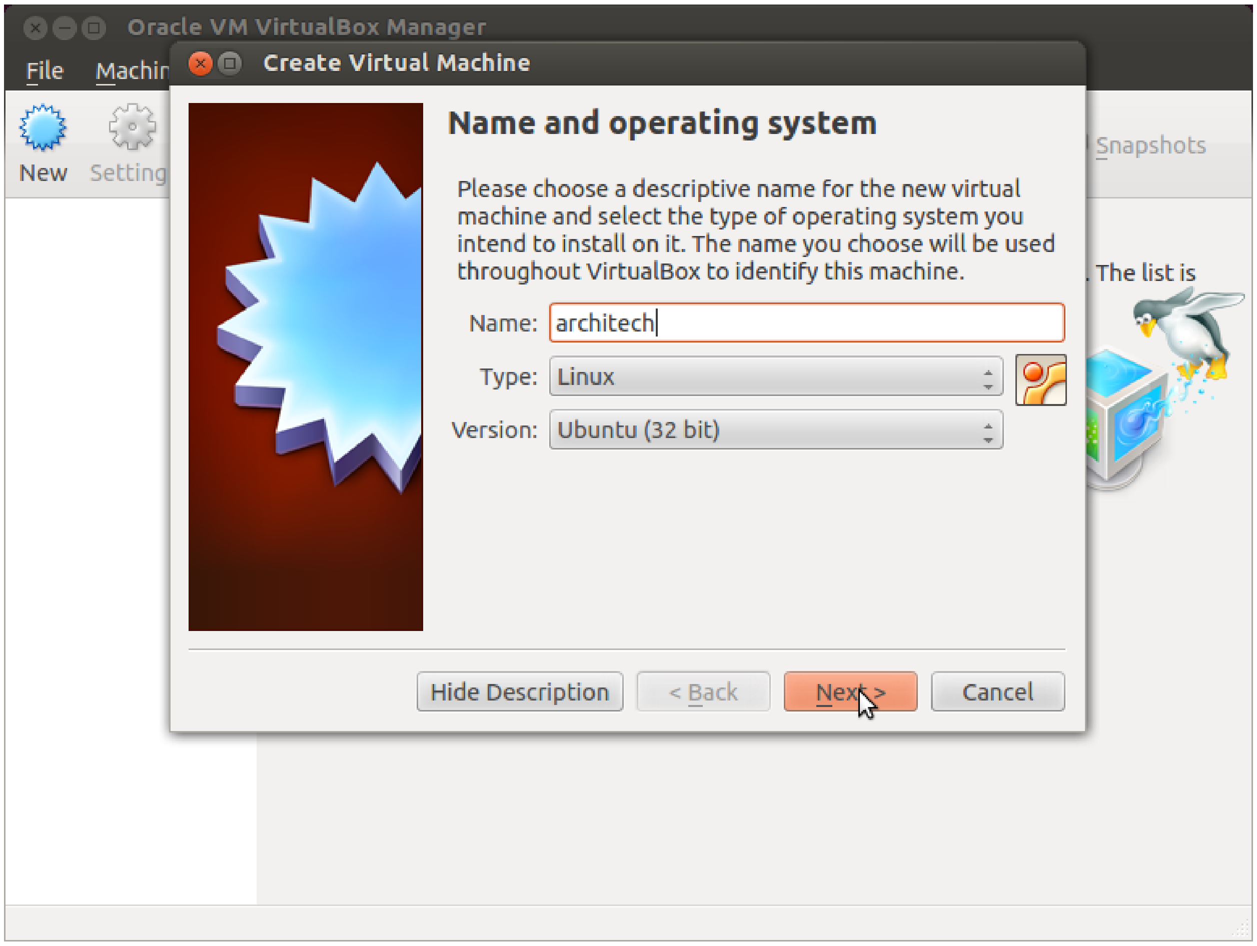
The left part of this window is a list of all virtual machines on your computer. The list is empty now because you haven't created any virtual machines yet.

In order to create a new virtual machine, press the **New** button in the main tool bar located at the top of the window.

You can press the **F1** key to get instant help, or visit www.virtualbox.org for the latest information and news.

Create a new virtual machine

3. Select the name of the virtual machine and the operating system type



Name and operating system

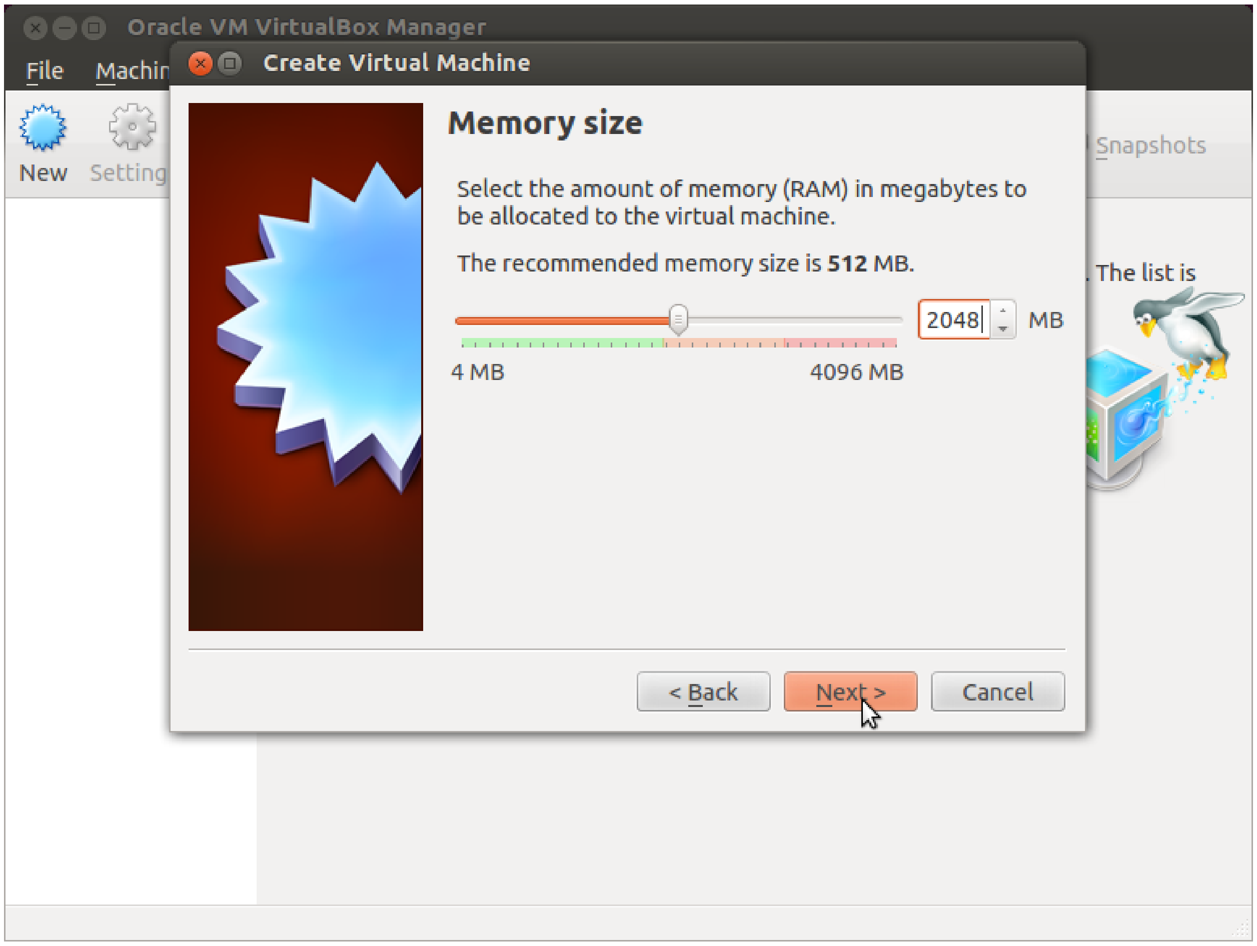
Please choose a descriptive name for the new virtual machine and select the type of operating system you intend to install on it. The name you choose will be used throughout VirtualBox to identify this machine.

Name: architech

Type: Linux

Version: Ubuntu (32 bit)

4. Select the amount of memory you want to give to your new virtual machine



Memory size

Select the amount of memory (RAM) in megabytes to be allocated to the virtual machine.

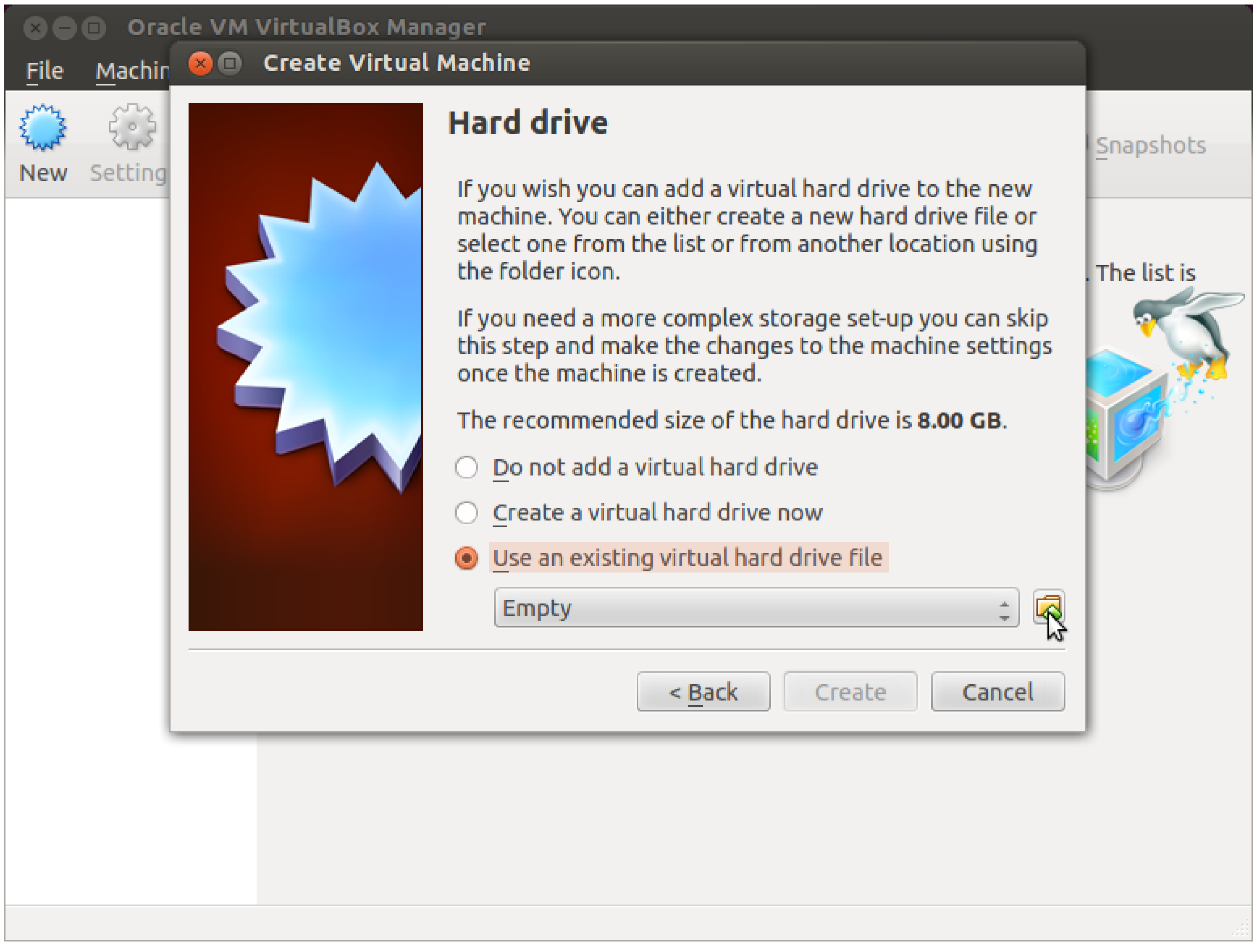
The recommended memory size is **512** MB.

2048 MB

4 MB

4096 MB

5. Make the virtual machine use Architech's virtual disk by pointing to the downloaded file. Than click on *Create*.



Hard drive

If you wish you can add a virtual hard drive to the new machine. You can either create a new hard drive file or select one from the list or from another location using the folder icon.

If you need a more complex storage set-up you can skip this step and make the changes to the machine settings once the machine is created.

The recommended size of the hard drive is **8.00 GB**.

- Do not add a virtual hard drive
- Create a virtual hard drive now
- [✓ Use an existing virtual hard drive file]

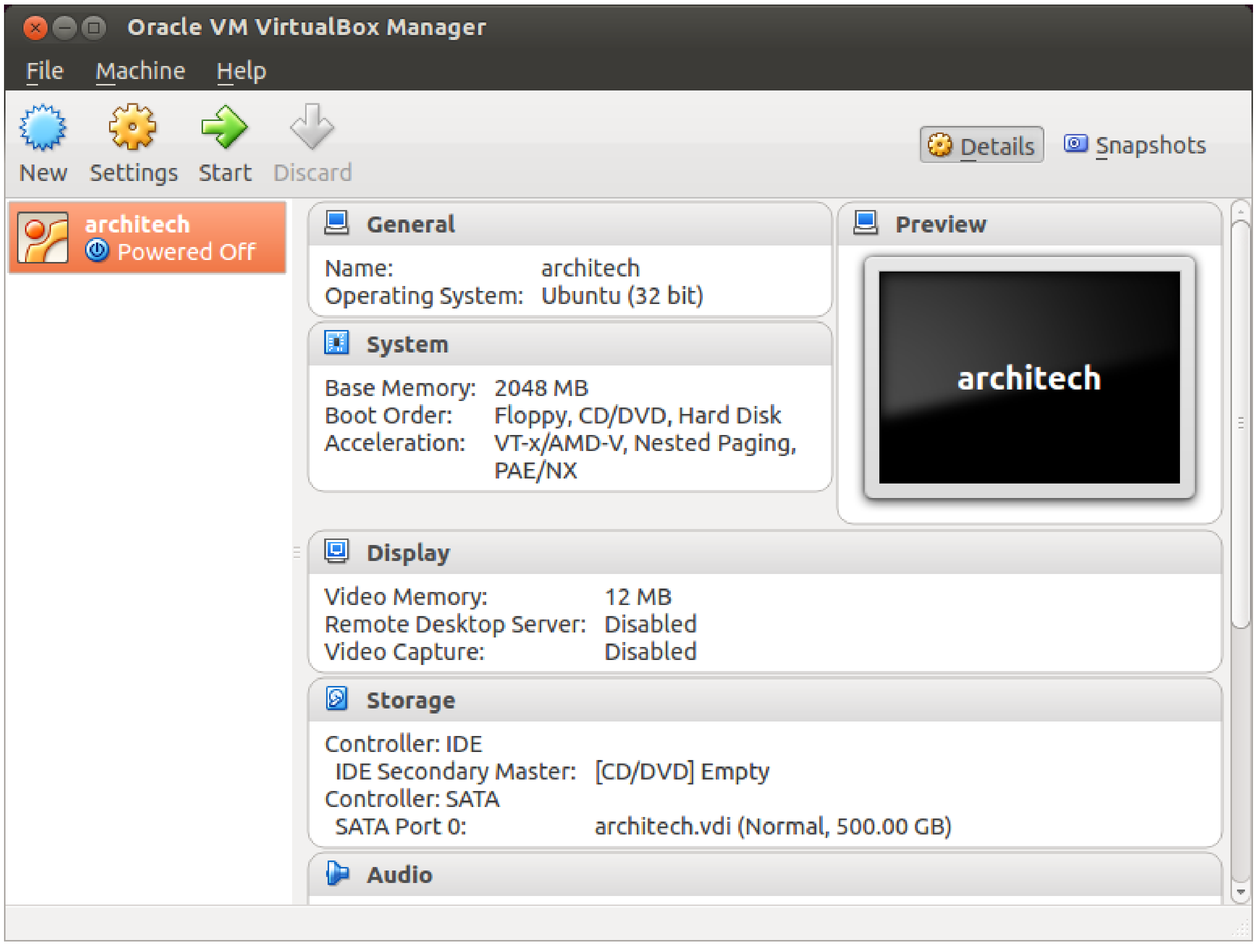
Empty

Setup the network

We need to setup a port forwarding rule to let you (later) use the virtual machine as a local repository of packages.

Note: The virtual machine must be off

1. Select Architech's virtual machine from the list of virtual machines



General

Name: architech
Operating System: Ubuntu (32 bit)

System

Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Display

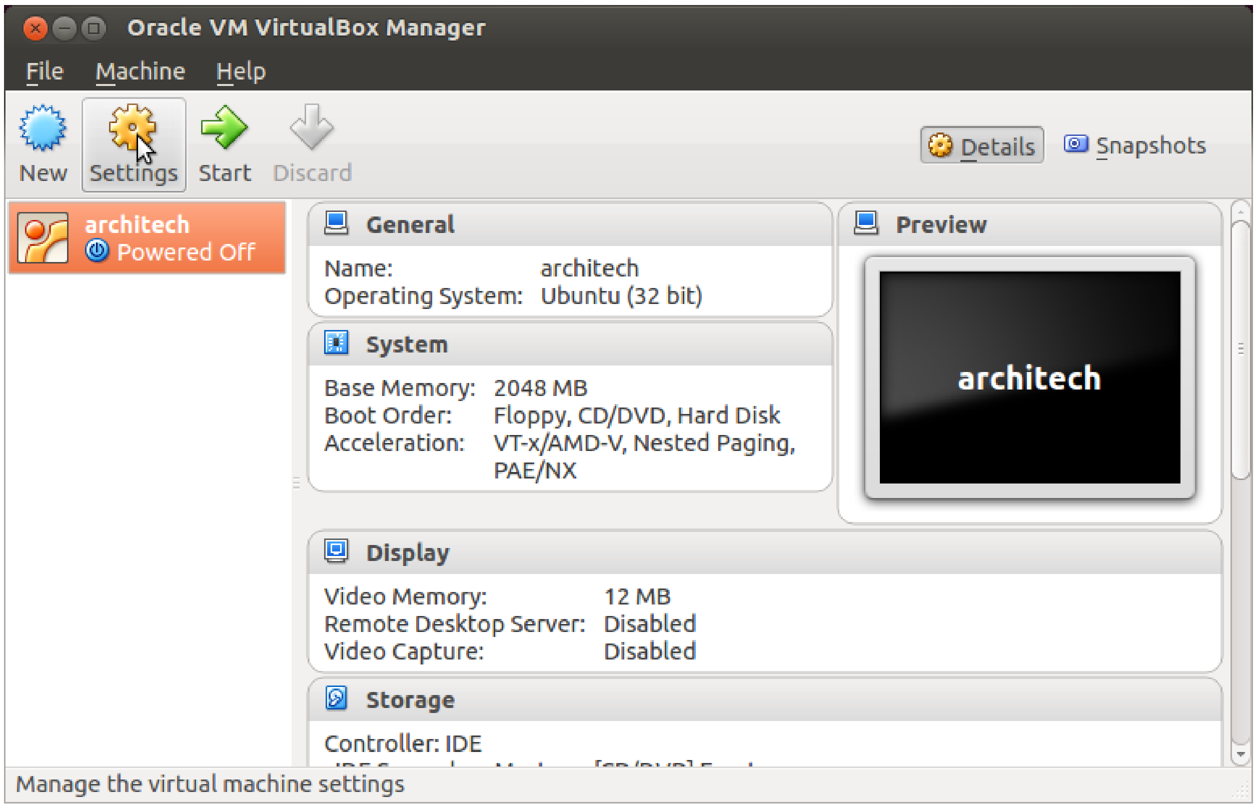
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: IDE
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: architech.vdi (Normal, 500.00 GB)

Audio

2. Click on *Settings*



Name: architech
Operating System: Ubuntu (32 bit)

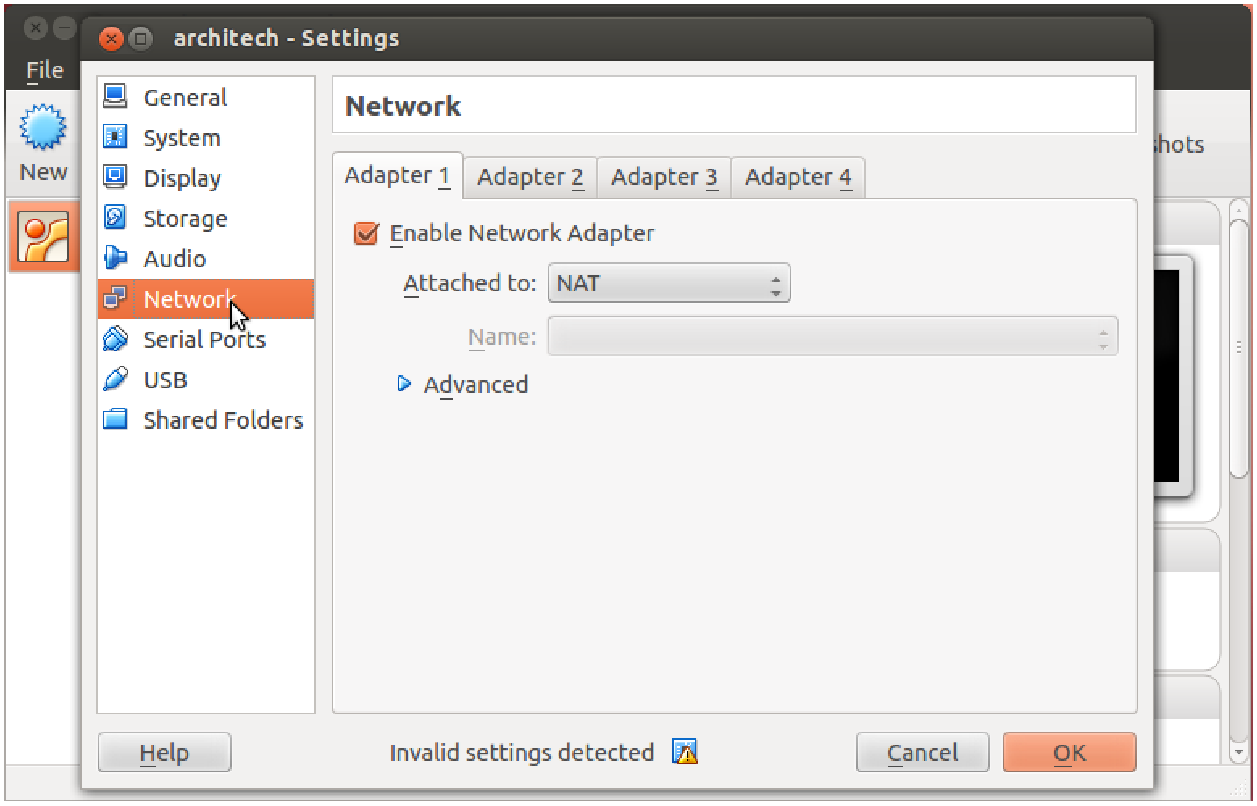
Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Controller: IDE

Manage the virtual machine settings

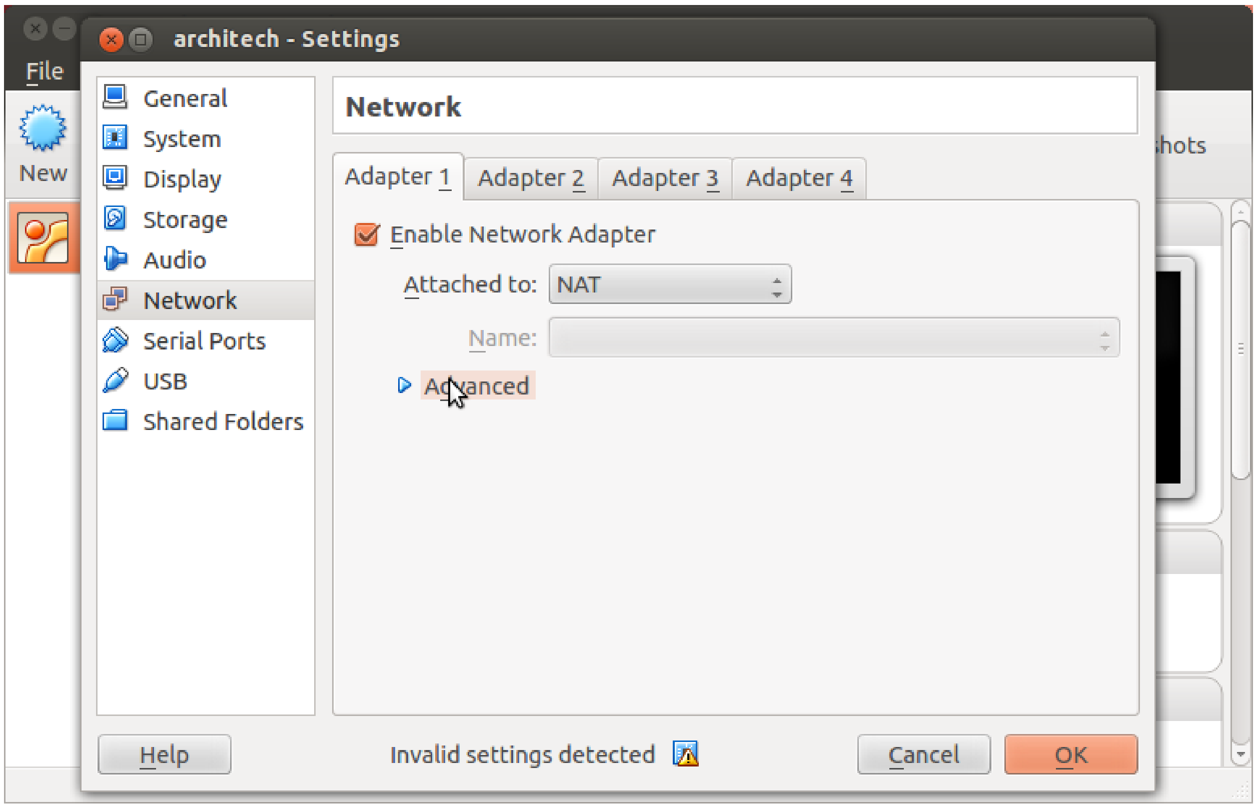
3. Select *Network*



☑ Enable Network Adapter
Attached to: NAT
Name:

Invalid settings detected

4. Expand *Advanced* of *Adapter 1*



Network

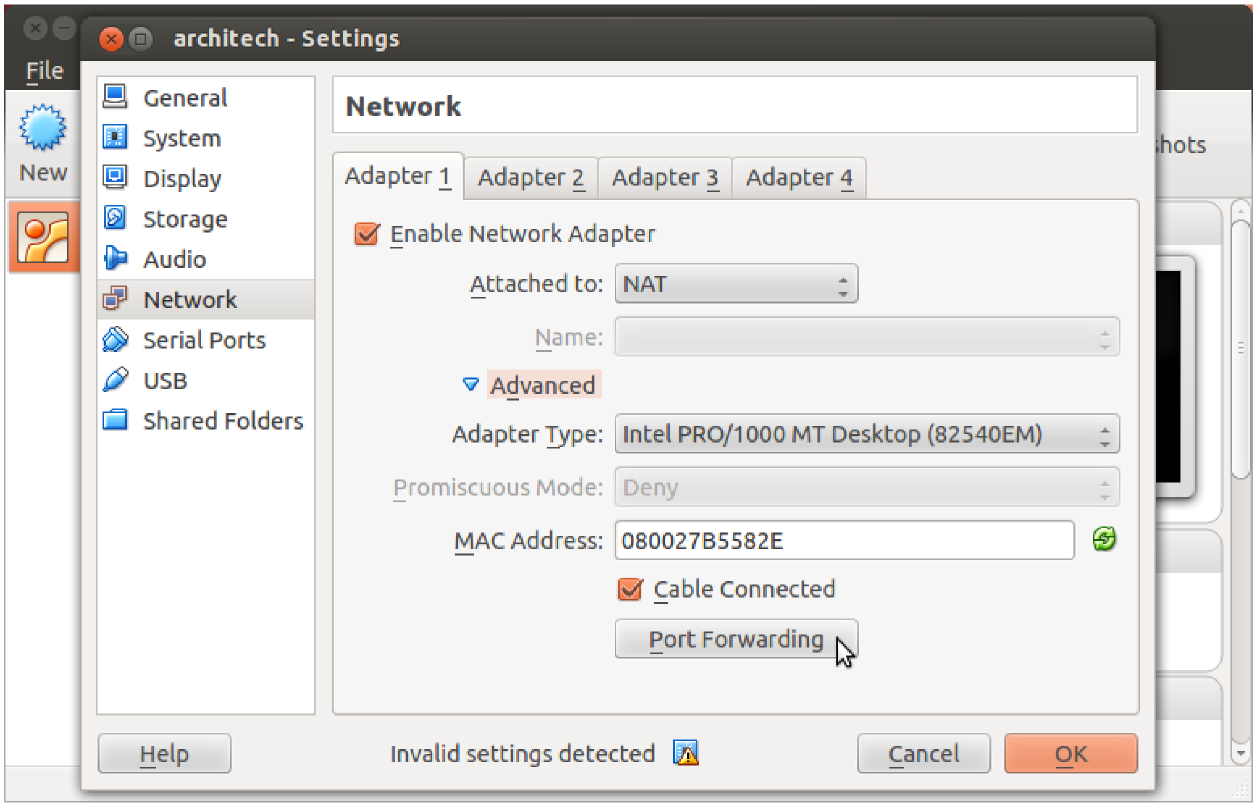
Enable Network Adapter

Attached to: NAT

Name:

