
Galileo Documentation

Release C

Architech

Mar 16, 2017

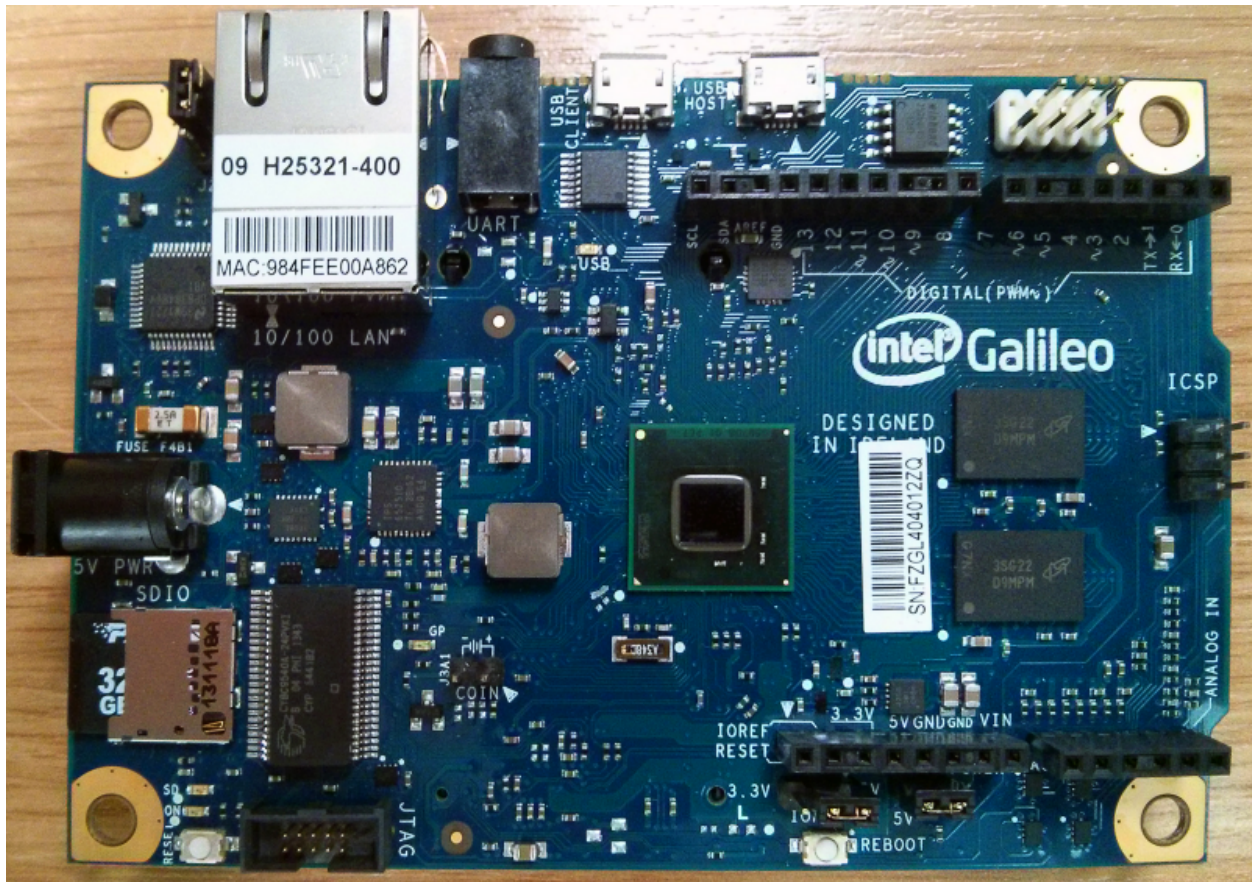
Contents

1	Notations	3
2	Chapters	5
2.1	Unboxing	5
2.2	Quick start guide	8
2.3	SDK Architecture	42
2.4	Create SDK	62
2.5	BSP	63
2.6	Toolchain	66
2.7	The board	80
2.8	FAQ	84
2.9	Appendix	85

Version 1.2.0C

Copyright (C)2016 Avnet Silica company

Date 22/01/2015



Welcome to **Galileo** documentation!

Have you just received your Galileo 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 Galileo 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