Invalid settings detected

5. Click on *Port Forwarding*



Network

Enable Network Adapter

Attached to: NAT

Name:

Adapter Type: Intel PRO/1000 MT Desktop (82540EM)

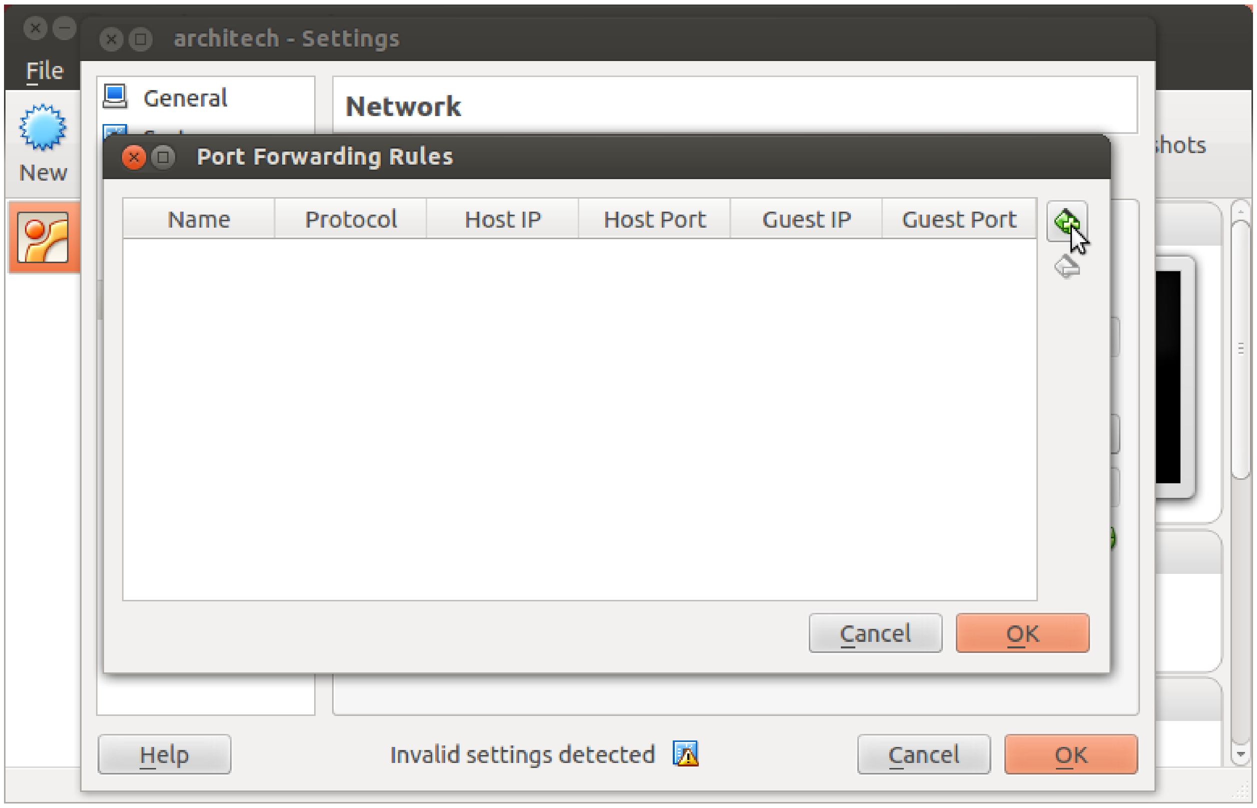
Promiscuous Mode: Deny

MAC Address: 080027B5582E

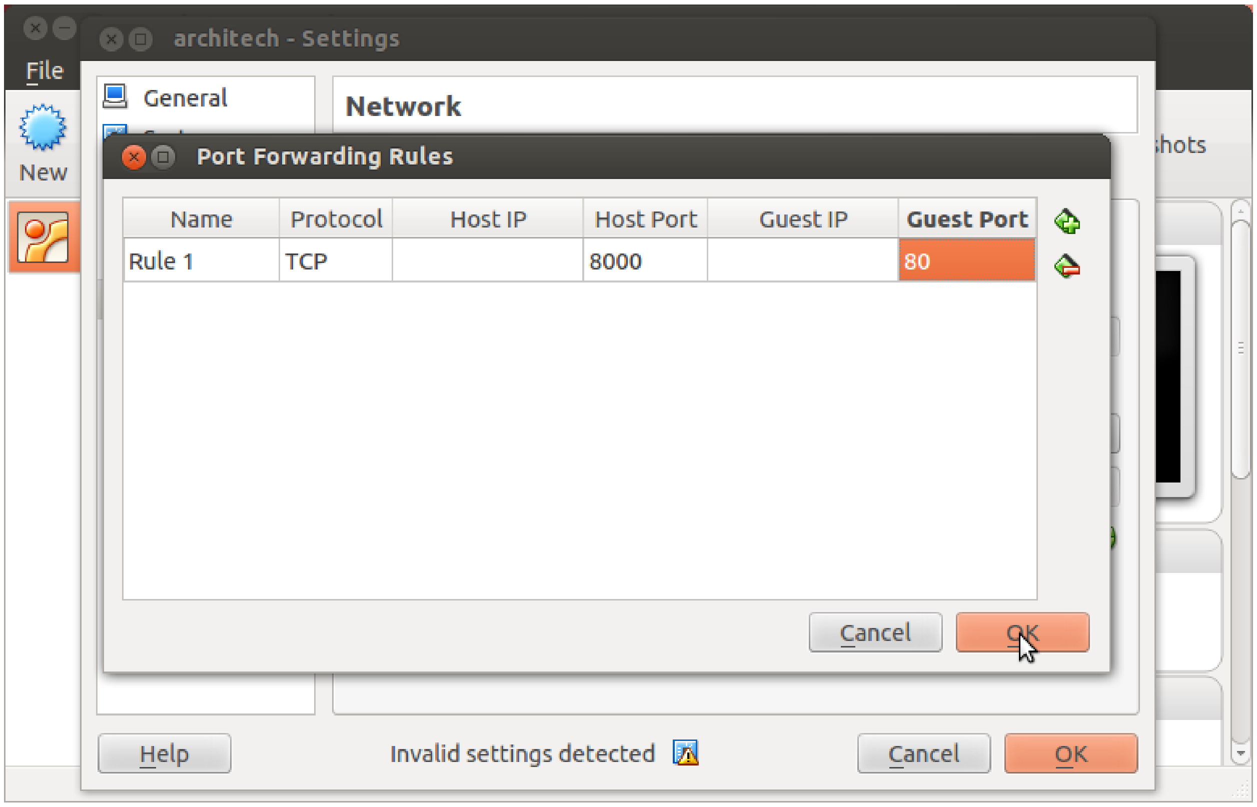
Cable Connected

Invalid settings detected

6. Add a new *rule*



7. Configure the *rule*



Name	Protocol	Host IP	Host Port	Guest IP	Guest Port
Rule 1	TCP		8000		80

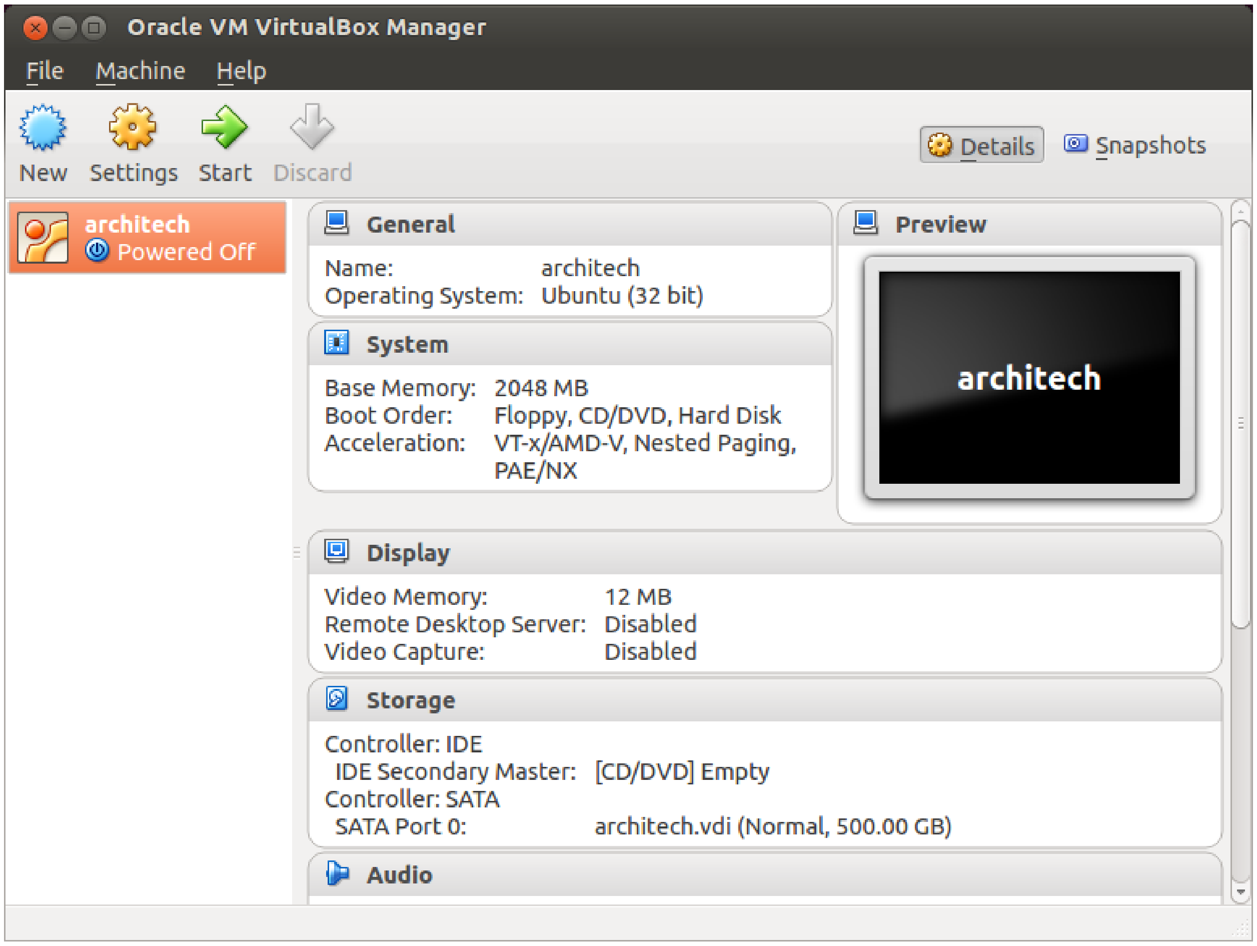
8. Click on *Ok*

Customize the number of processors

Building an entire system from the ground up is a business that can take up to several hours. To improve the performances of the overall build process, you can, if your computer has enough resources, assign more than one processor to the virtual machine.

Note: The virtual machine must be off

1. Select Architech's virtual machine from the list of virtual machines



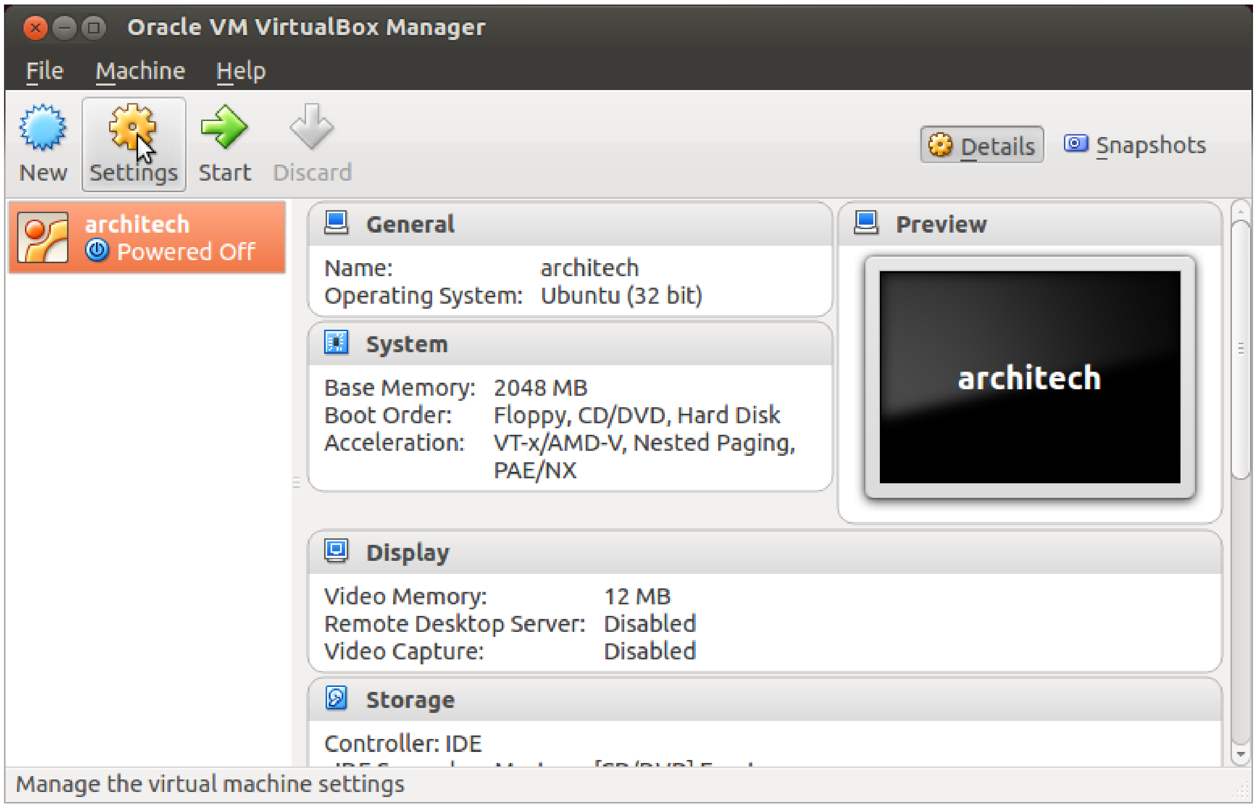
Name: architech
Operating System: Ubuntu (32 bit)

Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Controller: IDE
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: architech.vdi (Normal, 500.00 GB)

2. Click on *Settings*



Name: architech
Operating System: Ubuntu (32 bit)

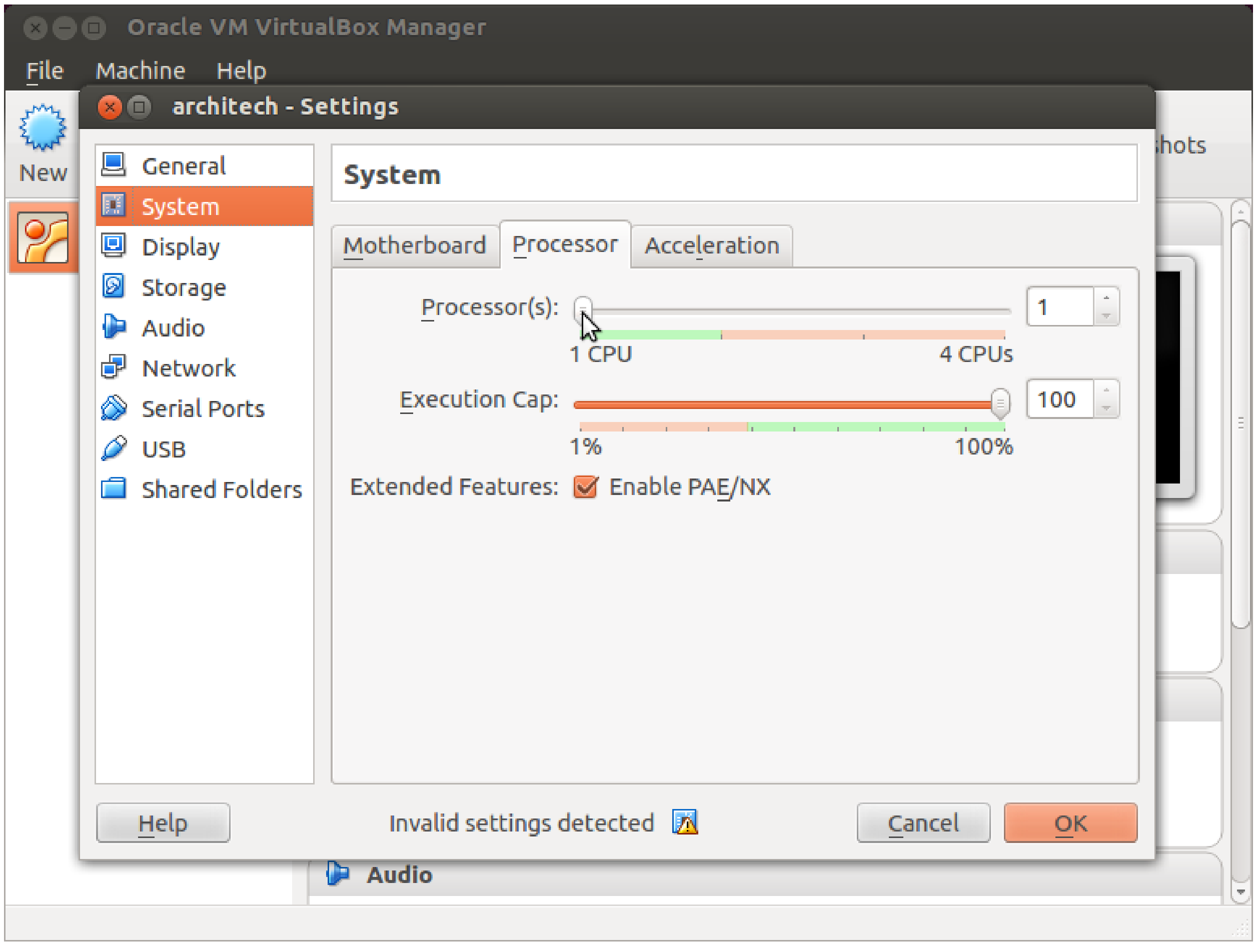
Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Controller: IDE

Manage the virtual machine settings

3. Select *System*
4. Select *Processor*
5. Assign the number of processors you wish to assign to the virtual machine

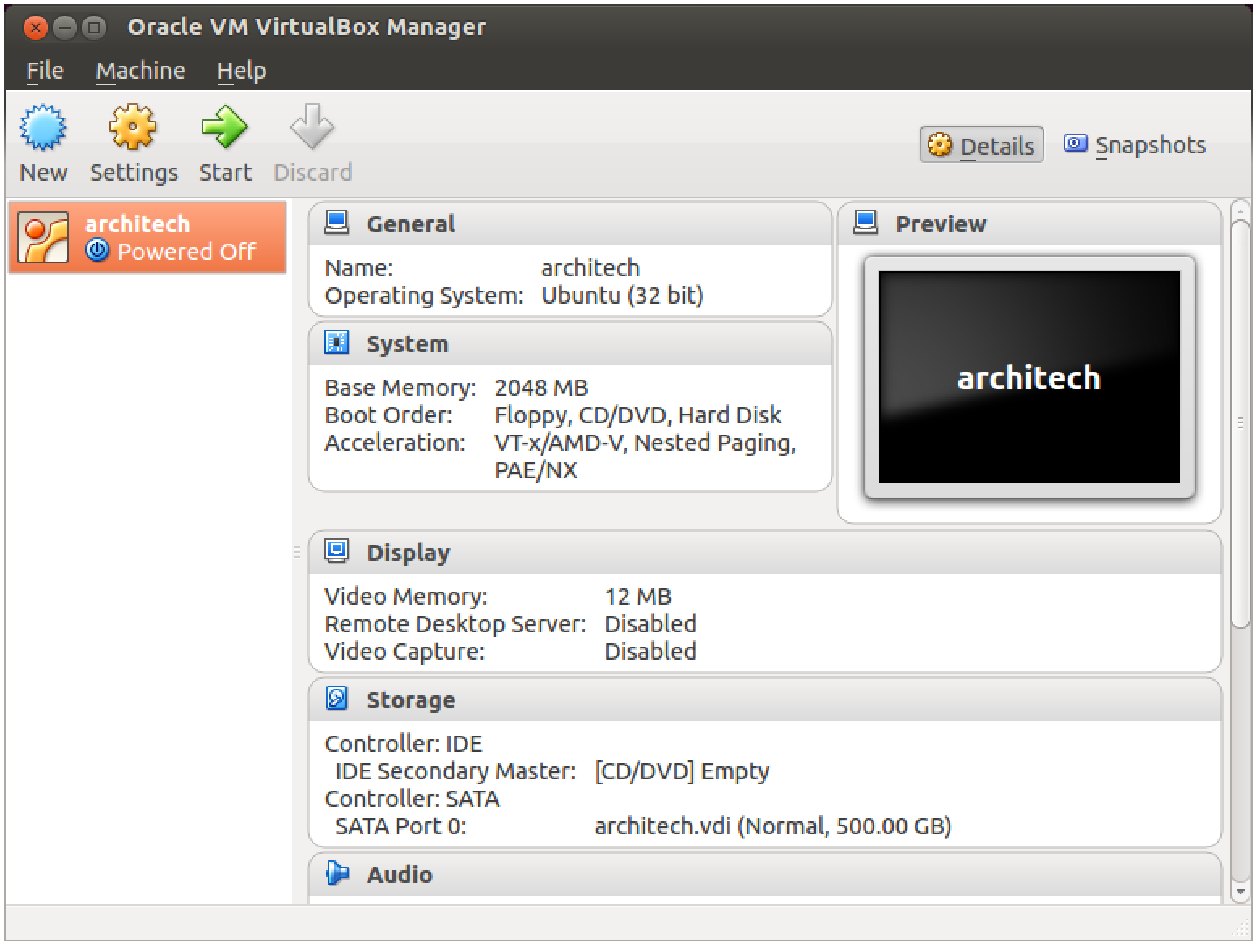


Create a shared folder

A shared folder is way for host and guest operating systems to exchange files by means of the file system. You need to choose a directory on your host operating system to share with the guest operating system.

Note: The virtual machine must be off

1. Select Architech's virtual machine from the list of virtual machines



General

Name: architech
Operating System: Ubuntu (32 bit)

System

Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Display

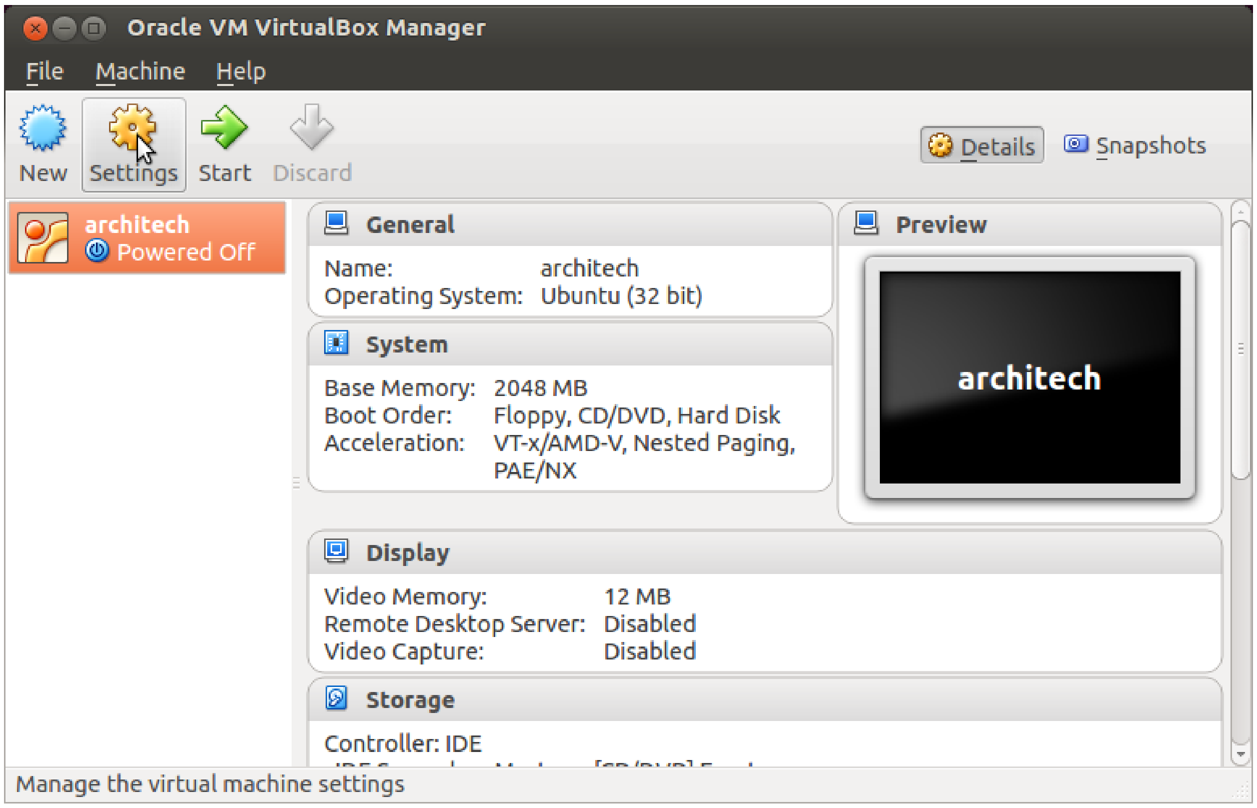
Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Storage

Controller: IDE
IDE Secondary Master: [CD/DVD] Empty
Controller: SATA
SATA Port 0: architech.vdi (Normal, 500.00 GB)

Audio

2. Click on *Settings*



Name: architech
Operating System: Ubuntu (32 bit)

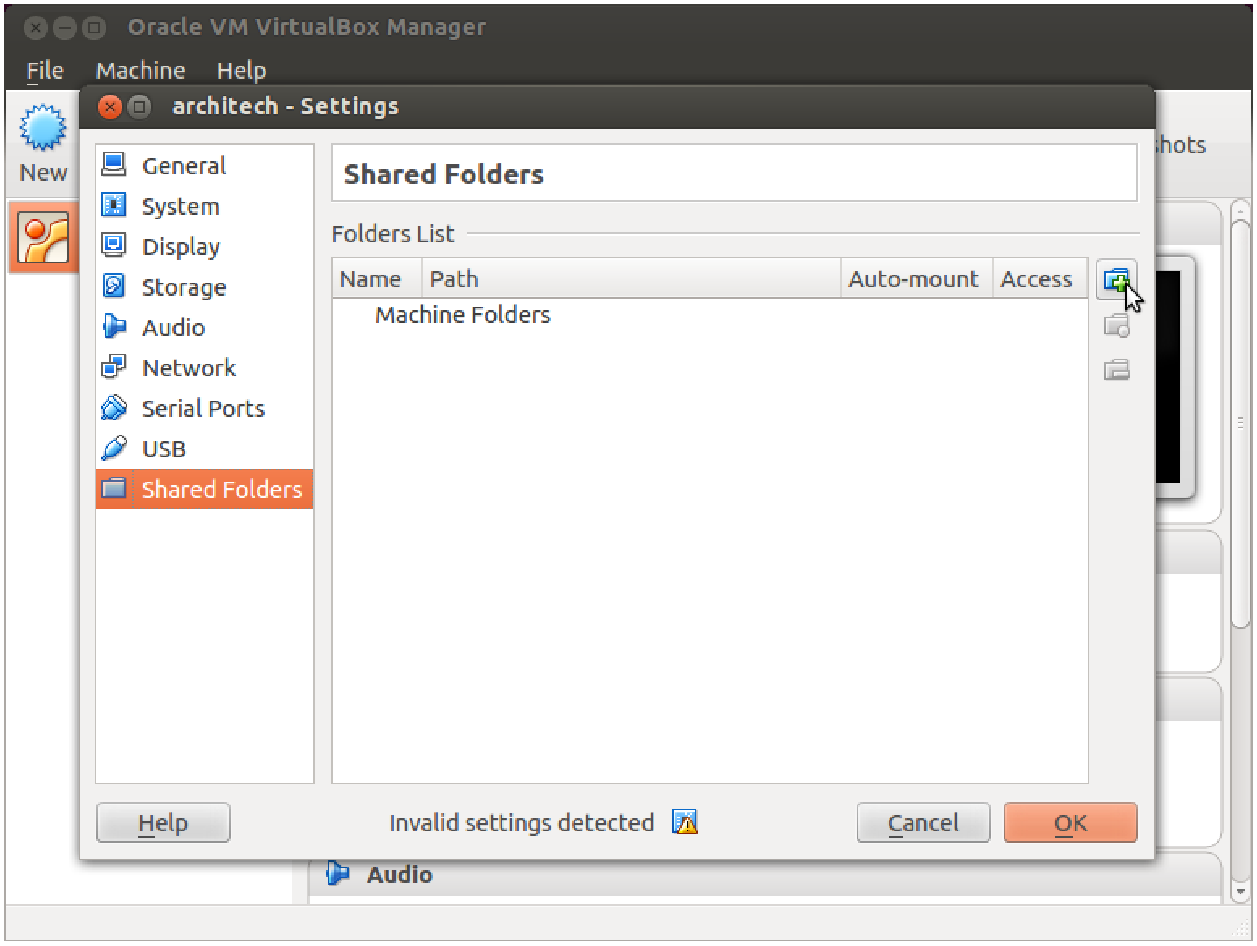
Base Memory: 2048 MB
Boot Order: Floppy, CD/DVD, Hard Disk
Acceleration: VT-x/AMD-V, Nested Paging, PAE/NX

Video Memory: 12 MB
Remote Desktop Server: Disabled
Video Capture: Disabled

Controller: IDE

Manage the virtual machine settings

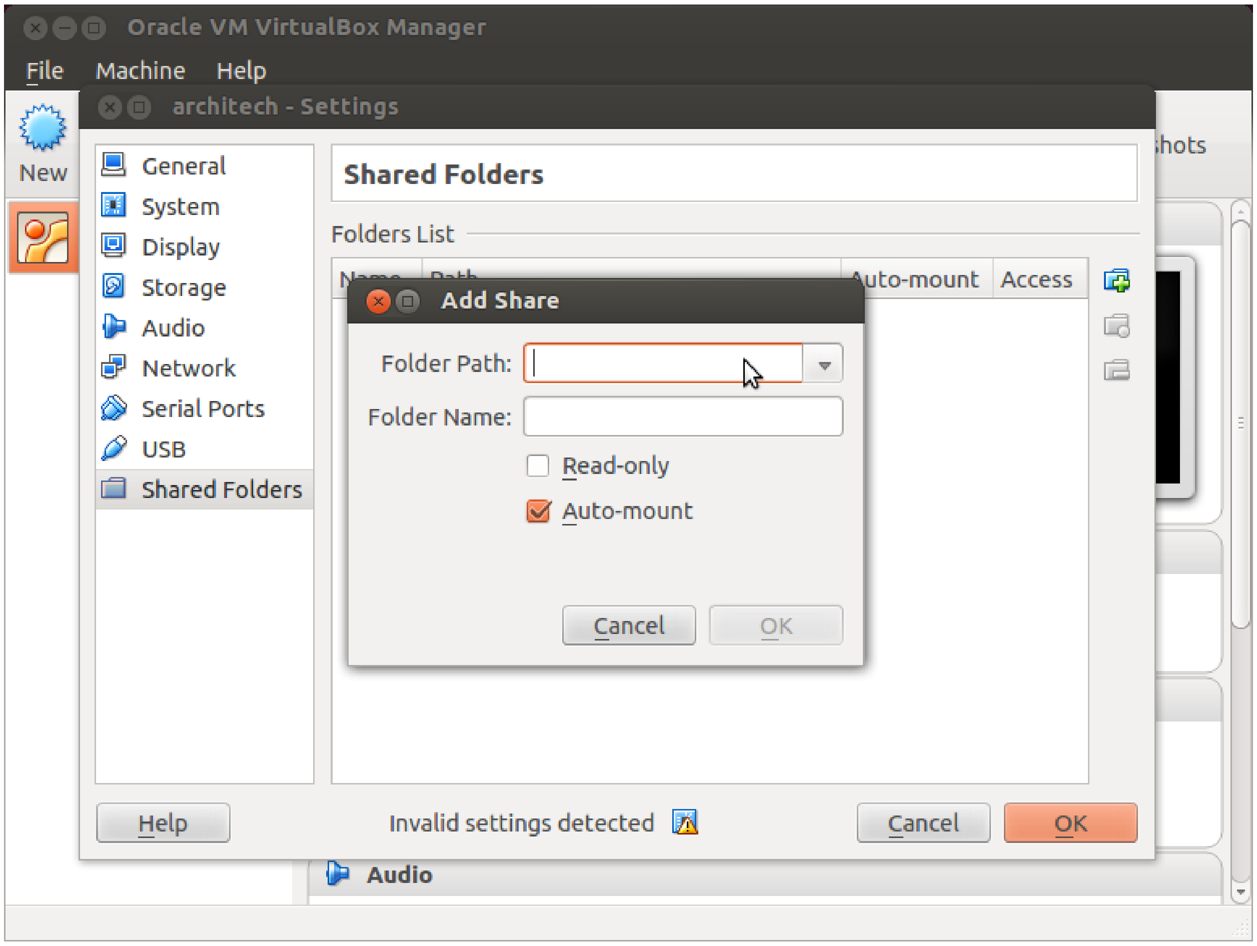
3. Select *Shared Folders*
4. Add a new shared folder



Shared Folders

Invalid settings detected

5. Choose a directory to share on your host machine. Make sure *Auto-mount* is selected.

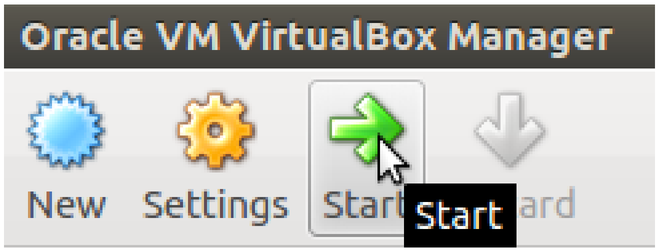


Once the virtual machine has been booted, the shared folder will be mounted under */media/* directory inside the virtual machine.

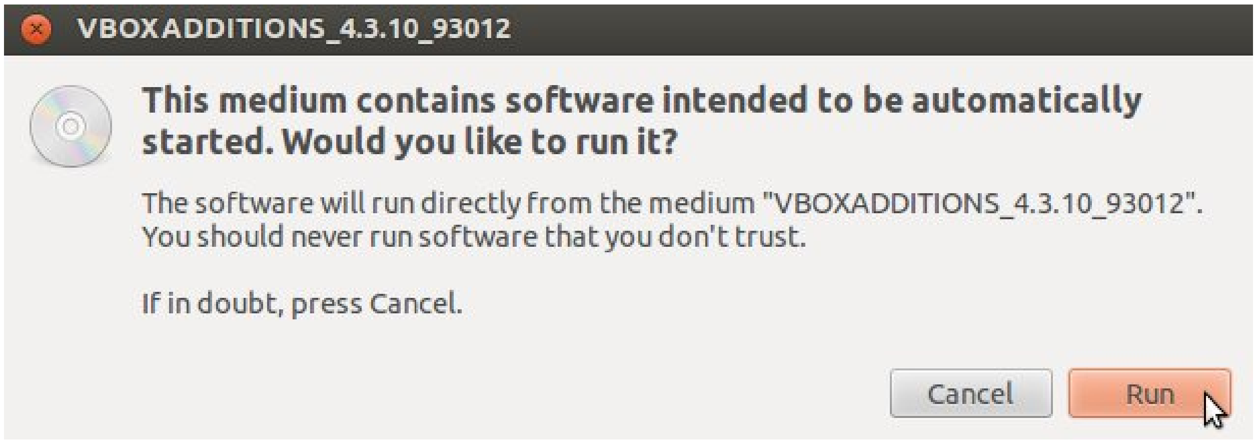
Install VBox Additions

The VBox addictions add functionalities to the virtual machine such as better graphic driver and more. It is already installed in the SDK but is important re-install it to configuring correctly the virtual machine with your operating system.

1. Starts the virtual machine



2. Click on the virtual box menu to the voice *Devices* and select *Insert Guest Additions CD Images...*. A message box will appear at the start of the installation, click on *run* button

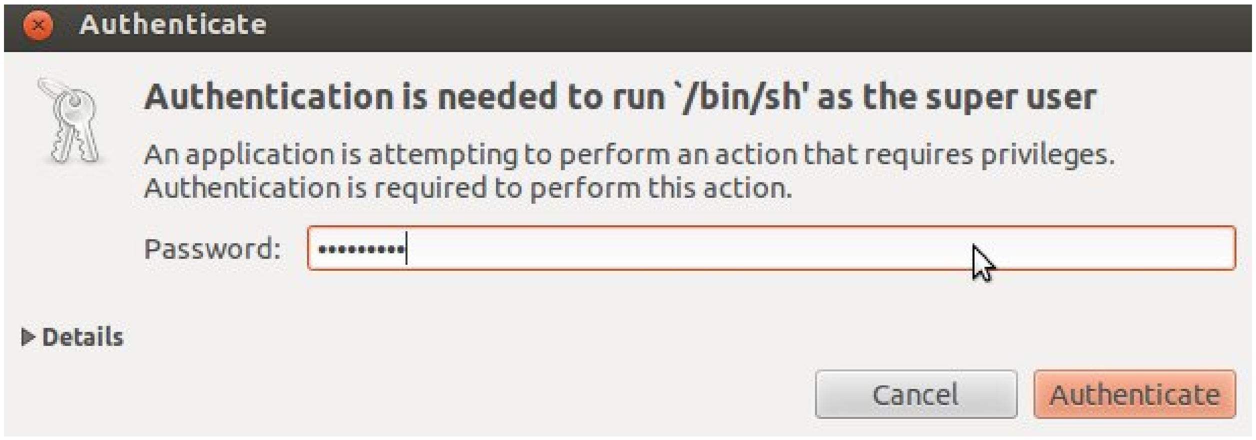

VBOXADDITIONS_4.3.10_93012

This medium contains software intended to be automatically started. Would you like to run it?

The software will run directly from the medium "VBOXADDITIONS_4.3.10_93012". You should never run software that you don't trust.

If in doubt, press Cancel.

4. To proceed are required admin privileges, so insert the password *architech* when asked

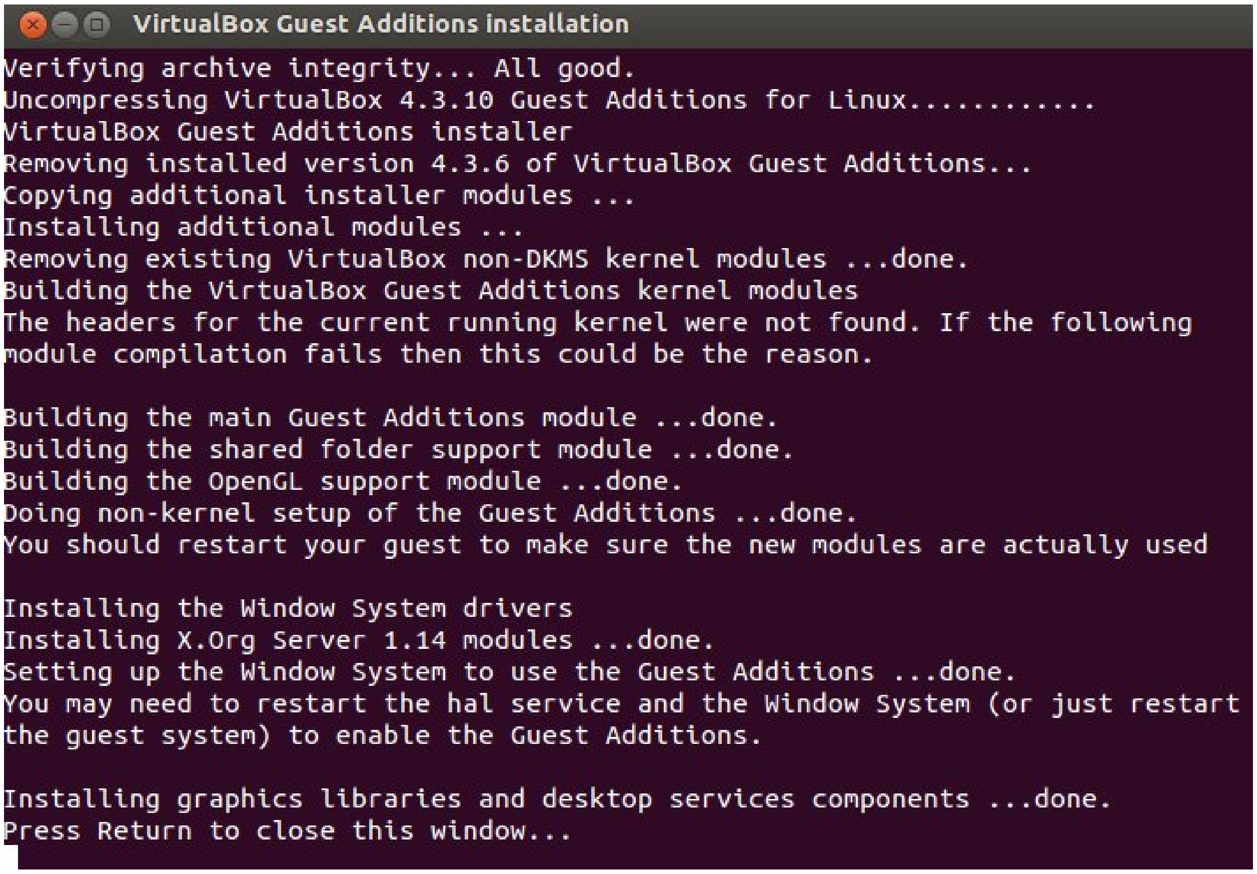

Authenticate

Authentication is needed to run `/bin/sh' as the super user

An application is attempting to perform an action that requires privileges. Authentication is required to perform this action.

Password: ••••••••••

5. Then a terminal will show the installation progress. When finished, press *Enter* key



VirtualBox Guest Additions installation

```
Verifying archive integrity... All good.
Uncompressing VirtualBox 4.3.10 Guest Additions for Linux...........
VirtualBox Guest Additions installer
Removing installed version 4.3.6 of VirtualBox Guest Additions...
Copying additional installer modules ...
Installing additional modules ...
Removing existing VirtualBox non-DKMS kernel modules ...done.
Building the VirtualBox Guest Additions kernel modules
The headers for the current running kernel were not found. If the following
module compilation fails then this could be the reason.

Building the main Guest Additions module ...done.
Building the shared folder support module ...done.
Building the OpenGL support module ...done.
Doing non-kernel setup of the Guest Additions ...done.
You should restart your guest to make sure the new modules are actually used

Installing the Window System drivers
Installing X.Org Server 1.14 modules ...done.
Setting up the Window System to use the Guest Additions ...done.
You may need to restart the hal service and the Window System (or just restart
the guest system) to enable the Guest Additions.

Installing graphics libraries and desktop services components ...done.
Press Return to close this window...
```

6. Before to use the SDK, it is required reboot the virtual machine

VM content

The virtual machine provided by Architech contains:

- A splash screen, used to easily interact with the boards tools
- Yocto/OpenEmbedded toolchain to build BSPs and file systems
- A cross-toolchain (derived from Yocto/OpenEmbedded) for all the boards
- Eclipse, installed and configured
- Qt creator, installed and configured

All the aforementioned tools are installed under directory **/home/architech/architech_sdk**, its sub-directories main layout is the following:

pengwyn directory contains all the tools composing the ArchiTech SDK for Pengwyn board, along with all the information needed by the splash screen application. In particular:

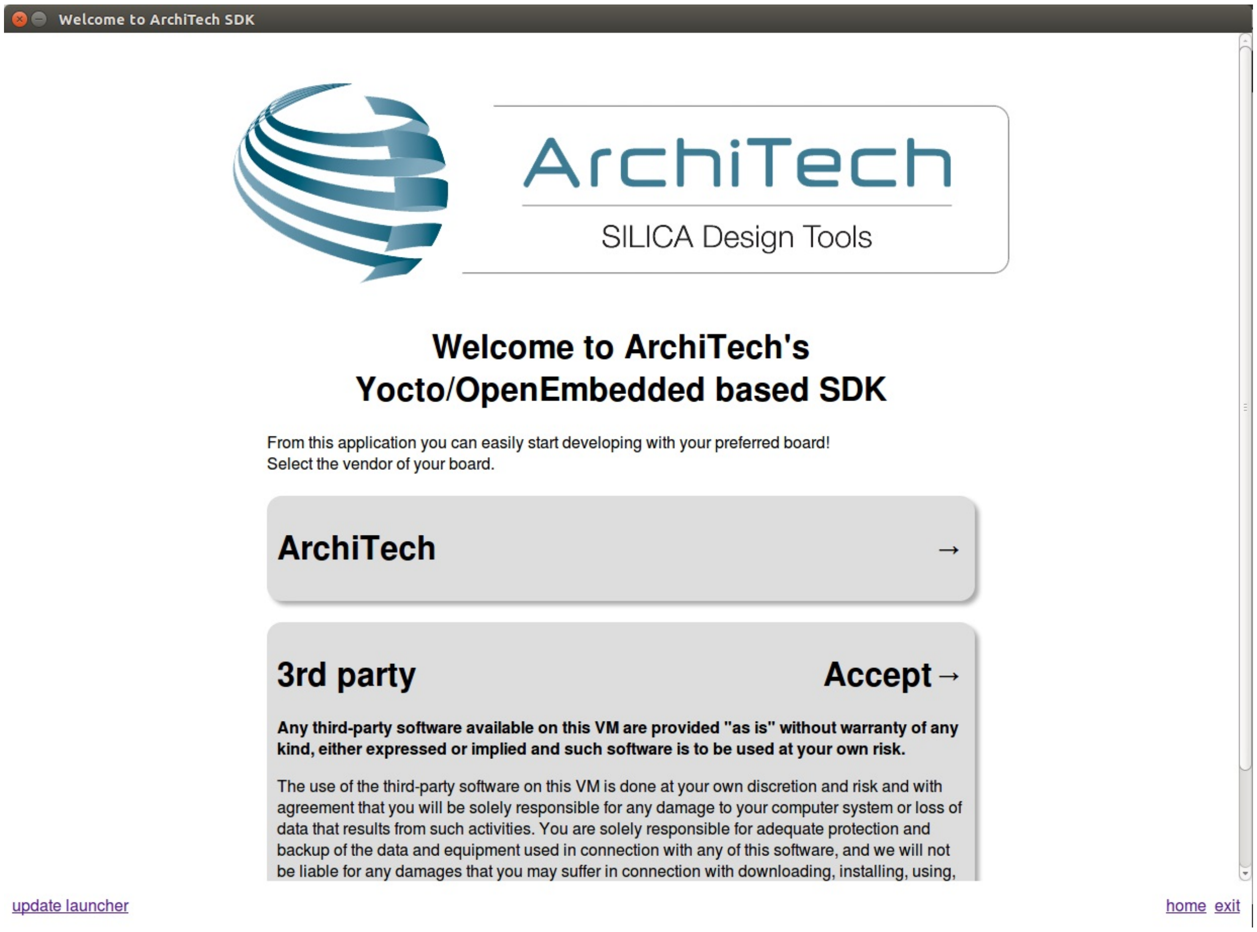
- *eclipse* directory is where Eclipse IDE has been installed
- *qtcreator* contains the installation of Qt Creator IDE
- *splashscreen* directory contains information and scripts used by the splash screen application,
- *sysroot* is supposed to contain the file system you want to compile against,
- *toolchain* is where the cross-toolchain has been installed installed

- *workspace* contains the the workspaces for Eclipse and Qt Creator IDEs
- *yocto* is where you find all the meta-layers Pengwyn requires, along with Poky and the build directory

Splash screen

The splash screen application has been designed to facilitate the access to the boards tools. It can be opened by clicking on its *Desktop* icon.

Once started, you can can choose if you want to work with Architech's boards or with partners' ones. For Pengwyn, choose **ArchiTech**.



Welcome to ArchiTech's
Yocto/OpenEmbedded based SDK

From this application you can easily start developing with your preferred board!
Select the vendor of your board.

ArchiTech →

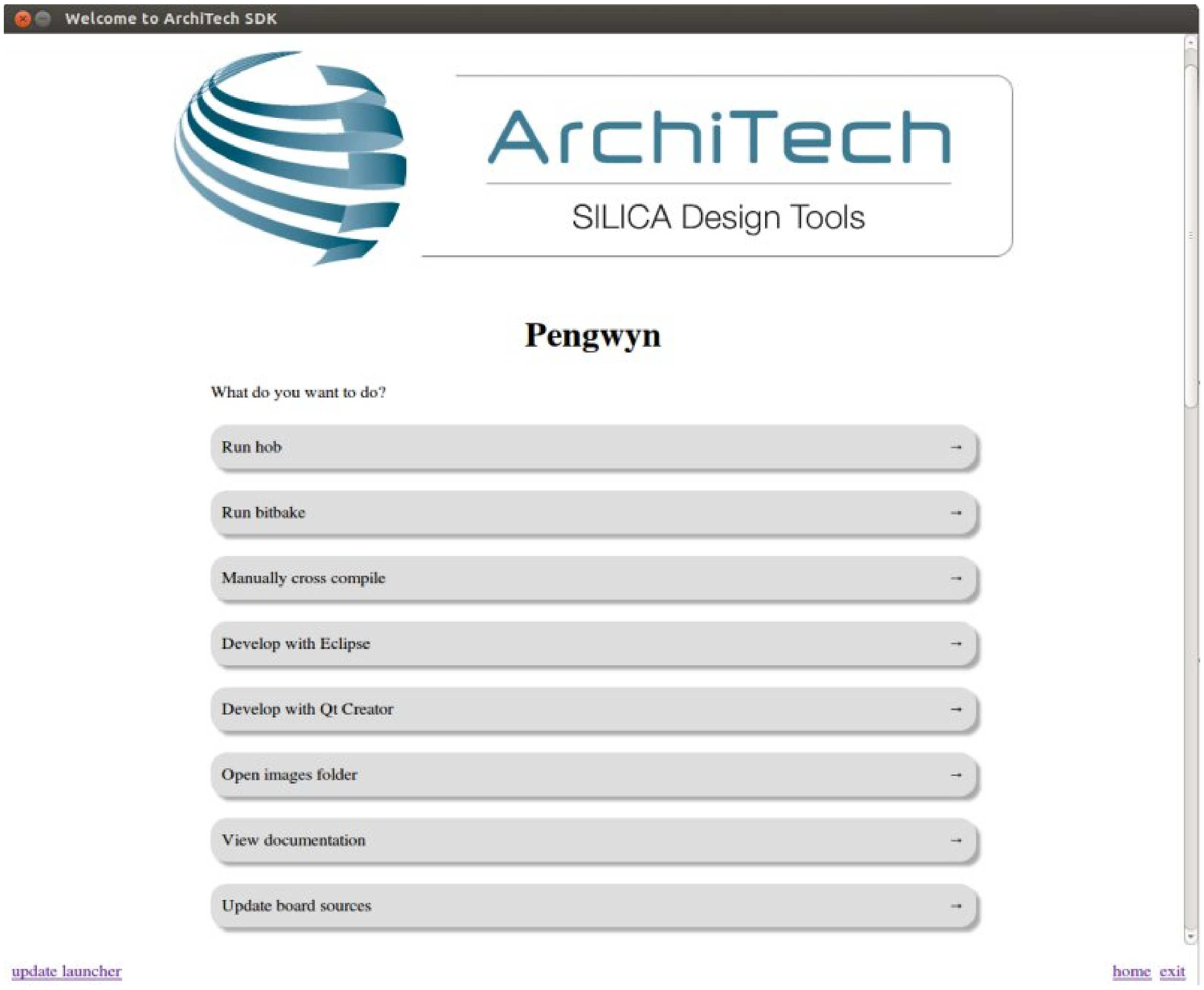
3rd party Accept→

Any third-party software available on this VM are provided "as is" without warranty of any kind, either expressed or implied and such software is to be used at your own risk.

The use of the third-party software on this VM is done at your own discretion and risk and with agreement that you will be solely responsible for any damage to your computer system or loss of data that results from such activities. You are solely responsible for adequate protection and backup of the data and equipment used in connection with any of this software, and we will not be liable for any damages that you may suffer in connection with downloading, installing, using,

A list of all available Architech's boards will open, select Pengwyn.

A list of actions related to Pengwyn that can be activated will appear.



Create SDK

If you have speed in mind, it is possible to install the SDK on a native Ubuntu machine (other Linux distributions may support this SDK with minor changes but won't be supported). This chapter will guide you on how to clone the entire SDK, to setup the SDK for one board or just **OpenEmbedded/Yocto** for Pengwyn board.

Installation

Architech's Yocto based SDK is built on top of **Ubuntu 12.04 32bit**, hence all the scripts provided are proven to work on such a system.

If you wish to use another distribution/version you might need to change some script option and/or modify the scripts yourself, remember that you won't get any support in doing so.

Install a clone of the virtual machine inside your native machine

To install the same tools you get inside the virtual machine on your native machine you need to download and run a system wide installation script:

where *-g* option asks the script to install and configure a few graphic customization, while *-p* option asks the script to install the required packages on the machine. If you want to install the toolchain on a machine not equal to Ubuntu 12.04 32bit then you may want to read the script, install the required packages by hand, and run it without options. You might need to recompile the Qt application used to render the splashscreen.

At the end of the installation process, you will get the same tools installed within the virtual machine, that is, all the tools necessary to work with Architech's boards.

Install just one board

If you don't want to install the tools for all the boards, you can install just the subset of tools related to Pengwyn:

This script needs the same tools/packages required by *machine_install*

Yocto

If you have launched machine_installer or run_install.sh script, yocto is already installed. The following steps are useful for understood how the sdk works "under the hood".

Installation with repo

The easiest way to setup and keep all the necessary meta-layers in sync with upstream repositories is achieved by means of Google's **repo** tool. The following steps are necessary for a clean installation:

1. Install repo tool, if you already have it go to step 4
2. Make sure directory *~/bin* is included in your *PATH* variable by printing its content
3. If *~/bin* directory is not included, add this line to your *~/.bashrc*
4. Open a new terminal
5. Change the current directory to the directory where you want all the meta-layers to be downloaded into
6. Download the manifest
7. Download the repositories

By the end of the last step, all the necessary meta-layers should be in place, anyway, you still need to edit your **local.conf** and **bblayers.conf** to compile for pengwyn machine and using all the downloaded meta-layers.

Updating with repo

When you want your local repositories to be updated, just:

1. Open a terminal
2. Change the current directory to the directory where you ran repo init
3. Sync your repositories with upstream

Install Yocto by yourself

If you really want to download everything by hand, just clone branch *dizzy* of *meta-pengwyn*:

and have a look at the README file.

To install *Eclipse*, *Qt Creator*, *cross-toolchain*, *NFS*, *TFTP*, etc., read **Yocto/OpenEmbedded** documentation, along with the other tools one.

BSP

The Board Support Package is composed by a set files, patches, recipes, configuration files, etc. This chapter gives you the information you need when you want to customize something, fix a bug, or simply learn how the all thing has been assembled.

U-boot

This chapter explains how to compile the u-boot.

Get the sources

The bootloader used by Pengwyn board is **U-Boot**. If you need to modify the bootloader or to recompile it you have two ways to get the sources:

- use the sources you find in Yocto build directory after having compiled at least once a yocto image, or
- download the official u-boot release, than patch it with the BSP patches for Pengwyn board.

Anyway, we will assume in this guide that u-boot sources will be copied to:

and such directory does not yet exists on your PC. Of course, you are free to choose the path you like the most for u-boot sources, just remember to replace the path used in this guide with your custom path. So, where can we get the sources?

1. From Yocto sources

The first way is based on the sources set up by the Yocto build system. However, it is never advisable to work with the sources in the Yocto build directory, if you really want to modify the source code inside the Yocto environment we strongly suggest to refer to the official Yocto documentation. To avoid messing up Yocto recipes and installation, it is desirable to copy the patched u-boot sources you find in the build directory elsewhere. The directory we are talking about is this one:

Replace:

all over this chapter with your custom build directory path if you are not working with the default SDK build directory.

2. From the official u-boot release

The second way is to get the official U-Boot sources and patch them with Pengwyn BSP patches. Pengwyn board uses U-Boot version 2014.07, which can be downloaded from:

Build U-boot

Patches are in the Yocto meta-layer **meta-pengwyn**. You can use them right away if you are working with the SDK:

However, if you are not working with the official SDK the most general solution to check them out and patch the sources is:

Configuration and board files for Pengwyn board are in:

Suppose you modified something and you wanted to recompile the sources to test your patches, well, you need a cross-toolchain. To use it to compile the bootloader or the operating system kernel run:

then you can run these commands to compile it:

Once the build process completes, you can find *u-boot.img* and *MLO* file inside directory */home/architech/Documents/u-boot*.

Linux Kernel

Like we saw for the *bootloader*, the first thing you need is: sources. Get them from *Bitbake* build directory (if you built the kernel with it) or get them from the Internet.

Bitbake will place the sources under directory:

If you are working with the virtual machine, you will find them under directory:

XXX is a random code assigned by bitbake.
We suggest you to **don't work under Bitbake build directory**, you will pay a speed penalty and you could have troubles syncronizing the all thing. Just copy them some place else and do what you have to do.

If you didn't build them already with *Bitbake* or you just want to do make every step by hand, you can always get them from the Internet by cloning the proper repository and checking out the proper hash commit:

and by properly patching the sources:

However, if you are not working with the official SDK the most general solution to check them out and patch the sources is:

Now that you have the sources, you can start browsing the code from the following files:

For build the kernel source the script to load the proper environment for the cross-toolchain:

and to compile it:

If you omit *-j* parameter, *make* will run one task after the other, if you specify it *make* will parallelize the tasks execution while respecting the dependencies between them. Generally, you will place a value for *-j* parameter corresponding to the double of your processor's cores number, for example, on a quad core machine you will place *-j 8*.

Once the kernel is compiled, the last build to do is the dtb file. This file permits at the boot time to configure the kernel with a specific hardware configuration. So if you are using a touchscreen you will build the *pengwyn-touch.dts* file else if you are using a display with dvi connector will be *pengwyn-dvi.dts* file. In the same directory where you have compiled the kernel launch the command:

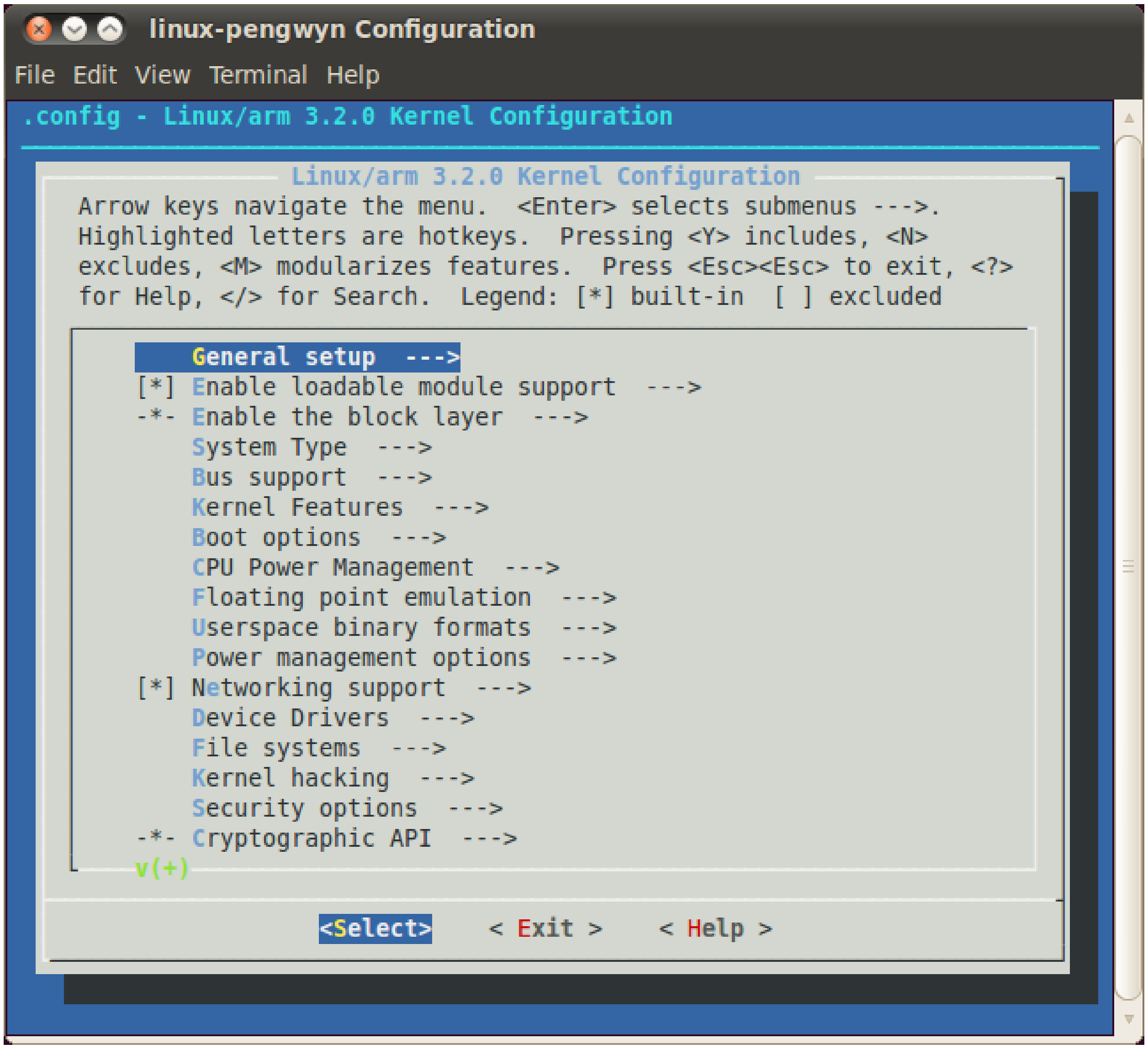
or

By the end of the build process you will get *uImage* under *arch/arm/boot* and *pengwyn-touch.dtb* or *pengwyn-dvi.dtb* under *arch/arm/boot/dts* directories.

Build from bitbake

The most frequent way of customization of the Linux Kernel is to change the .config file that contains the Kernel options. Setup the environment and run:

a new window, like the following one, will pop-up:



```
.config - Linux/arm 3.2.0 Kernel Configuration

                 Linux/arm 3.2.0 Kernel Configuration
  Arrow keys navigate the menu.  <Enter> selects submenus --->.
  Highlighted letters are hotkeys.  Pressing <Y> includes, <N>
  excludes, <M> modularizes features.  Press <Esc><Esc> to exit, <?>
  for Help, </> for Search.  Legend: [*] built-in  [ ] excluded

        General setup  --->
    [*] Enable loadable module support  --->
    -*- Enable the block layer  --->
        System Type  --->
        Bus support  --->
        Kernel Features  --->
        Boot options  --->
        CPU Power Management  --->
        Floating point emulation  --->
        Userspace binary formats  --->
        Power management options  --->
    [*] Networking support  --->
        Device Drivers  --->
        File systems  --->
        Kernel hacking  --->
        Security options  --->
    -*- Cryptographic API  --->
    v(+)

           <Select>    < Exit >    < Help >
```

follow the instructions, save and exit, than you ready to generate your preferred image based on your customized kernel. If you prefer, you can build just the kernel running:

At the end of the build process, the output file (uImage.bin), along with the built kernel modules, will be placed under *tmp/deploy/images/pengwyn/* inside your build directory, so, if you are building your system from the default directory, the destination directory will be */home/architech/architech_sdk/architech/pengwyn/yocto/build/tmp/deploy/images/pengwyn/*.

Meta Layer

A Yocto/OpenEmbedded meta-layer is a directory that contains recipes, configuration files, patches, etc., all needed by *Bitbake* to properly "see" and build a BSP, a distribution, a (set of) package(s), whatever. **meta-pengwyn** is a meta-layer which defines the customizations to make to TI's AM335x BSP and Yocto/OpenEmbedded in order to get a working system, tailor made of Pengwyn.

You can get it with *git*:

The machine name for Pengwyn is **pengwyn**.

The strictly BSP related recipes are located under:

The other recipes are there just to customize other aspects of the system or to offer some facility to help you easily manage some task, for example, working with flash memory or partitions.

Pengwyn is powered by a NAND memory, big enough to place a full featured root file system inside of it. However, you might not be interested in how to place the file system inside of it from the beginning and how to mount and unmount it inside your file system. There is a recipe inside meta-pengwyn, **pengwyn-flash-utils**, that will install three scripts inside the target file system to make the aforementioned tasks easy:

- *pengwyn_to_flash*
- *pengwyn_mount_flash*
- *pengwyn_umount_flash*

pengwyn_to_flash takes as input files, cleans and formats the NAND flash memory, and finally takes the files you gave him to setup the file system. For more information just run:

from Pengwyn shell.

pengwyn_mount_flash lets you mount the flash memory partition inside your filesystem (under */mnt/flash*) without any effort and, likewise, *pengwyn_umount_flash* helps you unmounting the partition.

Remember that to install those scripts inside the target, you need to add **meta-openmbedded/meta-oe** meta layer to your *bblayers.conf* file. If you are working with Architech virtual machine, you don't have to worry about that, everything is already in place.

pengwyn-flash-utils won't be placed by default inside your file system, if you want it you need to add a line like this one to your *local.conf* file

Probably the most comfortable way, at least at the beginning, to build a valid SD card is to use file *.sdcard* that *Bitbake* emits when builds an image. However, *Bitbake* prepares a final iso image to write to the medium without any knowledge of its size. If you write the image on an SD card, for example, the first thing you notice is that the file system does not fit the card.

Root FS

By default, Pengwyn's Yocto/OpenEmbedded SDK will generate an image *<name image>.tar.gz*:

The *.tar.gz* file can be flattened out in your final medium partition (on SD card, flash memory) or on your host development system and used for build purposes with the Yocto Project.

If you want use a new SD card to unpack your image .tar.gz then read the following section else skip it.

How to create the SD card

This section describes the steps to be followed to create a standalone bootable system on **SD card**.

Ensure that the following is available:

- An SD memory card reader/programmer to copy files from the Linux Host. The SD card reader must be accessible from VirtualBox.
- An USB device to read and write a SD card.

Warning: Not all computer built-in readers can be used, use USB connected devices instead.

- An SD card

Warning: Your sd-card will be formatted and all the data contained in it will be lost forever!

- The tool scripts (download link pengwyn-tools-dizzy.tar.bz2) and decompress in a directory. eg:
- A Linux host with fdisk, sfdisk, mkfs.ext3 and mkfs.vfat utilities. If you are running the VM with Ubuntu pre-installed and the environment already in place, all the packages are already available.
- The files *MLO*, *u-boot.img*, *zImage-pengwyn-dvi.dtb*, *zImage*, (optionally) the kernel modules (*modules-3.2.0-rX-pengwyn.tgz*, where X is the revision number for the kernel modules, without any modification to the kernel configuration it should be 0) and the *root file system* with name ending with **-pengwyn.tar.gz** are available inside the images deploy directory (*<build directory>/tmp/deploy/images/pengwyn/*, if your build directory is the default one, then the *deploy directory* is */home/architech/architech_sdk/architech/pengwyn/yocto/build/tmp/deploy/images/pengwyn*).

Note:

The scripts will get the latest files from folder
/home/architech/architech_sdk/architech/pengwyn/yocto/build/tmp/deploy/images/pengwyn.

If you want to specify a custom directory where there are the Yocto images, enter the path directly after the script name, e.g.:
sudo ./fast-create-sdcard.sh /home/architech/custom-dir

How to build the SD card

- Run the VM on VirtualBox
- Connect your SD card reader to your computer than to the virtual machine, from VirtualBox menu select *Devices → USB Devices → "your SDcard reader"*.
- Insert the SD card into the adapter (in this example we are inserting an SD card already partitioned with a FAT and an EXT3 partition, that is the basic configuration for the Pengwyn board).
- Run the following command (sudo password: **architech**)
- Find the device name from fdisk command output, in this example is */dev/sdb*.
- Run the script (in pengwyn-tools folder) that will prepare the SD card with all the needed files to run the system:

When the script starts asks for the *sudo password*, type *architech* followed by *enter*.
The list of available devices will be shown: the SD card should be the number **1** of the list with name sdb. Check the size shown on the table to be sure that the device is the correct one. Enter the device number **1** followed by enter-key.

The script will create two partitions on the SD card: the first one is a **FAT32** with the **boot files** (*MLO*, *u-boot.img*, *zImage-pengwyn-dvi.dtb*, *zImage* will be renamed to *MLO*, *u-boot.img*, *pengwyn.dtb* and *zImage*), the second one is an *ext3* with the *target file system*.
The operations will take few minutes.

When you build a new file system you can delete everything contained on the second partition and you can untar file *.tar.gz* to the second partition on the SD card.

If you have built a new kernel just overwrite the old one on the first partition.

After a writing operation use always *sync* command to make sure everything has been really written to the SD card:

Toolchain

Once your (virtual/)machine has been set up you can compile, customize the BSP for your board, write and debug applications, change the file system on-the-fly directly on the board, etc. This chapter will guide you to the basic use of the most important tools you can use to build customize, develop and tune your board.

Bitbake

Bitbake is the most important and powerful tool available inside Yocto/OpenEmbedded. It takes as input configuration files and recipes and produces what it is asked for, that is, it can build a package, the Linux kernel, the bootloader, an entire operating system from scratch, etc.

A **recipe** (*.bb* file) is a collection of metadata used by BitBake to set **variables** or define additional build-time **tasks**. By means of *variables*, a recipe can specify, for example, where to get the sources, which build process to use, the license of the package, an so on. There is a set of predefined *tasks* (the fetch task for example fetches the sources from the network, from a repository or from the local machine, than the sources are cached for later reuses) that executed one after the other get the job done, but a recipe can always add custom ones or override/modify existing ones. The most fine-graned operation that Bitbake can execute is, in fact, a single task.

Environment

To properly run Bitbake, the first thing you need to do is setup the shell environment. Luckily, there is a script that takes care of it, all you need to do is:

Inside the virtual machine, you can find *oe-init-build-env* script inside:

If you omit the build directory path, a directory named **build** will be created under your current working directory.

By default, with the SDK, the script is used like this:

Your current working directory changes to such a directory and you can customize configurations files (that the environment script put in place for you when creating the directory), run Bitbake to build whatever pops to your mind as well run hob. If you specify a custom directory, the script will setup all you need inside that directory and will change your current working directory to that specific directory.

Important: The build directory contains all the caches, builds output, temporary files, log files, file system images... everything!

The default build directory for Pengwyn is located under:

and the splash screen has a facility (a button located under Pengwyn's page) that can take you there with the right environment already in place so you are productive right away.

Important:

If you don't use the default build directory you need setup the local.conf file. See the paragraph below.

Configuration files

Configuration files are used by Bitbake to define variables value, preferences, etc..., there are a lot of them. At the beginning you should just worry about two of them, both located under *conf* directory inside your build directory, we are talking about **local.conf** and **bblayers.conf**.

local.conf contains your customizations for the build process, the most important variables you should be interested about are: **MACHINE**, **DISTRO**, **BB_NUMBER_THREADS** and **PARALLEL_MAKE**. *MACHINE* defines the target machine you want compile against. The proper value for Pengwyn is pengwyn:

DISTRO let you choose which distribution to use to build the root file systems for the board. The default distribution to use with the board is:

BB_NUMBER_THREADS and *PARALLEL_MAKE* can help you speed up the build process. *BB_NUMBER_THREADS* is used to tell Bitbake how many tasks can be executed at the same time, while *PARALLEL_MAKE* contains the **-j** option to give to *make* program when issued. Both *BB_NUMBER_THREADS* and *PARALLEL_MAKE* are related to the number of processors of your (virtual) machine, and should be set with a number that is two times the number of processors on your (virtual) machine. If for example, your (virtual) machine has/sees four cores, then you should set those variables like this:

bblayers.conf is used to tell Bitbake which meta-layers to take into account when parsing/looking for recipes, machine, distributions, configuration files, bbclasses, and so on. The most important variable contained inside *bblayers.conf* is **BBLAYERS**, it's the variable where the actual meta-layers layout get specified.

All the variables value we just spoke about are taken care of by Architech installation scripts.

Command line

With your shell setup with the proper environment and your configuration files customized according to your board and your will, you are ready to use Bitbake. The first suggestion is to run:

Bitbake will show you all the options it can be run with. During normal activity you will need to simply run a command like:

for example:

Such a command will build bootloader, Linux kernel and a root file system. *core-image-minimal-dev* tells Bitbake to execute whatever recipe

you just place the name of the recipe without the extension *.bb*.

Of course, there are times when you want more control over Bitbake, for example, you want to execute just one task like recompiling the Linux kernel, no matter what. That action can be achieved with:

where *-c compile* states the you want to execute the *do_compile* task and *-f* forces Bitbake to execute the command even if it thinks that there are no modifications and hence there is no need to to execute the same command again.

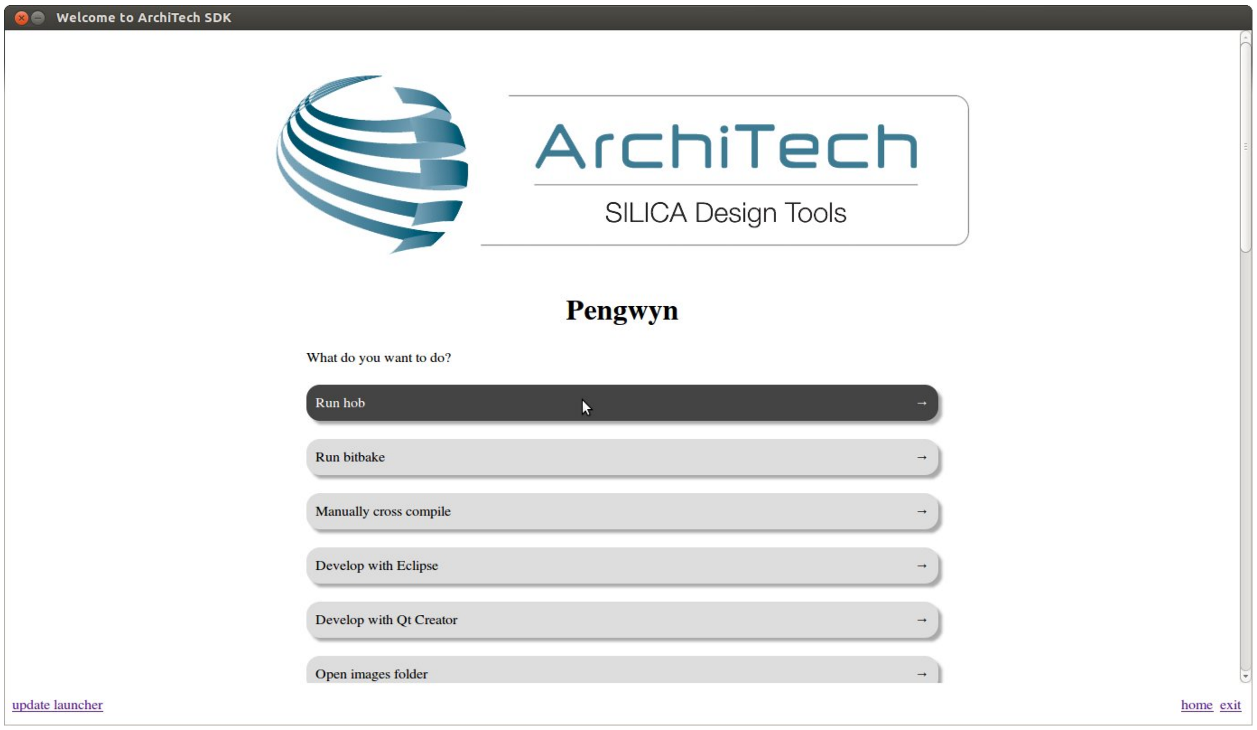
Another useful option is *-e* which gets Bitbake to print the environment state for the command you ran.

The last option we want to introduce is *-D*, which can be in fact repeated more than once and asks Bitbake to emit debug print. The amount of debug output you get depend on many times you repeated the option.

Of course, there are other options, but the ones introduced here should give you an head start.

Hob

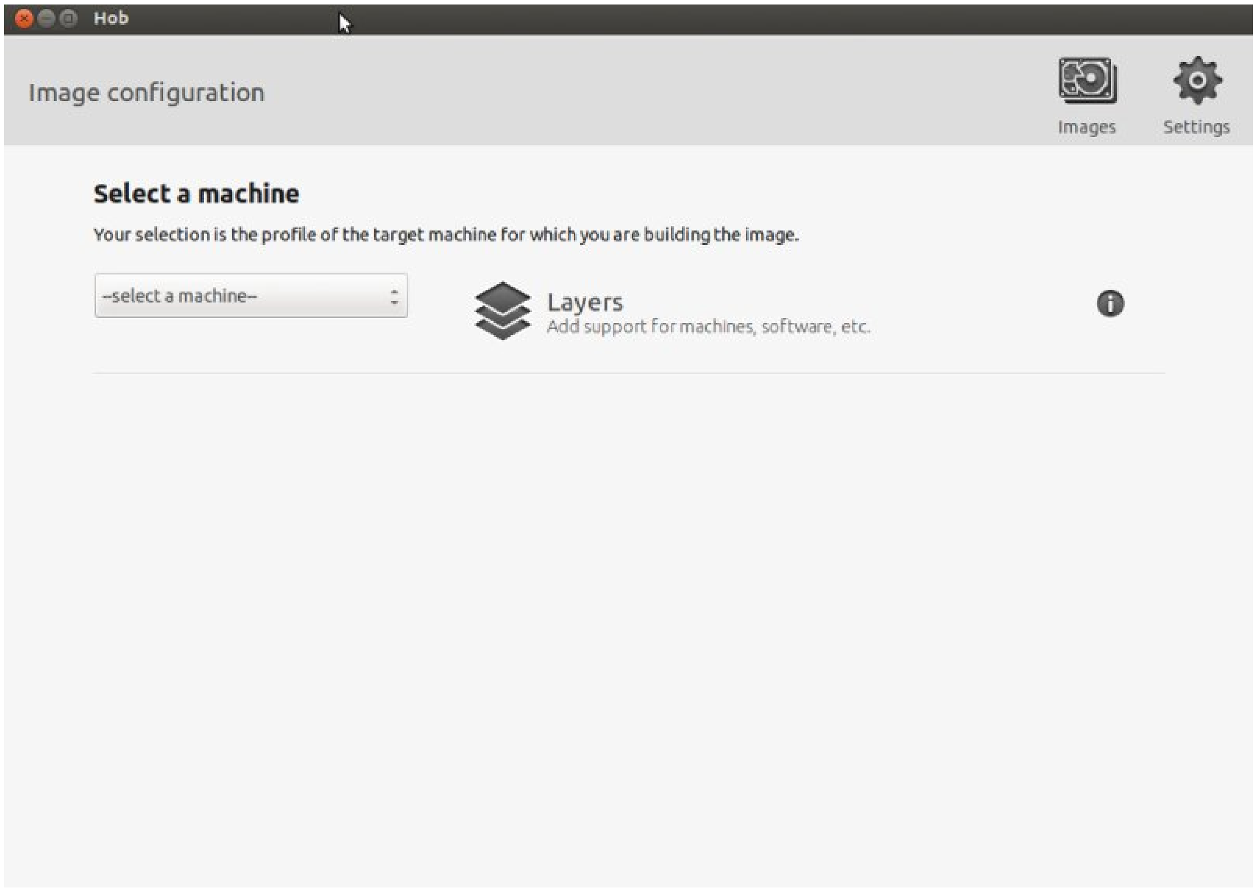
HOB is a graphical interface for bitbake. To run it, prepare the environment than run **hob** command or launch Hob using Architech Splashscreen just click on **Run hob**.



Warning:

Internet connection required.
At the first launch of Hob it takes some time before appears.

HOB window will appear.



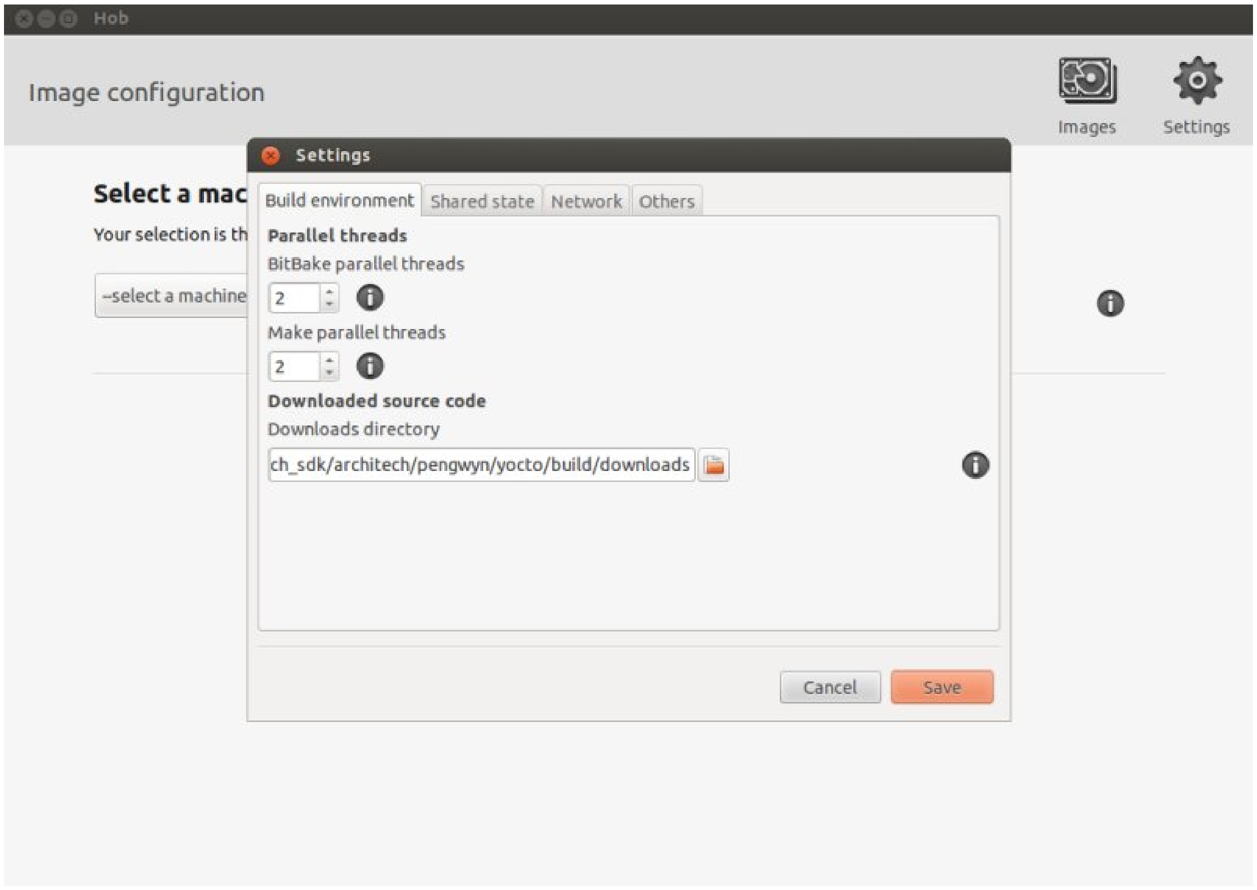
Select a machine

Your selection is the profile of the target machine for which you are building the image.

How to speedup the build process

When you imported the virtual machine you might have changed the number of processors made available to the virtual machine itself. If you did that, you can optimize the build time in this way:

- Click on **Settings**, a new window will appear. Select on **Build environment** tab.



Parallel threads

BitBake parallel threads: 2

Make parallel threads: 2

Downloaded source code

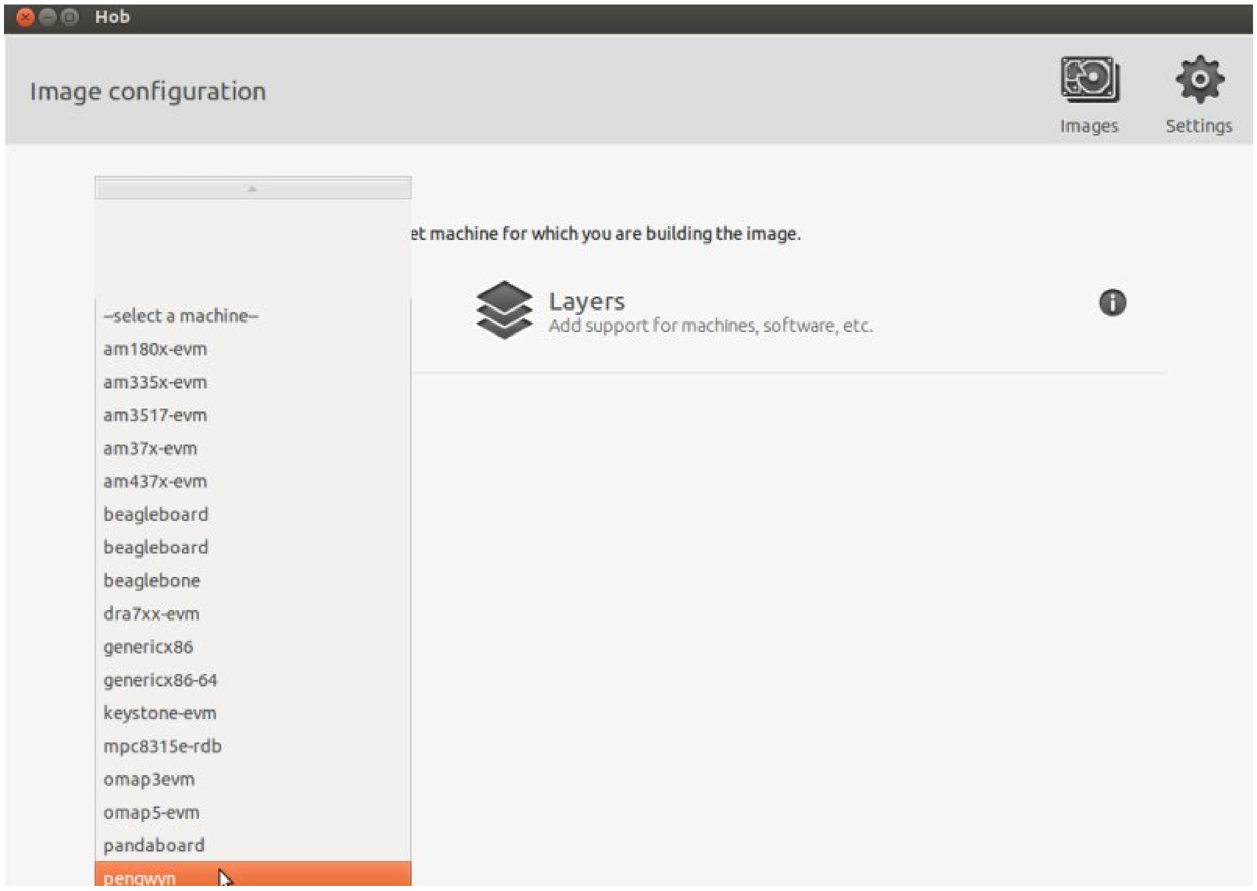
Downloads directory: ch_sdk/architech/pengwyn/yocto/build/downloads

- Change **BB number threads** value from 2 to *<number of processors used by the virtual machine> * 2*, set **Parallel make** to the same value. Click on save.

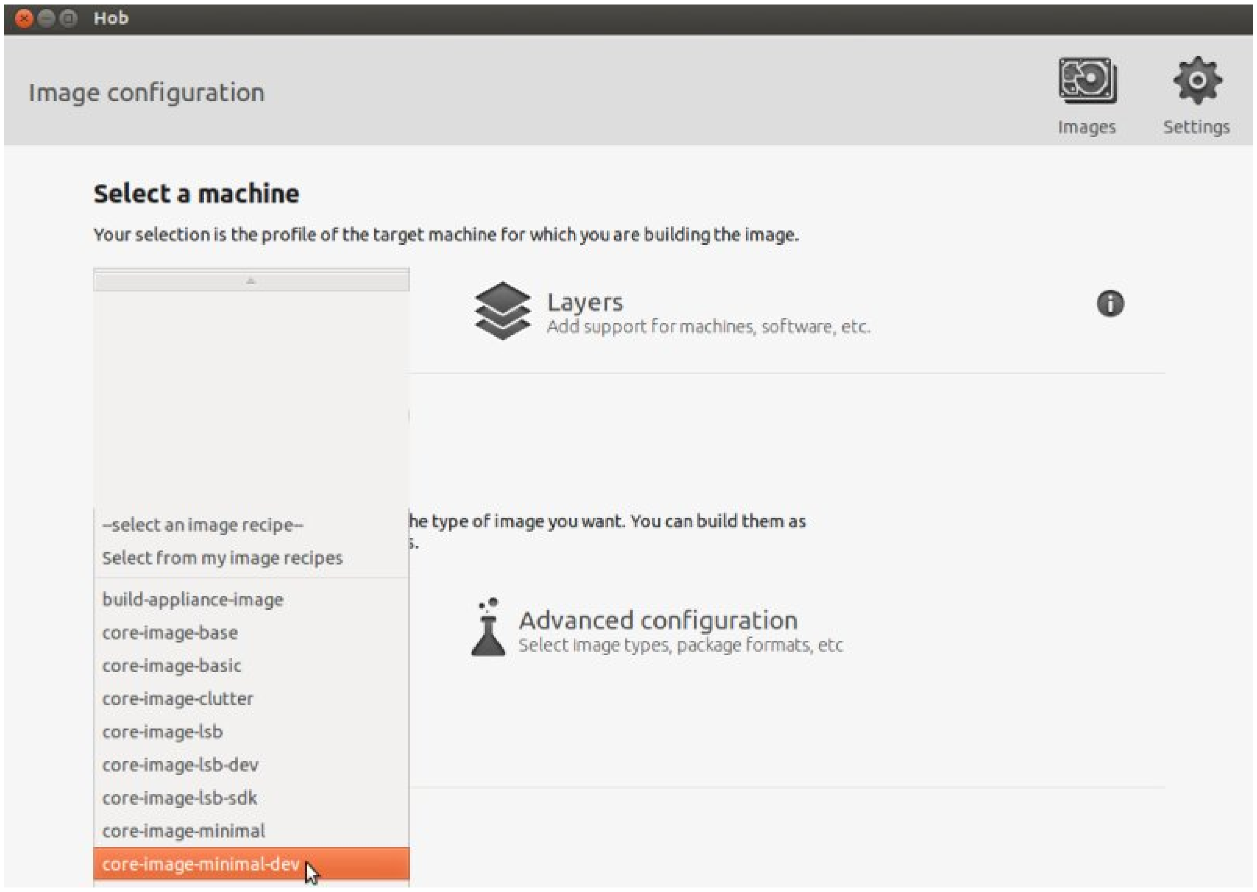
This modification will be available just for the current build directory.

How to build a predefined image

Select **pengwyn** from the combo-box,



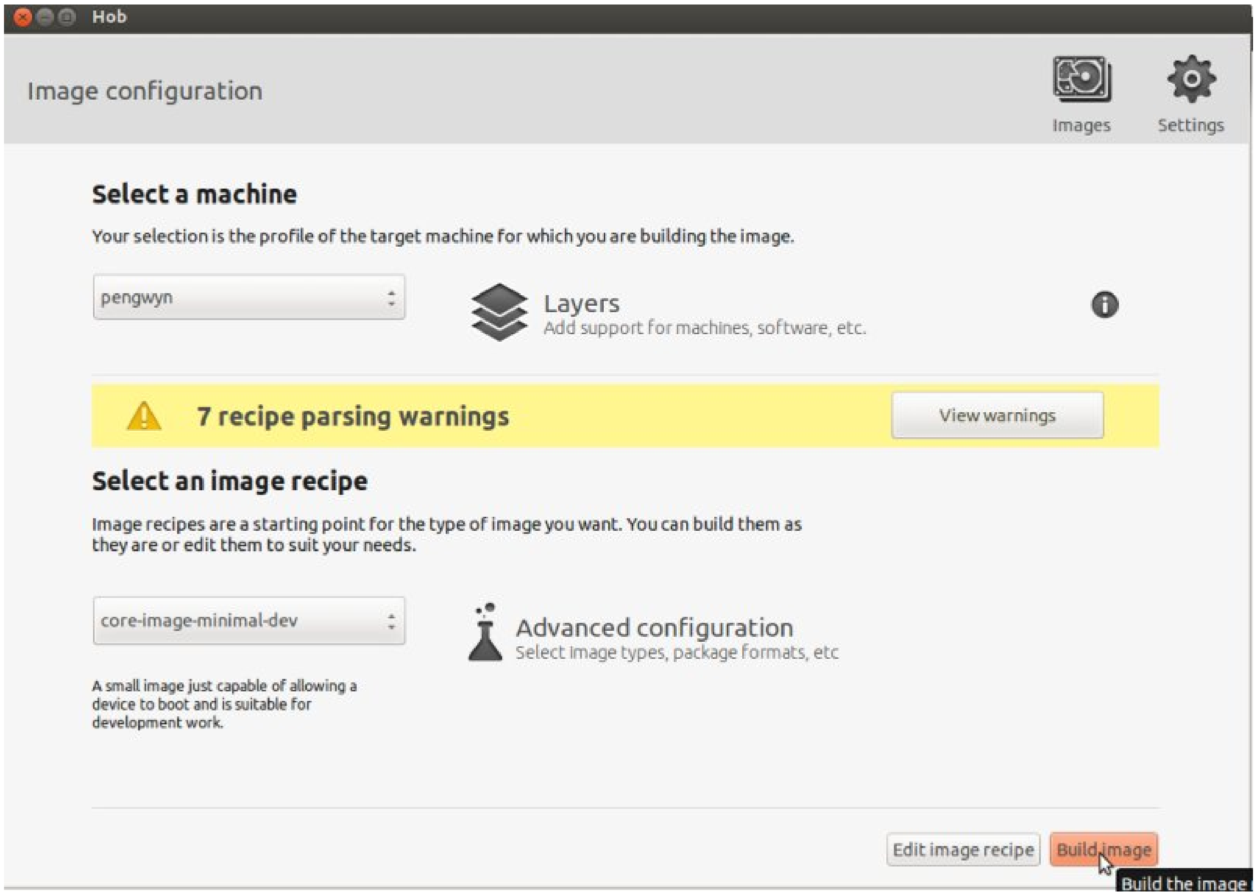
after the recipes have been parsed the section **Select a base image** will appear and you can choose your preferred image.


Hob
Image configuration
Images
Settings
Select a machine
Your selection is the profile of the target machine for which you are building the image.
Layers
Add support for machines, software, etc.
he type of image you want. You can build them as
s.
Advanced configuration
Select image types, package formats, etc
--select an image recipe--
Select from my image recipes
build-appliance-image
core-image-base
core-image-basic
core-image-clutter
core-image-lsb
core-image-lsb-dev
core-image-lsb-sdk
core-image-minimal
core-image-minimal-dev

Click on **Build image**

Warning:

The warning messages advice that some libraries supports only hardfp mode



Select a machine

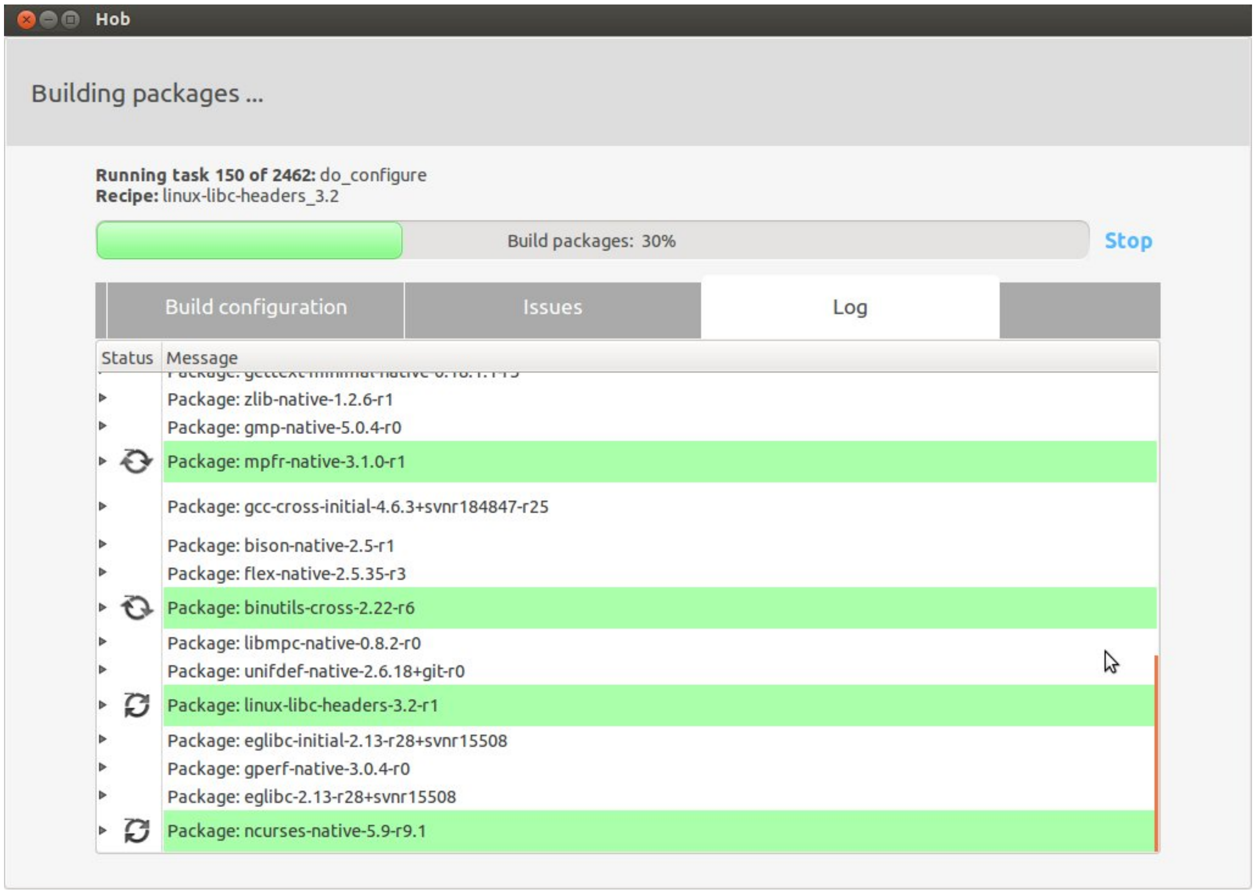
Your selection is the profile of the target machine for which you are building the image.

Select an image recipe

Image recipes are a starting point for the type of image you want. You can build them as they are or edit them to suit your needs.

A small image just capable of allowing a device to boot and is suitable for development work.

than the build process will start.

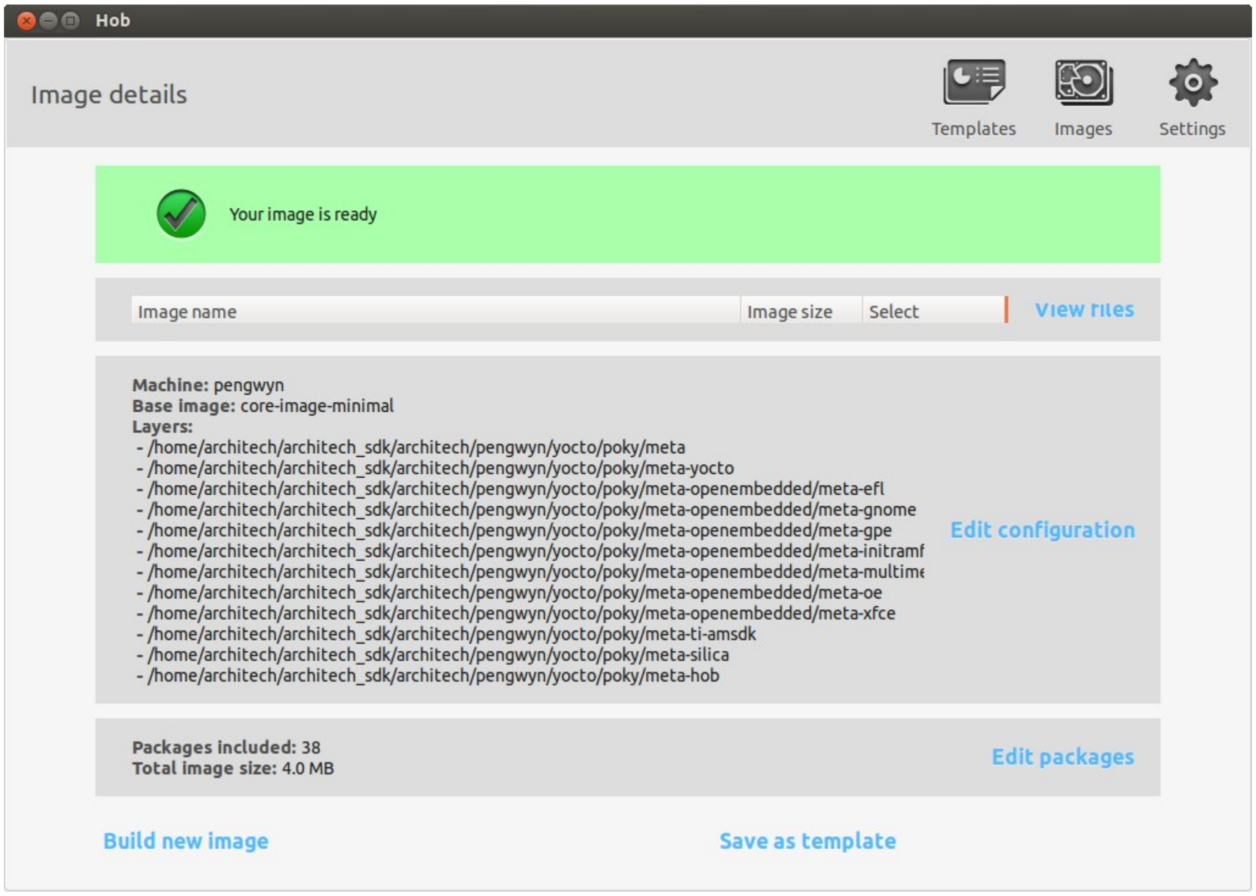


Running task 150 of 2462: do_configure
Recipe: linux-libc-headers_3.2

Build packages: 30%

```
Status  Message
        Package: gettext-minimal-native-0.18.1.1-r3
        Package: zlib-native-1.2.6-r1
        Package: gmp-native-5.0.4-r0
        Package: mpfr-native-3.1.0-r1
        Package: gcc-cross-initial-4.6.3+svnr184847-r25
        Package: bison-native-2.5-r1
        Package: flex-native-2.5.35-r3
        Package: binutils-cross-2.22-r6
        Package: libmpc-native-0.8.2-r0
        Package: unifdef-native-2.6.18+git-r0
        Package: linux-libc-headers-3.2-r1
        Package: eglibc-initial-2.13-r28+svnr15508
        Package: gperf-native-3.0.4-r0
        Package: eglibc-2.13-r28+svnr15508
        Package: ncurses-native-5.9-r9.1
```

Please note that the build process can last several hours till it finishes.



Machine: pengwyn
Base image: core-image-minimal
Layers:
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-yocto
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-openembedded/meta-efl
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-openembedded/meta-gnome
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-openembedded/meta-gpe
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-openembedded/meta-initramf
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-openembedded/meta-multime
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-openembedded/meta-oe
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-openembedded/meta-xfce
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-ti-amsdk
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-silica
- /home/architech/architech_sdk/architech/pengwyn/yocto/poky/meta-hob

Packages included: 38
Total image size: 4.0 MB

It might happen that a fetch task gives an error, if so, double check that the virtual machine has a proper network configuration. If the network configuration has been proved correct, the error might mean that the needed server is down for some reason, in that case the only option you have is to wait and try again later.

How to build a custom image

There are two possible ways to customize an image to build:

- modify a predefined image,
- create a new image from scratch.

Once you selected a predefined image, you can click on **View recipes** to add/remove recipes and tasks or you can click on **View packages** to add/remove previously built packages. After the image has been customized you can build your image. If you want to customize every detail you can choose **Create your own image** from the drop down menu of section **Select a base image**, than, as previously stated, you can customize the content of your file system and build it.

Eclipse

Eclipse is an integrated development environment (IDE). It contains a base workspace and the Yocto plug-in system to compile and debug a program for Pengwyn. Hereafter, the operating system that runs the IDE/debugger will be named host machine, and the board being debugged will be named target machine. The host machine could be running as a virtual machine guest operating system, anyway, the documentation for the host machine running as a guest operating system and as host operating system is exactly the same.

To write your application you need:

- a root file system filesystem (you can use *bitbake*/*hob* to build your preferred filesystem) with development support (that is, it must include all the necessary libraries, header files, the *tcf-agent* program and *gdbserver*) included
- a media with the *root filesystem* installed and, if necessary, the bootloader
- Pengwyn *powered up* with the aforementioned root file system
- a working *serial console* terminal
- a working *network* connection between your workstation and the board (connector *XF1*), so, be sure that:

1. your board has ip address 192.168.0.10 on interface eth0, and
2. your PC has an ip address in the same family of addresses, e.g. 192.168.0.100.

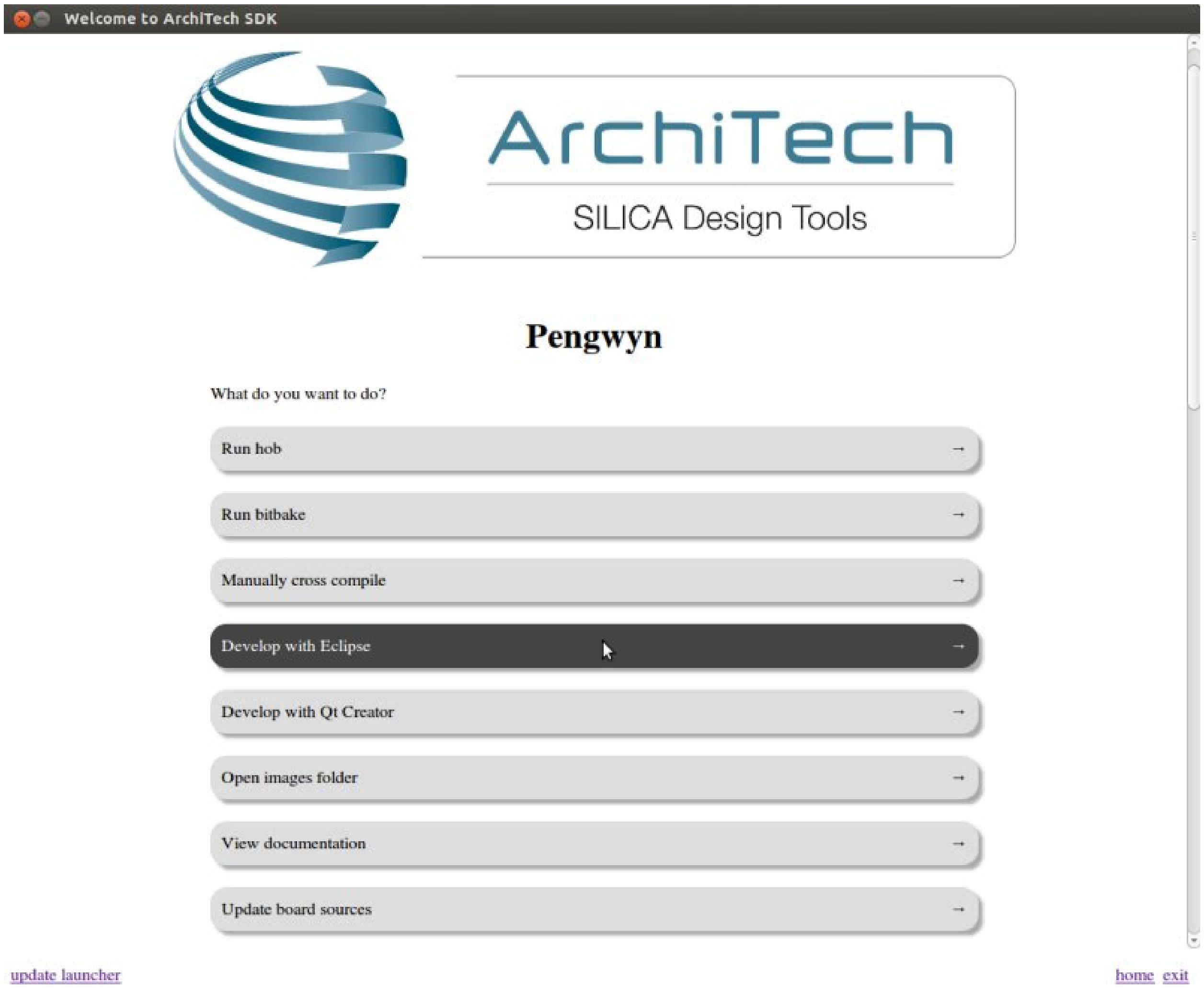
Note:

Before to start remember to copy the cross-toolchain libreries to sysroot

cp -r
/home/architech/architech_sdk/architech/pengwyn/toolchain/sysroots/cortexa8t2hf-vfp-neon-poky-linux-gnueabi/*
/home/architech/architech_sdk/architech/pengwyn/sysroot

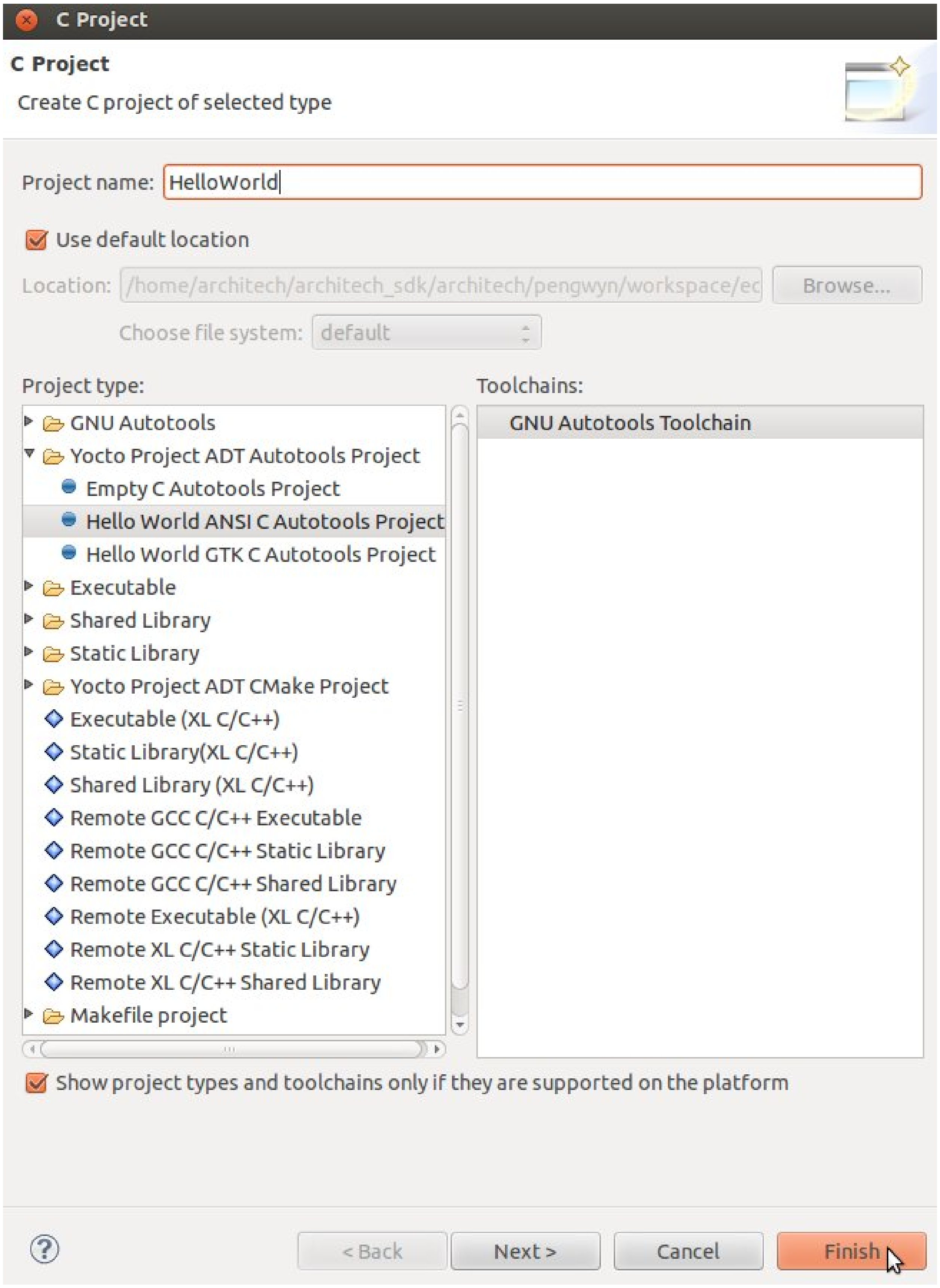
Creating the Project

You can create two types of projects: Autotools-based, or Makefile-based. This section describes how to create Autotools-based projects from within the **Eclipse IDE**. Launch Eclipse using Architech Splashscreen just click on **Develop with Eclipse**.



To create a project based on a Yocto template and then display the source code, follow these steps:

- Select File→New→Project...
- Under *C/C++*, double click on *C Project* to create the project.
- Click on “Next” button
- Expand *Yocto Project ADT Autotools Project*.
- Select *Hello World ANSI C Autotools Project*. This is an Autotools-based project based on a Yocto Project template.



C Project

C Project

Create C project of selected type

Project name: HelloWorld

☑ Use default location

Location: /home/architech/architech_sdk/architech/pengwyn/workspace/ec

Choose file system: default

Project type:

- GNU Autotools
- Yocto Project ADT Autotools Project
  - Empty C Autotools Project
  - Hello World ANSI C Autotools Project
  - Hello World GTK C Autotools Project
- Executable
- Shared Library
- Static Library
- Yocto Project ADT CMake Project
- Executable (XL C/C++)
- Static Library(XL C/C++)
- Shared Library (XL C/C++)
- Remote GCC C/C++ Executable
- Remote GCC C/C++ Static Library
- Remote GCC C/C++ Shared Library
- Remote Executable (XL C/C++)
- Remote XL C/C++ Static Library
- Remote XL C/C++ Shared Library
- Makefile project

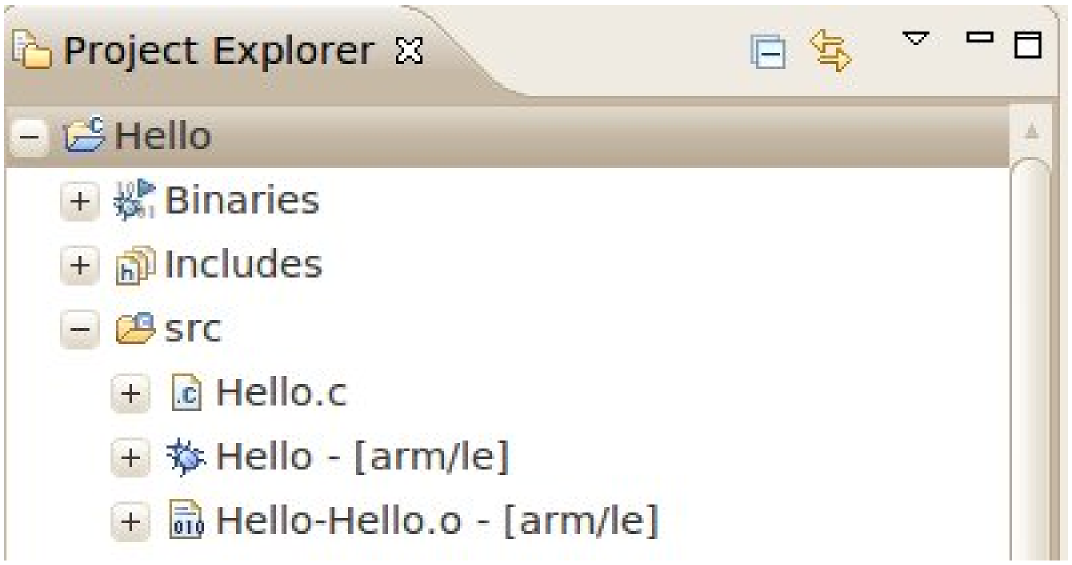
Toolchains:

GNU Autotools Toolchain

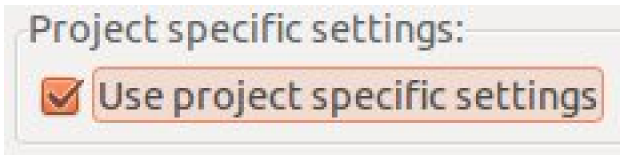
☑ Show project types and toolchains only if they are supported on the platform

- Put a name in the Project *name:* field. Do not use hyphens as part of the name.
- Click *Next*.
- Add information in the *Author* and *Copyright* notice fields.
- Be sure the *License* field is correct.
- Click *Finish*.

Note: If the "open perspective" prompt appears, click *Yes* so that you enter in C/C++ perspective. The left-hand navigation panel shows your project. You can display your source by double clicking on the project source file.



- Select *Project→Properties→Yocto Project Settings* and check *Use project specific settings*

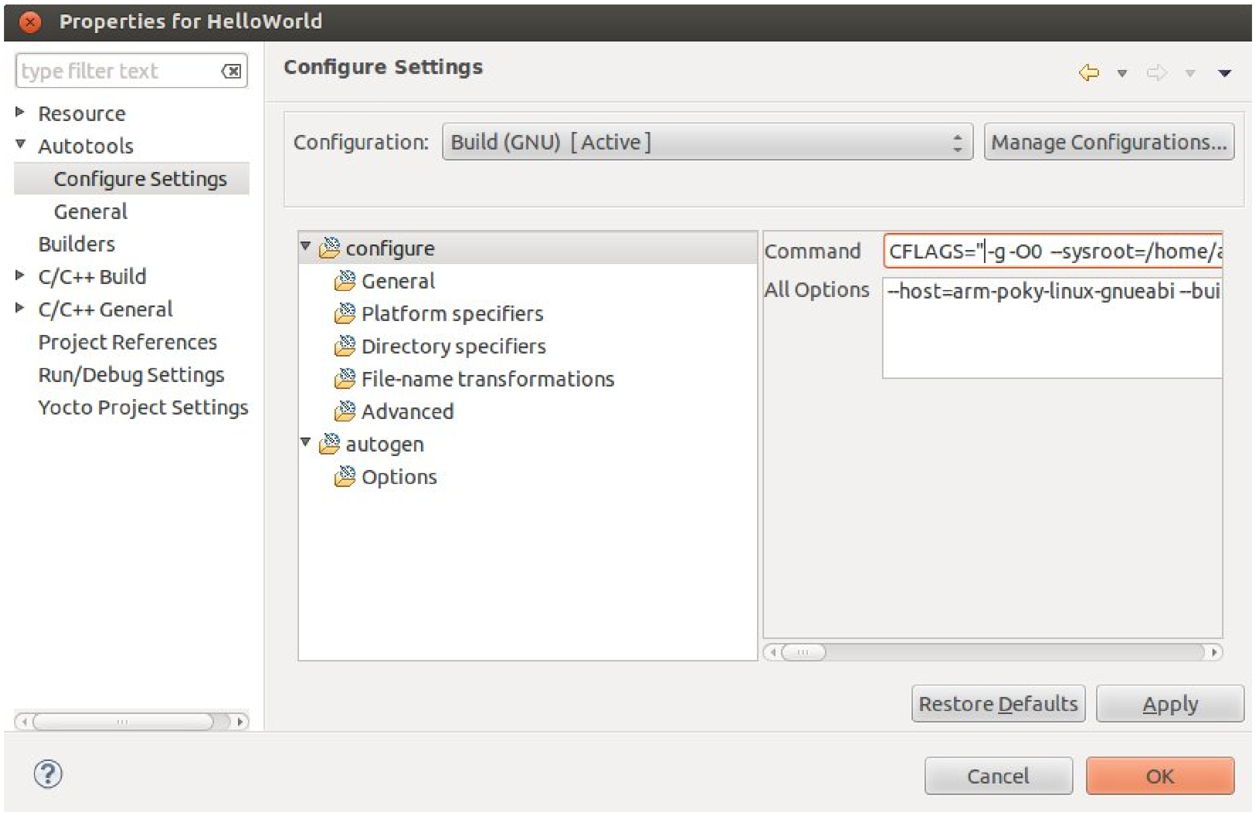


Building the Project

To build the project, select Project→Build Project. The console should update with messages from the cross-compiler. To add more libraries to compile:

- Click on Project→Properties.
- Expand the box next to Autotools.
- Select Configure Settings.
- In CFLAGS field, you can add the path of includes with -Ipath_include
- In LDFLAGS field, you can specify the libraries you use with -lname_library and you can also specify the path where to look for libraries with -Lpath_library
- Click on Project→Build All to compile the project

Note: All libraries must be located in */home/architech/architech_sdk/architech/pengwyn/sysroot* subdirectories.



Command: CFLAGS="-g -O0 --sysroot=/home/a

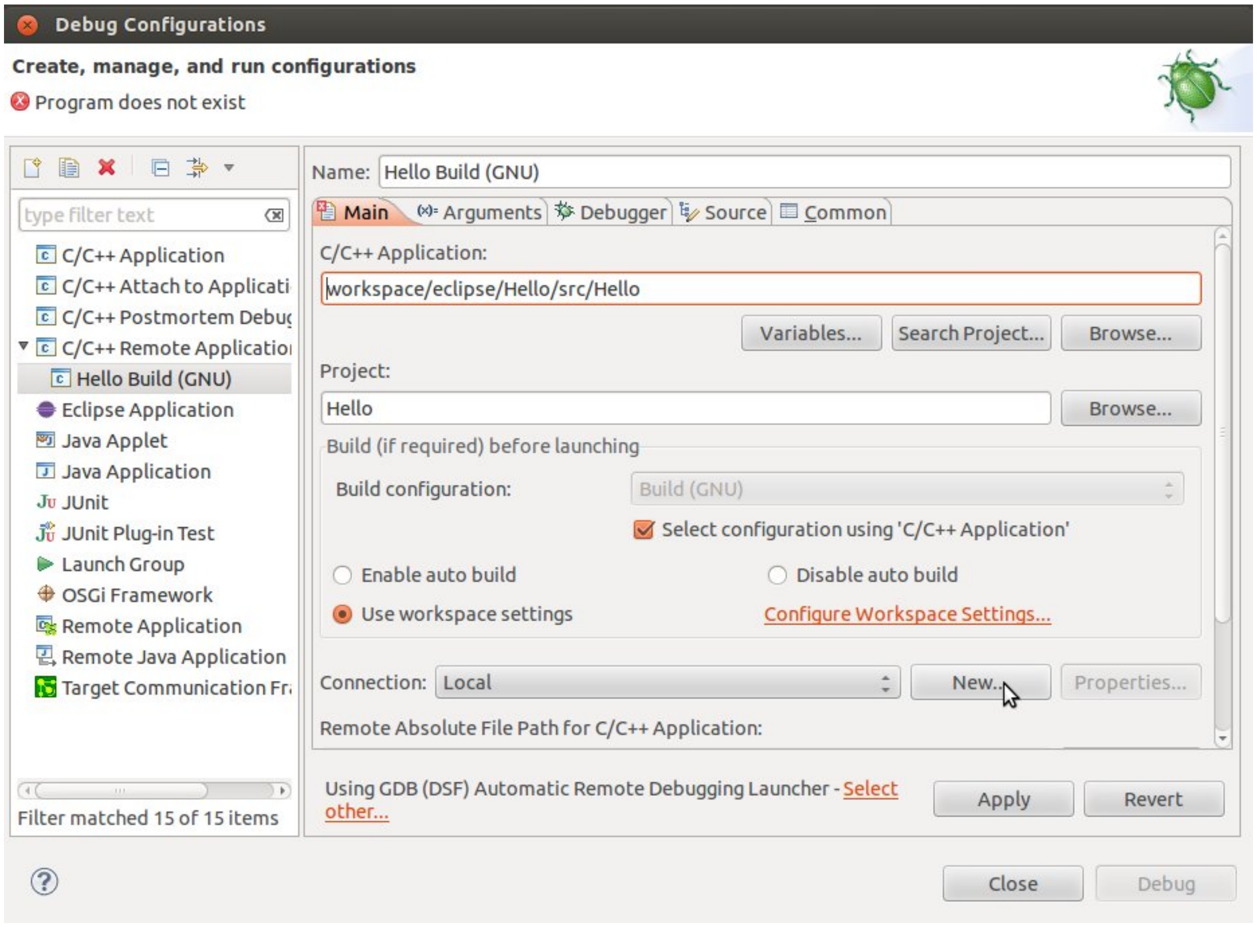
All Options: --host=arm-poky-linux-gnueabi --bui

Deploying and Debugging the Application

Connect Pengwyn console to your PC and power-on the board. Once you built the project and the board is running the image, use minicom to run **tcf-agent** program in target board:

On the Host machine, follow these steps to let **Eclipse** deploy and debug your application:

- Select Run→Debug Configurations...
- In the left area, expand *C/C++ Remote Application*.
- Locate your project and select it to bring up a new tabbed view in the *Debug Configurations* Dialog.



Name: Hello Build (GNU)

C/C++ Application: workspace/eclipse/Hello/src/Hello

Project: Hello

Build (if required) before launching

Build configuration: Build (GNU)

☑ Select configuration using 'C/C++ Application'

○ Enable auto build ○ Disable auto build

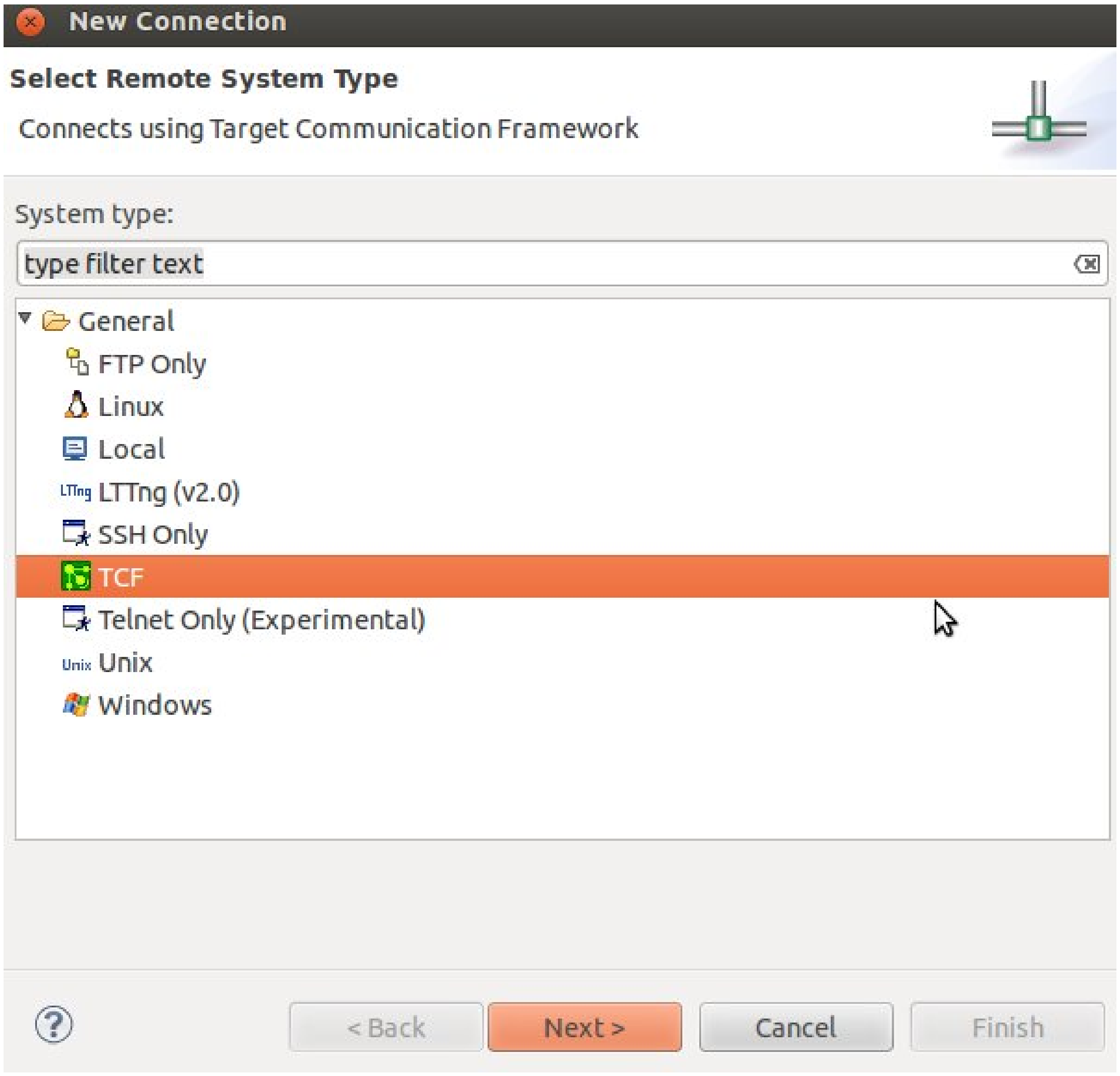
● Use workspace settings Configure Workspace Settings...

Connection: Local

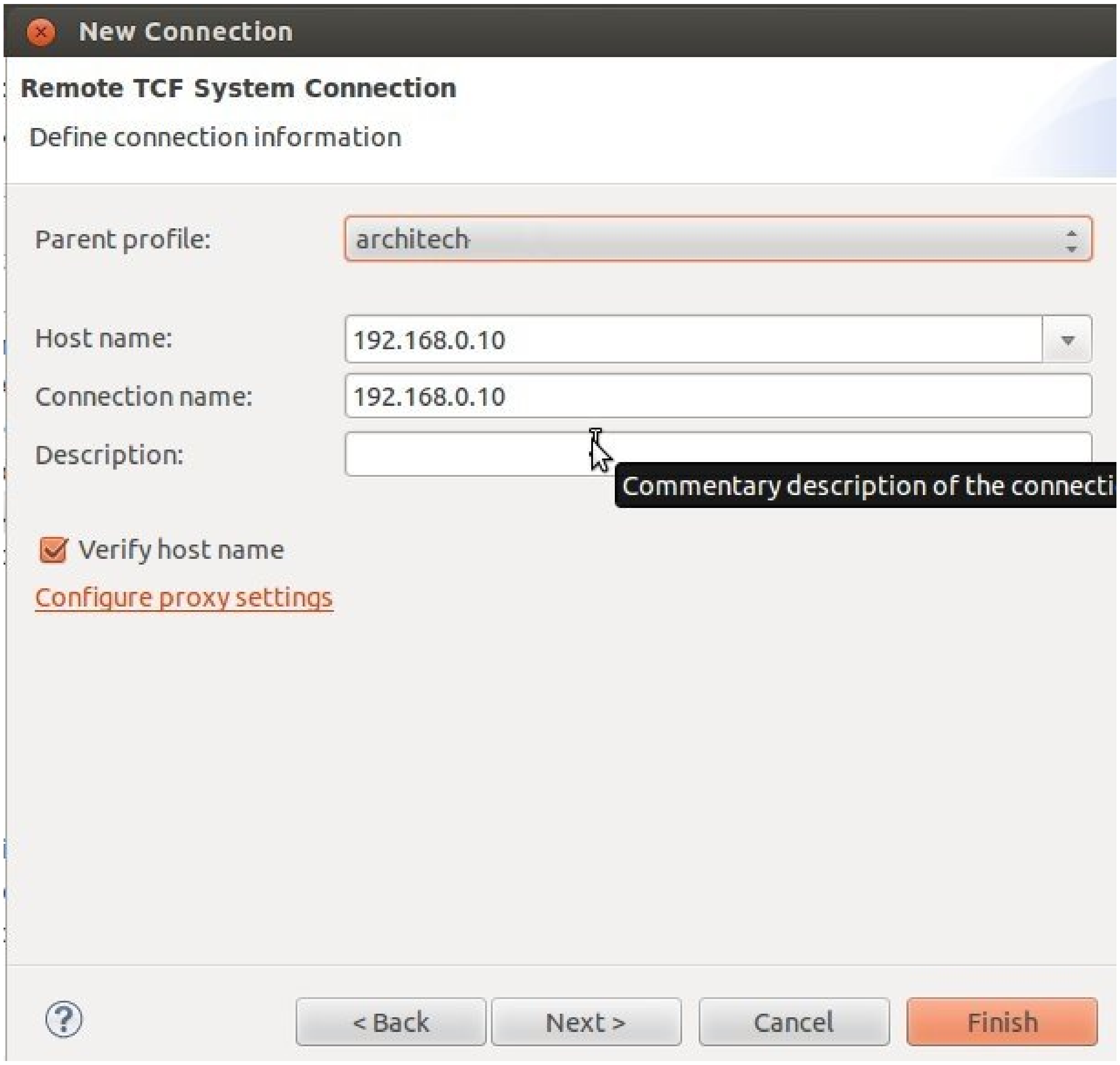
Remote Absolute File Path for C/C++ Application:

Using GDB (DSF) Automatic Remote Debugging Launcher - Select other...

- Insert in *C/C++ Application* the filepath of your application binary on your host machine.
- Click on "New" button near the drop-down menu in the *Connection* field.
- Select *TCF* icon.



- Insert in *Host Name* and *Connection Name* fields the IP address of the target board. (e.g. 192.168.0.10)



Remote TCF System Connection

Define connection information

Parent profile: architech

Host name: 192.168.0.10

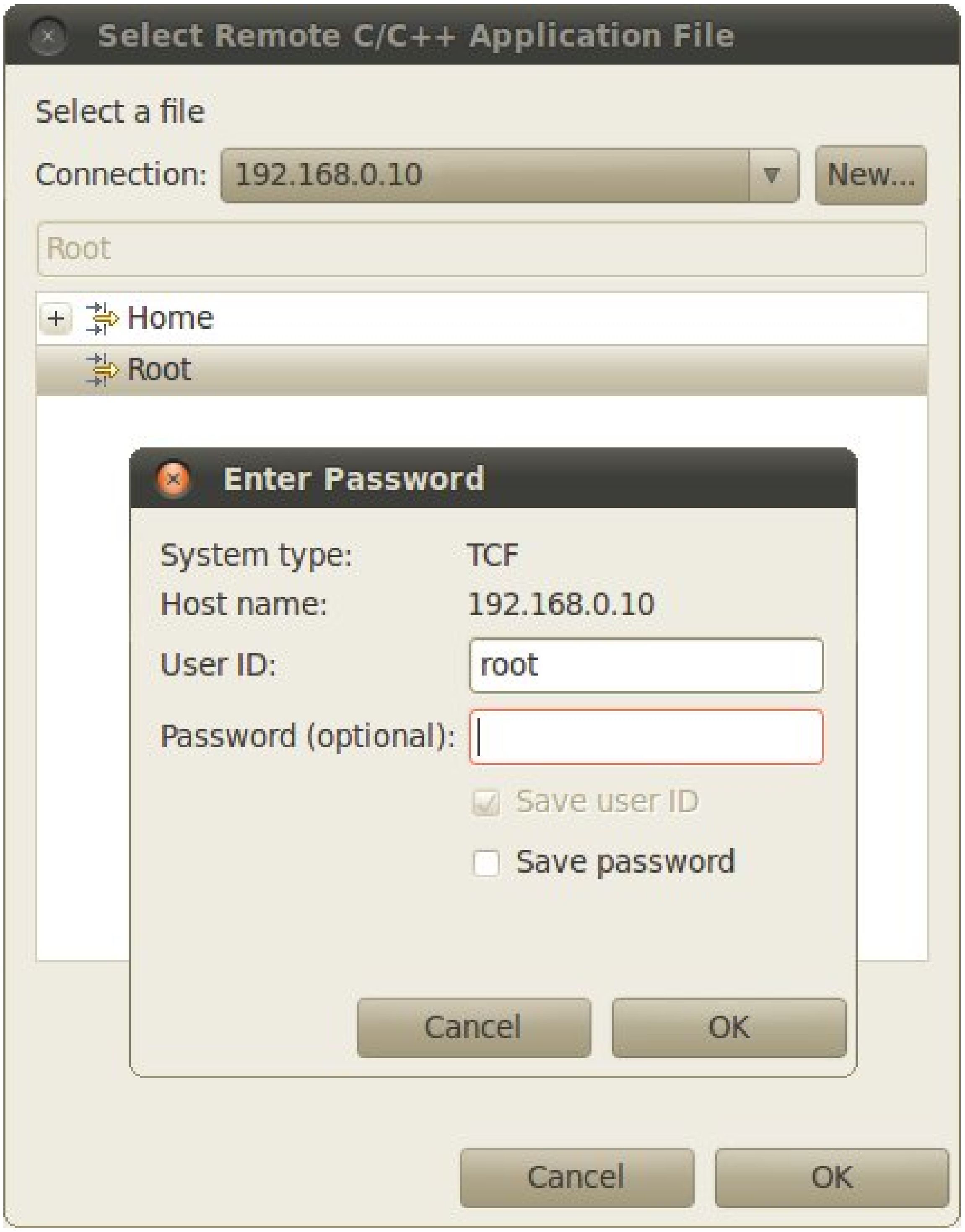
Connection name: 192.168.0.10

Description:

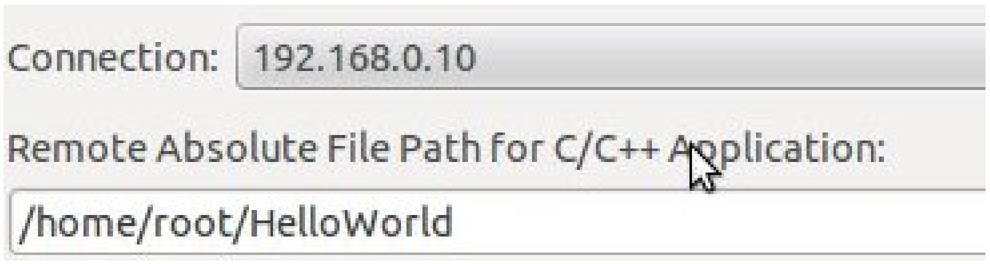
Commentary description of the connecti

[✓] Verify host name

- Press *Finish*.
- Use the drop-down menu now in the *Connection* field and pick the IP Address you entered earlier.
- Enter the absolute path on the target into which you want to deploy the application. Use *Browse* button near *Remote Absolute File Path for C/C++Application:* field. No password is needed.



- Enter also in the target path the name of the application you want to debug. (e.g. HelloWorld)

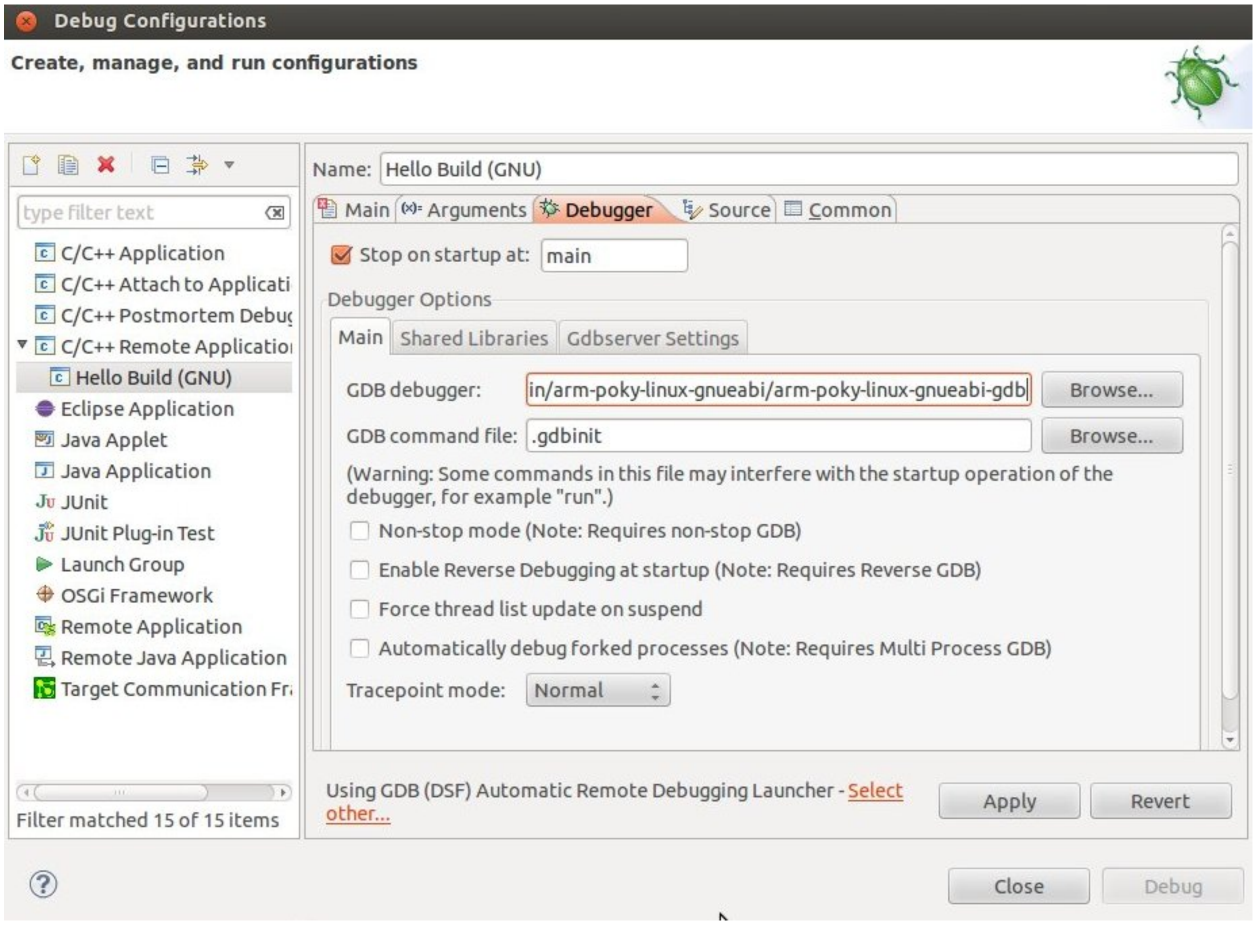


Connection: 192.168.0.10

Remote Absolute File Path for C/C++ Application:

/home/root/HelloWorld

- Select *Debugger* tab



Debug Configurations

Create, manage, and run configurations

Name: Hello Build (GNU)

[✓] Stop on startup at: main

GDB debugger: in/arm-poky-linux-gnueabi/arm-poky-linux-gnueabi-gdb

GDB command file: .gdbinit

(Warning: Some commands in this file may interfere with the startup operation of the debugger, for example "run".)

[ ] Non-stop mode (Note: Requires non-stop GDB)

[ ] Enable Reverse Debugging at startup (Note: Requires Reverse GDB)

[ ] Force thread list update on suspend

[ ] Automatically debug forked processes (Note: Requires Multi Process GDB)

Tracepoint mode: Normal

Using GDB (DSF) Automatic Remote Debugging Launcher - Select other...

- In GDB Debugger field, insert the filepath of gdb for your toolchain
- In *Debugger* window there is a tab named *Shared Library*, click on it.
- Add the libraries paths *lib* and *usr/lib* of the rootfs (which must be the same used in the target board)
- Click *Debug* to bring up a login screen and login.
- Accept the debug perspective.

Important: If debug does not work, check on the board if *tcf-agent* is running and *gdbserver* has been installed.

Qt cross-toolchain

The Qt Framework used by this SDK is composed of libraries for your host machine and your target. To compile the libraries for *x86* you only need your distribution toolchain, while to compile the libraries for Pengwyn board you need the proper cross-toolchain (see Chapter *Cross compiler* for further information on how to get it).

First of all you need to compile the cross-toolchain with Yocto:

The recipe builds **poky-glibc-i686-meta-toolchain-qte-cortexa8t2hf-vfp-neon-toolchain-qte-1.7.1.sh** installation script. You should find the installation script in */home/architech/architech_sdk/architech/pengwyn/yocto/build/tmp/deploy/sdk*. The cross-toolchain allows to compile a Qt embedded 4.8.5 application.

To install the toolchain run the following commands:

The installation script will ask to select an installation path.

Before to run Qt creator you must set the environment variables:

Note:

qtopia.sh is used to allow the compilation for the **qt4e-demo-image**

Qt Creator

Qt is a cross-platform application framework that is used to build applications. One of the best features of Qt is its capability of generating Graphical User Interfaces (GUIs).
Qt Creator is a cross-platform C++ IDE which includes a visual debugger, an integrated GUI layout and form designer. It makes possible to compile and debug applications on both **x86** (host) and **ARM** (target) machines.
This SDK relies on **version 4.8.5** of Qt and **version 2.8.1** of Qt Creator.

Before getting our hands dirty, make sure all these steps have been followed:

1. Use *Hob* or *Bitbake* to build an image which includes: *openssh*, support for C++, *tcf-agent* and *gdbserver*.

Note:

To follow this guide build *qt4e-demo-image* image. Remember to complete its file system (by the local.conf) with *tcf-agent*, *gdbserver* and *openssh*.

If the borad uses the touchscreen add the following line into the local.conf:

IMAGE_INSTALL_append = ” tslib tslib-conf tslib-tests tslib-calibrate”

2. Deploy the *root file system* just generated on the final media used to boot the board
3. Replicate the same root file system into directory
4. Copy the Qt Libraries to the board media used to boot
5. Copy the Qt Libraries and cpp libraries to your sdk sysroot directory
6. Unmount the media used to boot the board from your computer and insert it into the board
7. *Power-On* the board
8. Open up the *serial console*.

If you based your root file system on *qt4e-demo-image*, be sure you execute this command

to stop the execution of the demo application.

9. Provide a working *network* connection between your workstation and the board (connector *XF1*), so, be sure that:

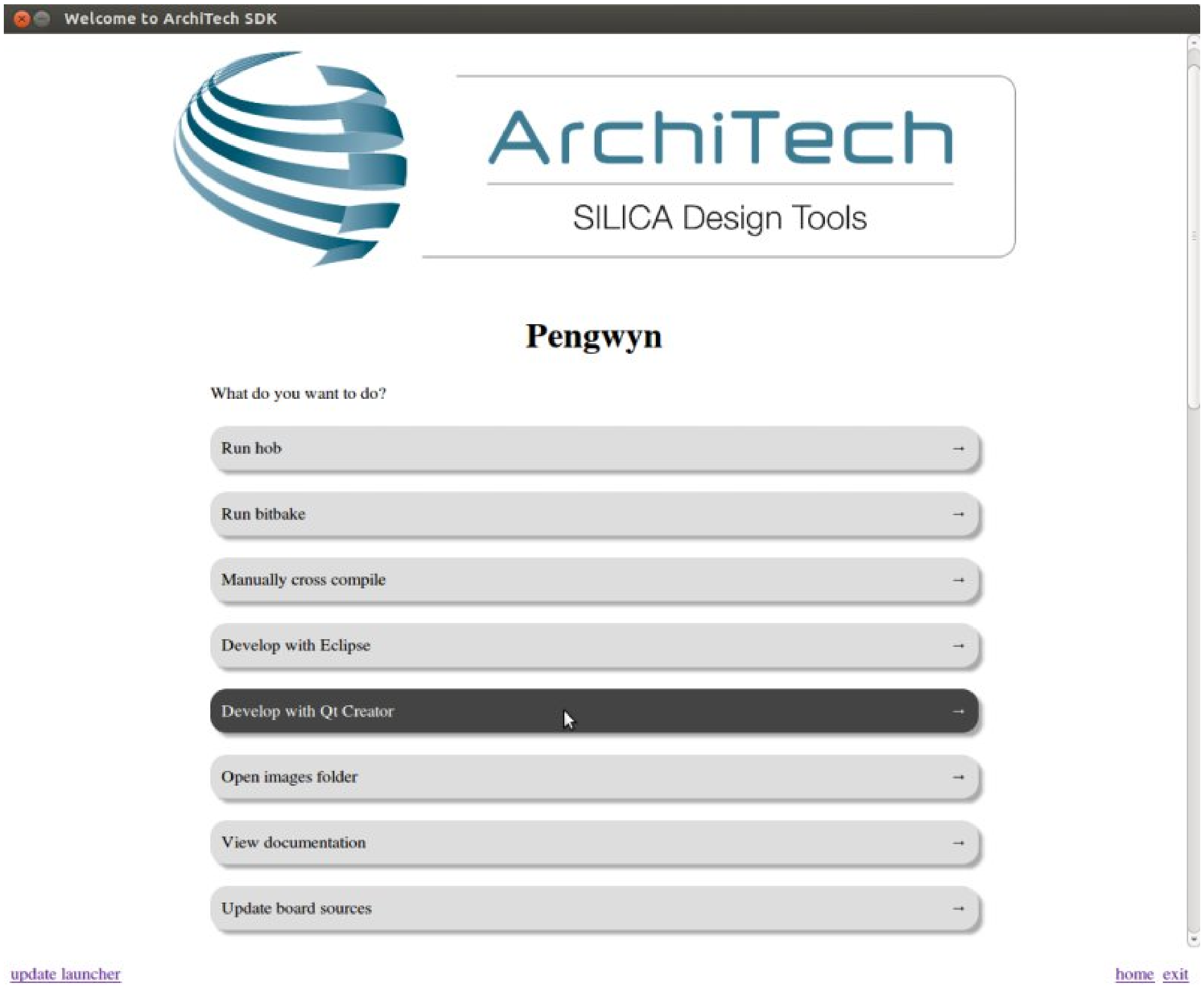
1. your board has ip address 192.168.0.10 on interface eth0, and
2. your PC has an ip address in the same family of addresses, e.g. 192.168.0.100.

Hello World!

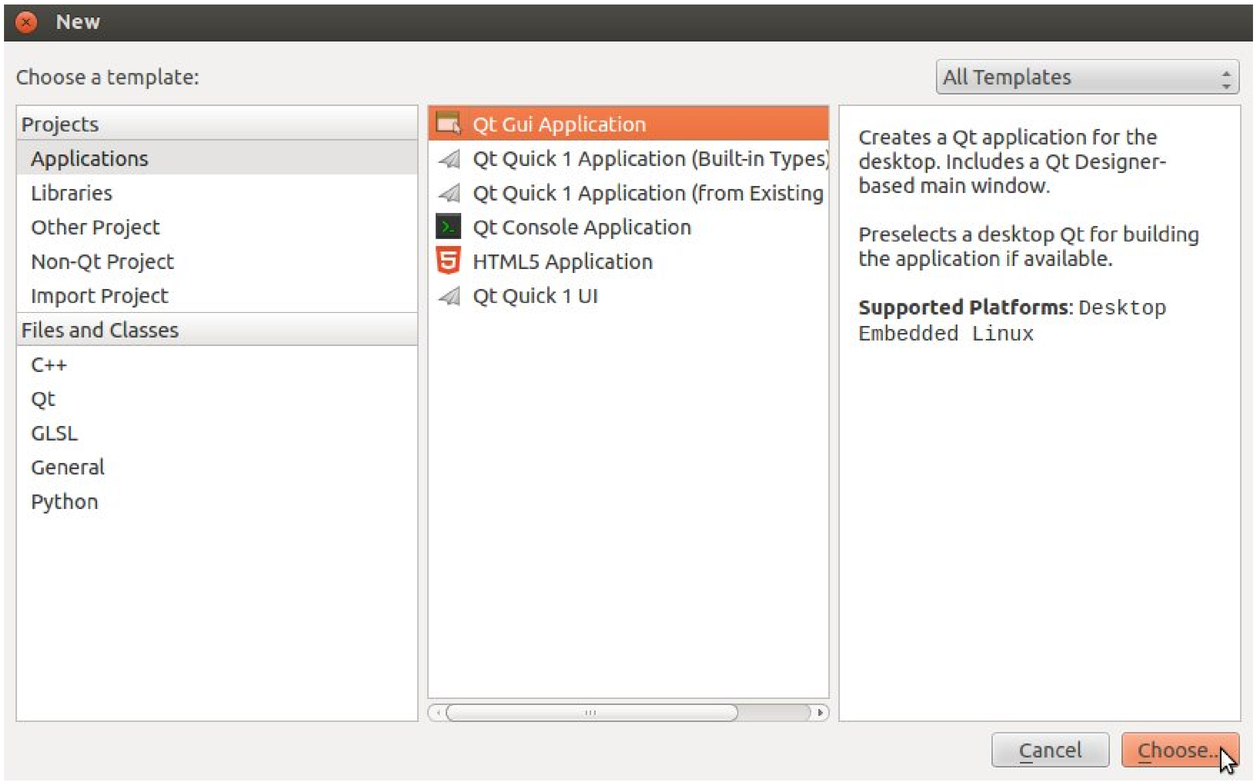
The purpose of this example project is to generate a form with an “Hello World” label in it, at the beginning on the x86 virtual machine and than on Pengwyn board.

To create the project follow these steps:

1. Use the **Welcome Screen** to run Qt Creator by selecting *Architech→Pengwyn→Develop with Qt Creator*



2. Go to *File -> New File or Project*. In the new window select *Applications* as project and *Qt Gui Application*. Click on *Choose...* button.

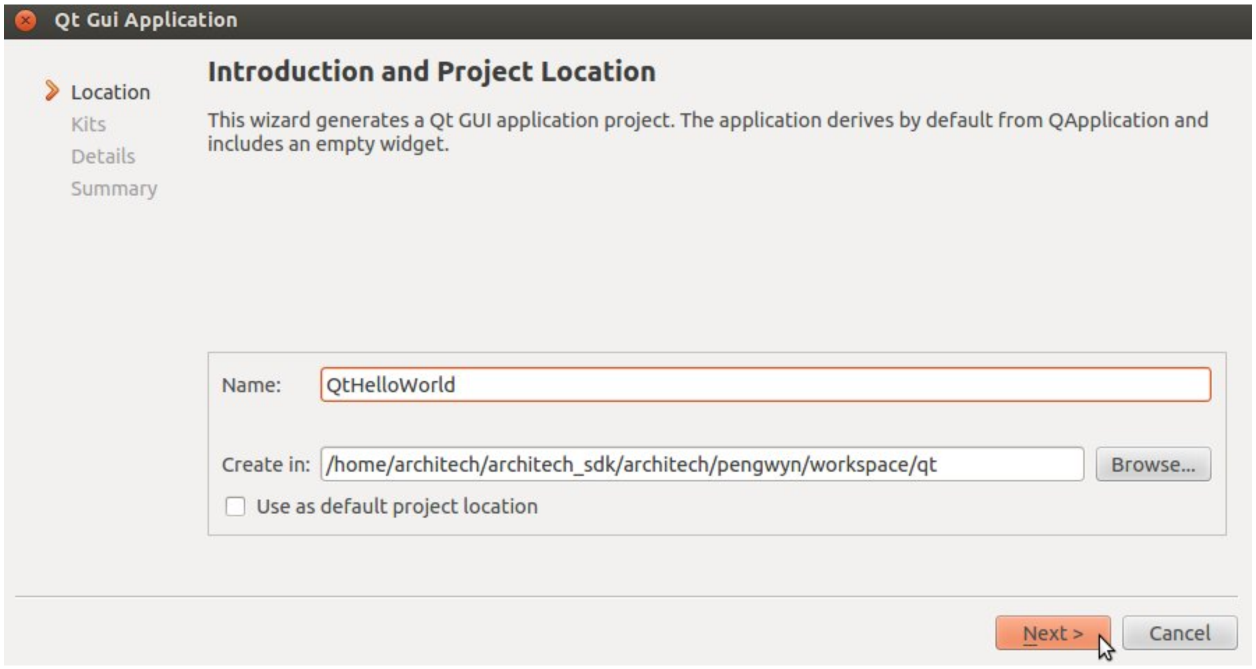


Creates a Qt application for the desktop. Includes a Qt Designer-based main window.

Preselects a desktop Qt for building the application if available.

Supported Platforms: `Desktop Embedded Linux`

3. Select a name for your project for example *QtHelloWorld* and press *next* button.



Introduction and Project Location

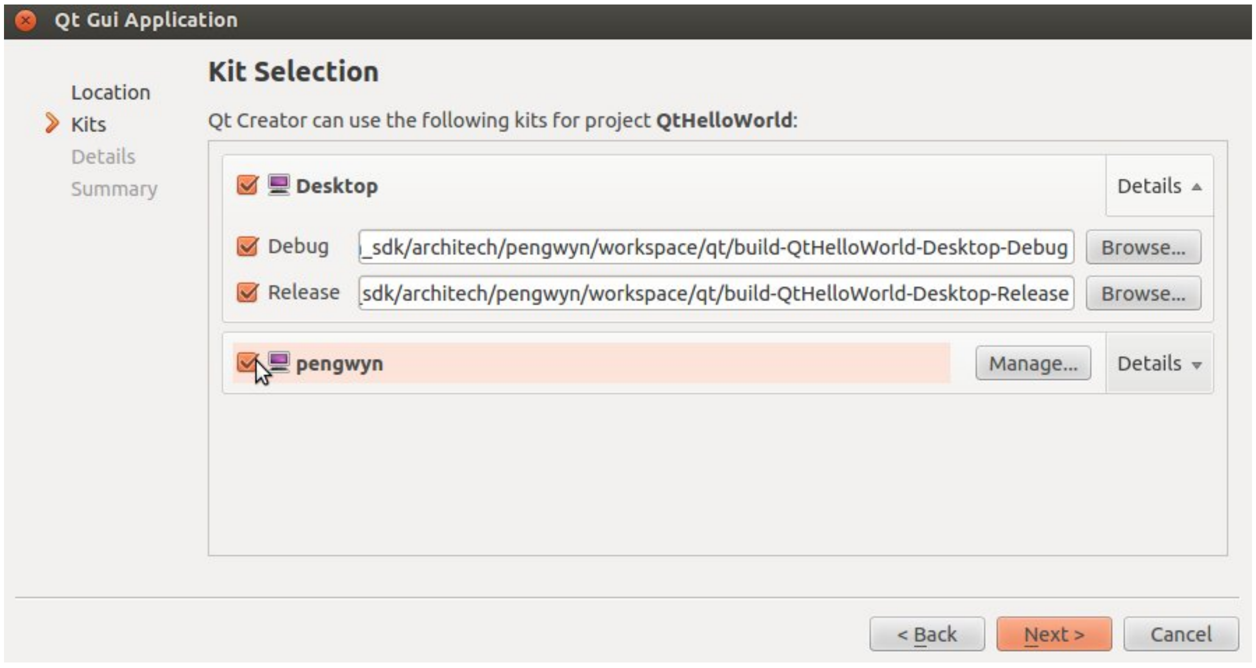
This wizard generates a Qt GUI application project. The application derives by default from QApplication and includes an empty widget.

Name: QtHelloWorld

Create in: /home/architech/architech_sdk/architech/pengwyn/workspace/qt

☐ Use as default project location

3. Check also *Pengwyn* kit and continue to press *next* button to finish the creation of the project.



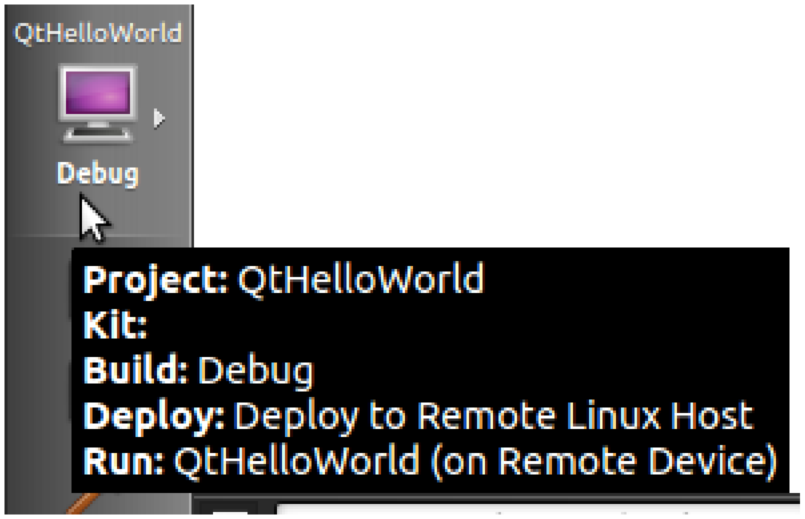
Kit Selection

Qt Creator can use the following kits for project **QtHelloWorld**:

- [✓] Desktop
  - [✓] Debug `_sdk/architech/pengwyn/workspace/qt/build-QtHelloWorld-Desktop-Debug`
  - [✓] Release `sdk/architech/pengwyn/workspace/qt/build-QtHelloWorld-Desktop-Release`
- [✓] pengwyn

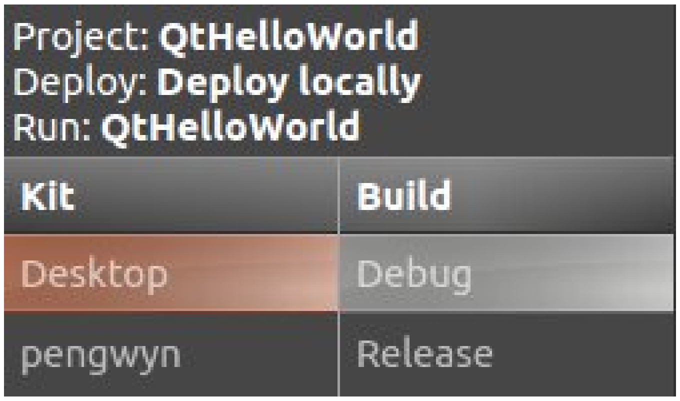
Note: Now you can edit your application adding labels and more, how to do this is not the purpose of this guide.

4. To compile the project click on "QtHelloWorld" icon to open project menu.



Project: QtHelloWorld
Kit:
Build: Debug
Deploy: Deploy to Remote Linux Host
Run: QtHelloWorld (on Remote Device)

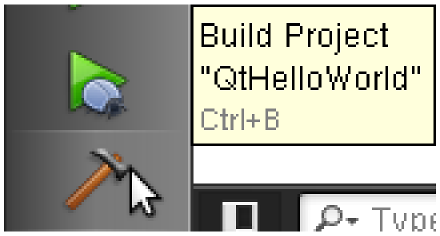
5. Select the build configuration: **Desktop - Debug**.



Project: **QtHelloWorld**
Deploy: **Deploy locally**
Run: **QtHelloWorld**

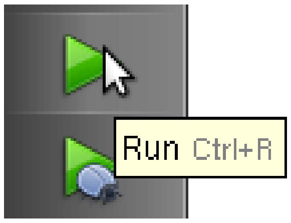
Kit	Build
Desktop	Debug
pengwyn	Release

6. To build the project, click on the bottom-left icon.



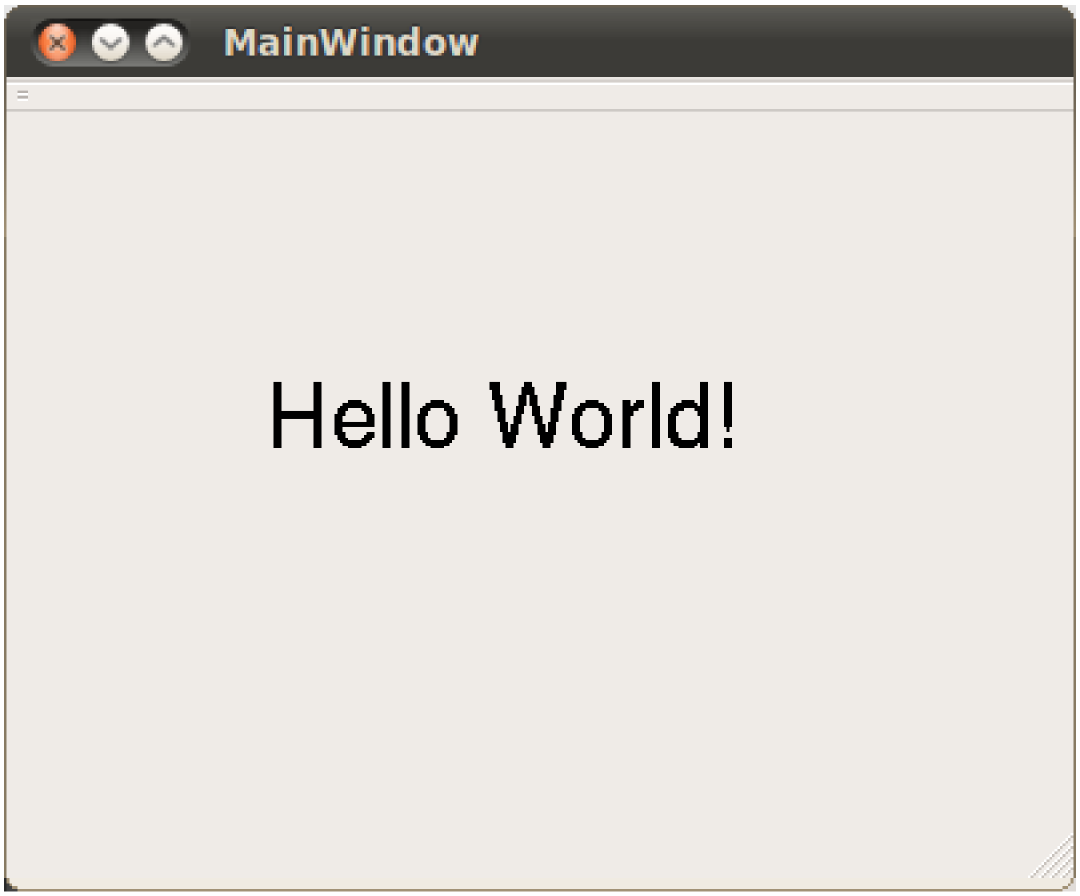
Build Project "QtHelloWorld" Ctrl+B

7. Once you built the project, click on the green triangle to run it.



Run Ctrl+R

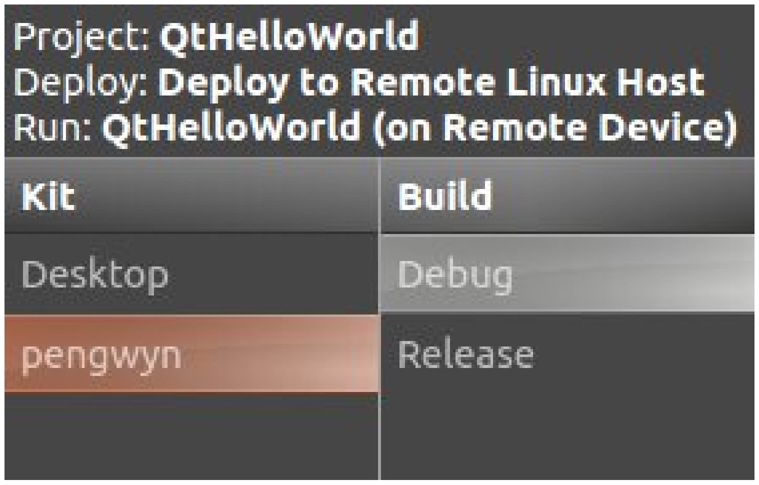
8. Congratulations! You just built your first Qt application for x86.



In the next section we will debug our Hello World! application directly on Pengwyn.

Debug Hello World project

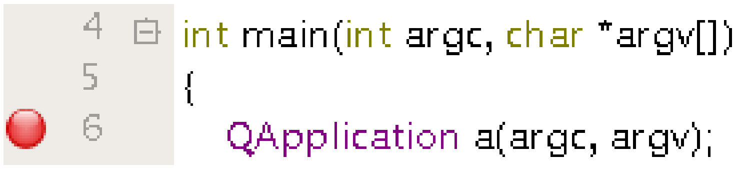
1. Select build configuration: **pengwyn - Debug** and build the project.


Project: QtHelloWorld
Deploy: Deploy to Remote Linux Host
Run: QtHelloWorld (on Remote Device)

Kit	Build
Desktop	Debug
pengwyn	Release

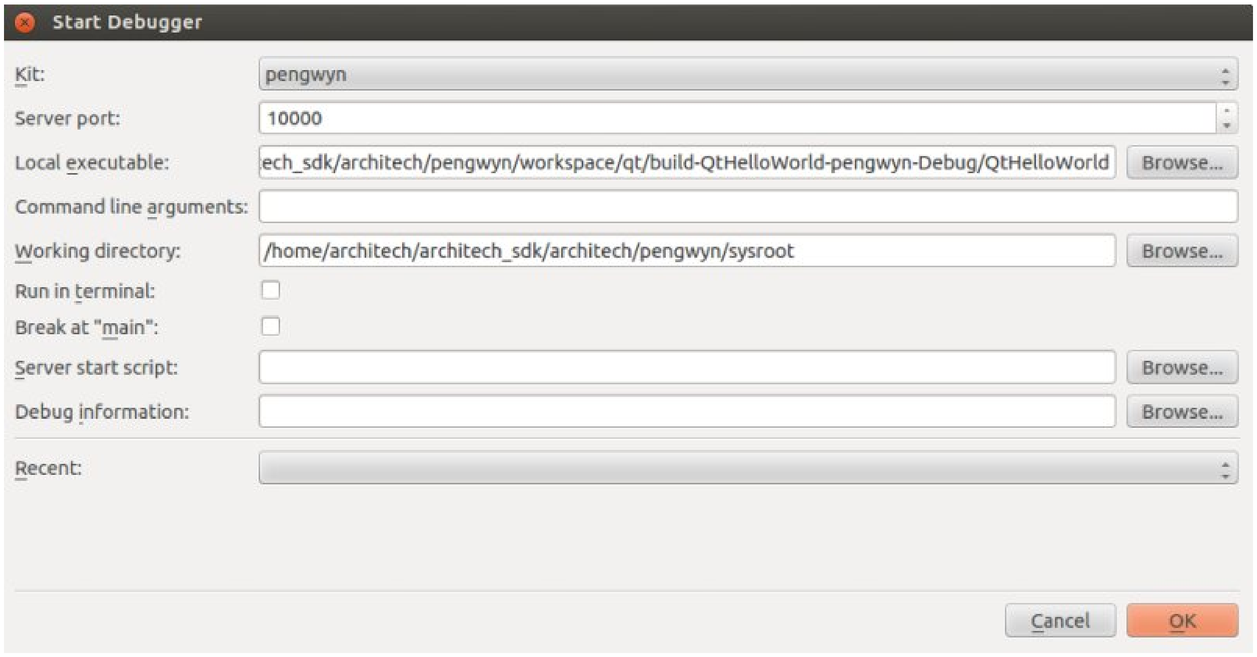
2. Copy the generated executable to the target board (e.g /home/root/).
3. Use minicom to launch gdbserver application on the target board:

4. In Qt Creator, open the source file main.cpp and set a breakpoint at line 6. | To do this go with the mouse at line 6 and click with the right button to open the menu, select **Set brackpoint at line 6**



```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
```

5. Go to *Debug→Start Debugging→Attach To Remote Debug Server*, a form named "Start Debugger" will appear, insert the following data:

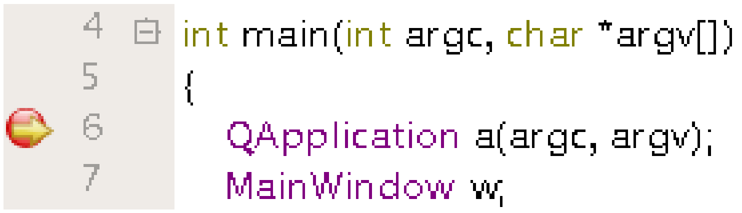


Start Debugger

Kit:	pengwyn
Server port:	10000
Local executable:	ech_sdk/architech/pengwyn/workspace/qt/build-QtHelloWorld-pengwyn-Debug/QtHelloWorld
Command line arguments:	
Working directory:	/home/architech/architech_sdk/architech/pengwyn/sysroot
Run in terminal:	☐
Break at "main":	☐
Server start script:	
Debug information:	
Recent:	

- Kit: **pengwyn**
- Local executable:

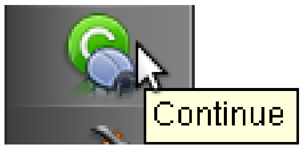
Press **OK** button to start the debug.



```
int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
```

6. The hotkeys to debug the application are:

- **F10**: Step over
- **F11**: Step into
- **Shift + F11**: Step out
- **F5**: Continue, or press this icon:



7. To successfully exit from the debug it is better to close the graphical application from the target board with the mouse by clicking on the 'X' symbol.

Cross compiler

Yocto/OpenEmbedded can be driven to generate the cross-toolchain for your platform. There are two common ways to get that:

or

The first method provides you the toolchain, you need to provide the file system to compile against, the second method provides both the toolchain and the file system along with -dev and -dbg packages installed.

Both ways you get an installation script.

The virtual machine has a cross-toolchain installed for each board, each generated with *meta-toolchain*. To use it just do:

to compile Linux user-space stuff. If you want to compile kernel or bootloader then do:

and you are ready to go.

Opkg

Opkg (Open PacKaGe Management) is a lightweight package management system. It is written in C and resembles apt/dpkg in operation. It is intended for use on embedded Linux devices and is used in this capacity in the OpenEmbedded and OpenWrt projects.

Useful commands:

- update the list of available packages:
- list available packages:
- list installed packages:
- install packages:
- list package providing <file>
- Show package information

- show package dependencies:
- remove packages:

Force Bitbake to install Opkg in the final image

With some images, *Bitbake* (e.g. *core-image-minimal*) does not install the package management system in the final target. To force *Bitbake* to include it in the next build, edit your configuration file

and add this line to it:

Create a repository

opkg reads the list of packages repositories in configuration files located under */etc/opkg/*. You can easily setup a new repository for your custom builds:

1. Install a web server on your machine, for example **apache2**:
2. Configure apache web server to "see" the packages you built, for example:
3. Create a new configuration file on the target (for example */etc/opkg/my_packages.conf*) containing lines like this one to index the packages related to a particular machine:

To actually reach the virtual machine we set up a port forwarding mechanism in Chapter *Virtual Machine* so that every time the board communicates with the workstation on port 8000, VirtualBox actually turns the communication directly to the virtual machine operating system on port 80 where it finds *apache* waiting for it.

4. Connect the board and the personal computer you are developing on by means of an ethernet cable
5. Update the list of available packages on the target

Update repository index

Sometimes, you need to force bitbake to rebuild the index of packages by means of:

The board

This chapter introduces the board, its hardware and how to boot it.

Hardware

The hardware documentation of Pengwyn can be found here:

http://downloads.architechboards.com/doc/Pengwyn/download.html

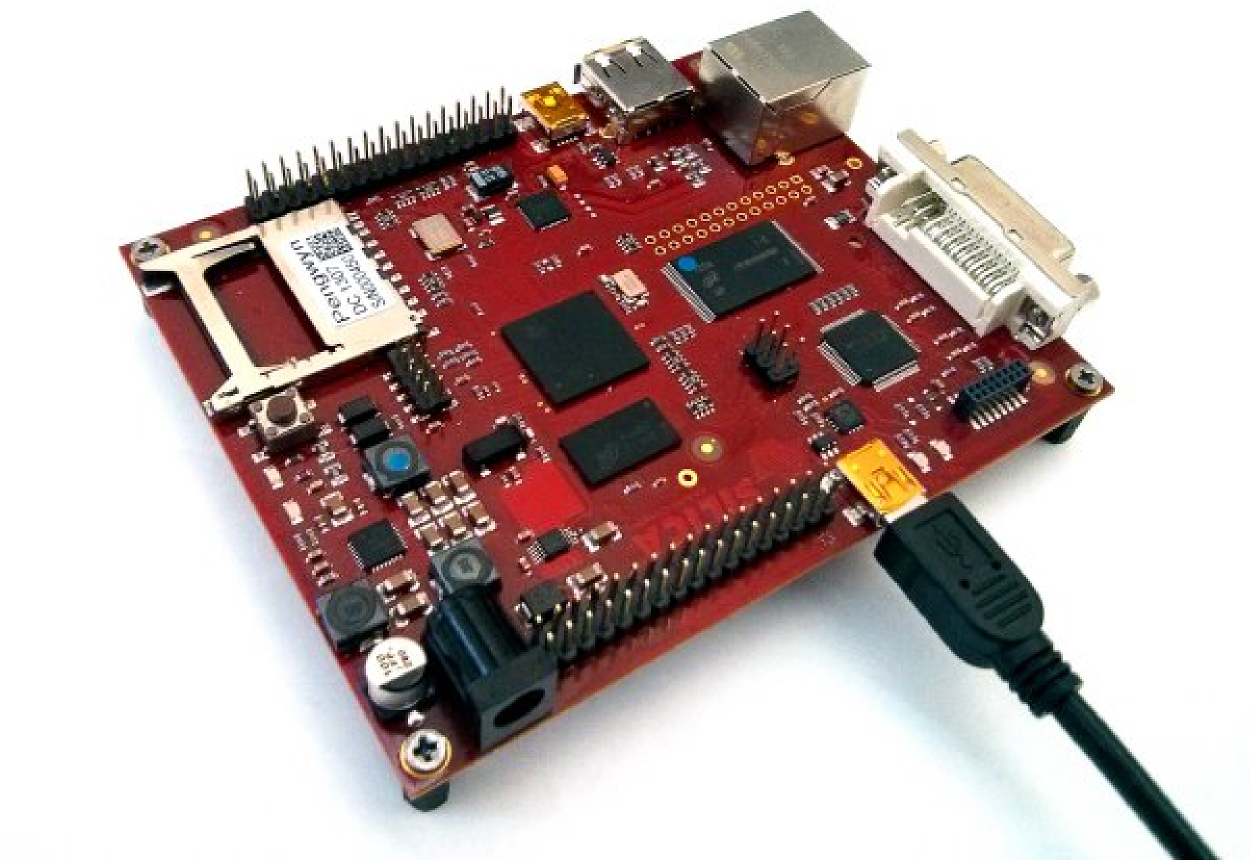
Power-On

Pengwyn takes the power from the mini-USB connector *CN6* and/or connector **CN1**. The board is not shipped with an external power adapter.

On connector *CN6* you can also have the serial console, so, during your daily development use, you would just connect your workstation to the board using a mini-USB to connector *CN6*. If you connect some power hungry device to the board, you can give more power to the board by connecting the power adapter.

Serial Console

On Pengwyn you can use the same USB cable used to power up the board to get access to the serial console.
The serial console connector **CN6**



which you can connect, by means of a mini-USB cable, to your personal computer.

Note: Every operating system has its own killer application to give you a serial terminal interface. In this guide, we are assuming your **host** operating system is **Ubuntu**.

On a Linux (Ubuntu) host machine, the console is seen as a *ttyUSB***X** device (where X is a number) and you can access to it by means of an application like *minicom*.

Minicom needs to know the name of the serial device. The simplest way for you to discover the name of the device is by looking to the kernel messages, so:

1. clean the kernel messages
2. connect the mini-USB cable to the board already powered-on
3. display the kernel messages
3. read the output

As you can see, here the device has been recognized as */dev/ttyUSB0*.

Now that you know the device name, run minicom:

If minicom is not installed, you can install it with:

then you can setup your port with these parameters:

If on your system the device has not been recognized as */dev/ttyUSB0*, just replace */dev/ttyUSB0* with the proper device.

Once you are done configuring the serial port, you are back to minicom main menu and you can select *exit*.

Let's boot

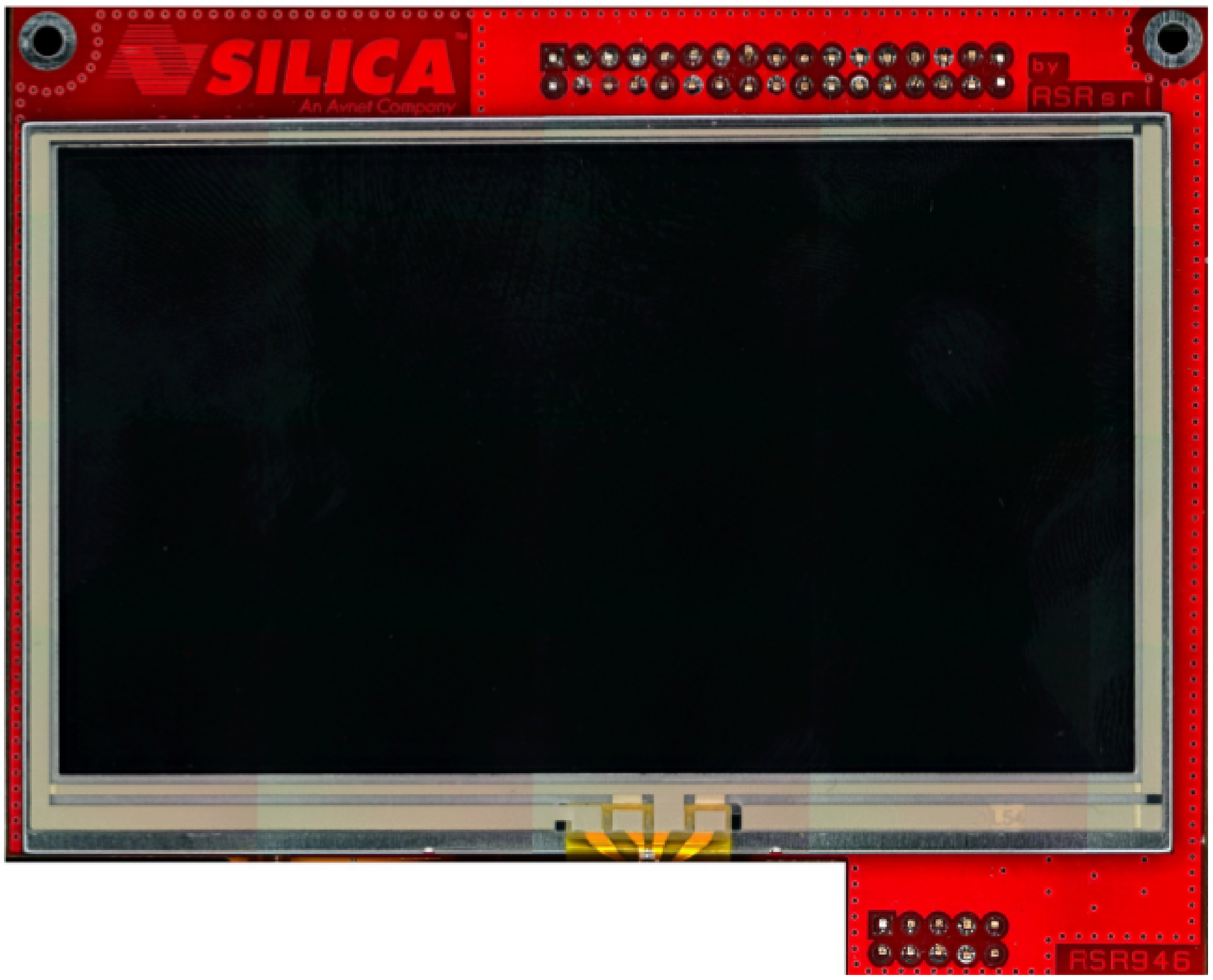
The boot process of the pengwyn is selected by jumpers **J1**, **J2** and **J3**. After a Power On Reset (POR) the processor starts executing the internal ROM program. The boot mode is based on information gathered from the **SYSBOOT**:

Jumper settings

			boot sequence			
J1	J2	J3	1st	2nd	3rd	4th
open	open	open	MMC0	SPI0	UART0	USB0
open	open	close	EMAC1	MMC0	XIP MUX2	NAND
open	close	open	Fast ext	EMAC1	UART0	Reserved
open	close	close	UART0	EMAC1	Reserved	Reserved
close	open	open	NAND	NANDI2C	MMC0	UART0
close	open	close	UART0	SPI0	XIP MUX2	MMC0
close	close	open	XIP MUX2	UART0	SPI0	MMC0
close	close	close	USB0	NAND	SPI0	MMC0

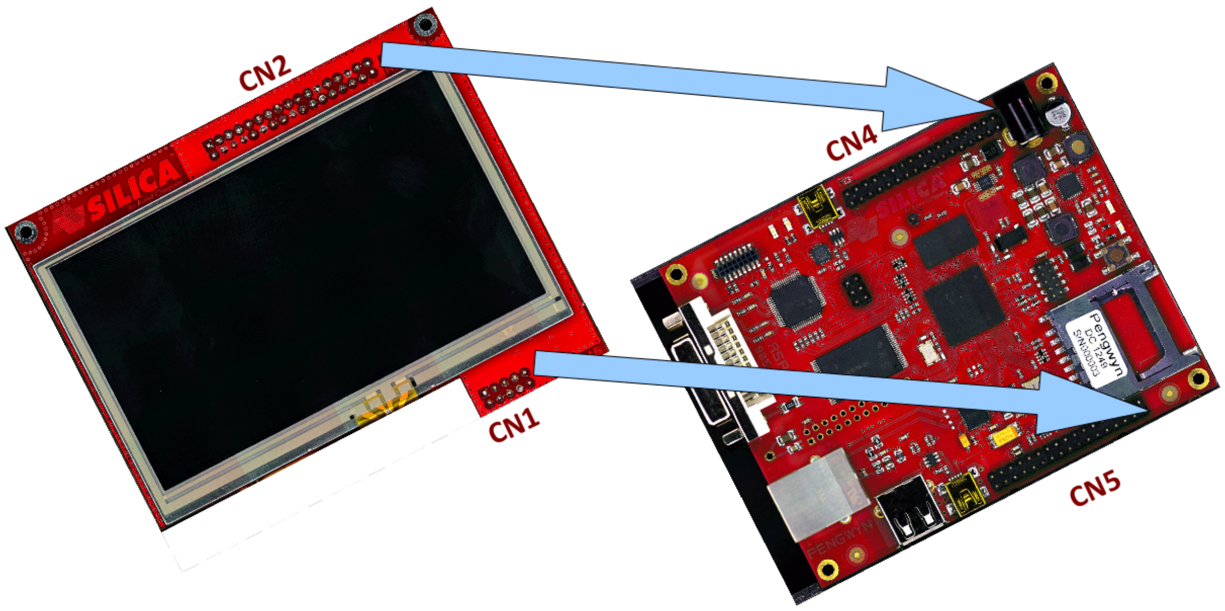
Touch Screen

This procedure will guide you to the installation of the display on the Pengwyn board and the configuration of the software to test it.



Installing the board

1. switch off the board
2. connect display



3. switch on the board without SD card

Installing the software

If you don't have a SD card formatted with 2 partitions, one for the boot and one for the root filesystem, create it as in *Deploy*. Now we want install in rootfs the qt4e-demo-image-pengwyn.tar.gz image.

And substitute the pengwyn.dtb with this one:

Make sure everything has been really written to the SD card:

Then insert SD card on Pengwyn board and wait Linux start-up. First time, **the touch screen calibration is needed**, than qt4 demo will start.

Network

Pengwyn networking is powered by TI's chip AM335x. Under Linux, instead, the default network configuration is:

If you want that configuration to be brought up at boot you can add a few line in file */etc/network/interfaces*, for example, if you want *eth0* to have a fixed ip address (say 192.168.0.10) and MAC address of value 1e:ed:19:27:1a:b6 you could add the following lines:

FAQ

Virtual Machine

What is the password for the default user of the virtual machine?

The password for the default user, that is **architech**, is:

Host

```
architech
```

What is the password of sudo?

The default password of **architech** is **architech**. If you are searching more information about **sudo** command please refer to *sudo* section of the *appendix*.

What is the password for user root?

By default, Ubuntu 12.04 32bit comes with no password defined for **root** user, to set it run the following command:

Host

```
sudo passwd root
```

Linux will ask you (twice, the second time is just for confirmation) to write the password for user root.

What are device files? How can I use them?

Please refer to *device files* section of the *appendix*.

I have problems to download the vm, the server cut down the connection

The site has limitation in bandwith. Use download manager and do not try to speed up the download. If you try to download fastly the server will broke up your download.

Pengwyn

Appendix

In this page you can find some useful info about how Linux works. If you are coming from Microsoft world, the next paragraphs can help you to have a more soft approach to Linux world.

sudo command

sudo is a program for Unix-like computer operating systems that allows users to run programs/commands with the security privileges of another user, normally the superuser or root. Not all the users can call sudo, only the **sudoers**, **architech** (the default user of the virtual machine) user is a sudoer. When you run a command preceeded by sudo Linux will ask you the user password, for **architech** user the password is **architech**.

Device files

Under Linux, (almost) all hardware devices are treated as files. A device file is a special file which allows users to access an hardware device by means of the standard file operations (open, read, write, close, etc), hiding hardware details. All device files are in */dev* directory. In order to access a filesystem in Linux you first need to mount it. Mounting a filesystem simply means making the particular filesystem accessible at a certain point in the Linux directories tree.

In Linux, memory cards are generally named starting with *mmcblk*. For example if you insert 2 memory cards in 2 different slots of the same computer, Linux will create 2 device files:

The number identifies a specific memory card. A memory card itself can have one or more partitions. Even in this case, Linux will create a device file for every partition present in the sd card. So, for example if the "mmcblk0" countains 3 partitions, the operating system will add these files under */dev* directory:

Not all devices are named according to the aforementioned naming scheme. For example, usb pens and hard disks are named with *sd* followed by a letter which is incremented every time a new device gets connected (starting with *a*), as opposed to the naming scheme adopted by SD cards where a number (starting with *0*) was incremented. A machine with an hard disk and two pen drives would tipically have the following devices:

Usually */dev/sda* file is the primary hard disk (this might depend on your hardware).

As memory cards, the pen can have one or more partitions, so if for example we have a pen drive which has been recognized as *sdc*, and the pen drive has 2 partitions on it, we will have the following device files:

Commands like mount, umount, dd, etc., use partition device files. FIXME mkfs

Warning:

Be very careful when addressing device files, addressing the wrong one may cost you the loss of important data

Disks discovery

When dealing with plug and play devices, it is quite comfortable to take advantage of **dmesg** command. The kernel messages (printk) are arranged into a ring buffer, which the user can be easly access by means of **dmesg** command. Every time the kernel recognizes new hardware, it prints information about the new device within the ring buffer, along with the device filename. To better filter out the information regarding the plug and play device we are interested in, it is better if we first clean up the ring buffer:

now that the ring buffer has been emptied, we can plug the device and, after that, display the latest messages from the kernel:

On the Ubuntu machine (with kernel version *3.2.0-65-generic*) this documentation has been written with, we observed the following messages after inserting a pen drive:

As you can see, the operating system have recognized the usb device as *sdb* (this translates to */dev/sdb*) and its only partition as *sdb1* (this translates to */dev/sdb1*)

The most useful command to gather information about mass storage devices and related partitions is **fdisk**. On the very same machine of the previous example, the execution of this command:

produces the following output:

The machine has two mass storage devices, a 500GB hard disk and a 1GB USB pen disk. As you can see from the output, *sudo fdisk -l* command lists information regarding the disks seen by the kernel along with the partitions found on them, disk after disk. The first disk (sda) presented by *fdisk* is the primary hard disk (where Linux is running), it has 4 partitions, two of which (sda1 and sda2) are used by a Microsoft operating system while the other two (sda3 and sda4) are used by a Linux operating system. The second disk (sdb) depicted by *fdisk* is an USB disk with a single FAT32 partition (sdb1)

As already stated, in order to access a filesystem in Linux you first need to mount it. Mounting a partition means binding a directory to it, so that files and directories contained inside the partition will be available in Linux filesystem starting from the directory used as mount point.

mount command

Suppose you want to read a file named *readme.txt* which is contained inside the USB disk of the previous example, in the main directory of the disk. Before accessing the device you must understand if it is already mounted. **mount** is the command that lets you control the mounting of filesystems in Linux. It is a complex command that permits to mount different devices and different filesystems. In this brief guide we are using it only for a very common use case. Launching **mount** without any parameter lists all mounted devices with their respective mounting points. Every line of the list, describes the name of the mounted device, where it has been mounted (path of the directory in the Linux filesystem, that is the mount point), the type of filesystem (ext3, ext4, etc.), and the options used to mount it (read and write permissions,etc.). Launching the command on the same machine of the previous section example, we don't find the device */dev/sdb1*.

```
$ mount
/dev/sda2 on /media/windows7 type fuseblk (rw,noexec,nosuid,nodev,allow_other,blksize=4096)
/dev/sda3 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
sysfs on /sys type sysfs (rw,noexec,nosuid,nodev)
none on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
udev on /dev type devtmpfs (rw,mode=0755)
devpts on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
tmpfs on /run type tmpfs (rw,noexec,nosuid,size=10%,mode=0755)
none on /run/lock type tmpfs (rw,noexec,nosuid,nodev,size=5242880)
none on /run/shm type tmpfs (rw,nosuid,nodev)
binfmt_misc on /proc/sys/fs/binfmt_misc type binfmt_misc (rw,noexec,nosuid,nodev)
rpc_pipefs on /run/rpc_pipefs type rpc_pipefs (rw)
vmware-vmblock on /run/vmblock-fuse type fuse.vmware-vmblock
(rw,nosuid,nodev,default_permissions,allow_other)
gvfs-fuse-daemon on /home/roberto/.gvfs type fuse.gvfs-fuse-daemon (rw,nosuid,nodev,user=roberto)
```

This tells us that the USB disk has not been mounted yet.

The mount operation requires three essential parameters: - the device to mount - the directory to associate - the type of filesystem used by the device

Thanks to the previously introduced **fdisk** command, we know the partition to mount (*/dev/sdb1*) and the type of filesystem used (FAT32). The directory to bind can be anything you like, by convention the user should mount his own devices under */media* or */mnt*. We haven't created it yet, so:

At this point, we have the information we need to execute the mounting. To semplify our life, we leave the duty of understanding what filesystem is effectively used by the device to the **mount** command by using option *-t auto* (if we would have wanted to tell mount exactly which filesystem to use we would have written *-t vfat*), like

The partition is now binded to */media/usbdisk* directory and its data are accessible from this directory.

now we can open the file, read it and, possibly, modify it.

When you want to disconnect the device, you need the inverse operation of **mount** which is **umount**. This command saves all data still contained in RAM (and waiting to be written on the device) and unbind the directory from the device file.

Once the directory */media/usbdisk* is unmounted it's empty, feel free to delete it if doesn't interest you anymore. It is now possible to remove the device from the machine.

What if you wanted to know the amount of free disk space available on a mounted device?

df command shows the disk space usage of all currently mounted partitions. For every partition, **df** prints its device file, size, free and used space, and the partition mount point. On our example machine we have:

-h option tells **df** to print sizes in human readable format.

Index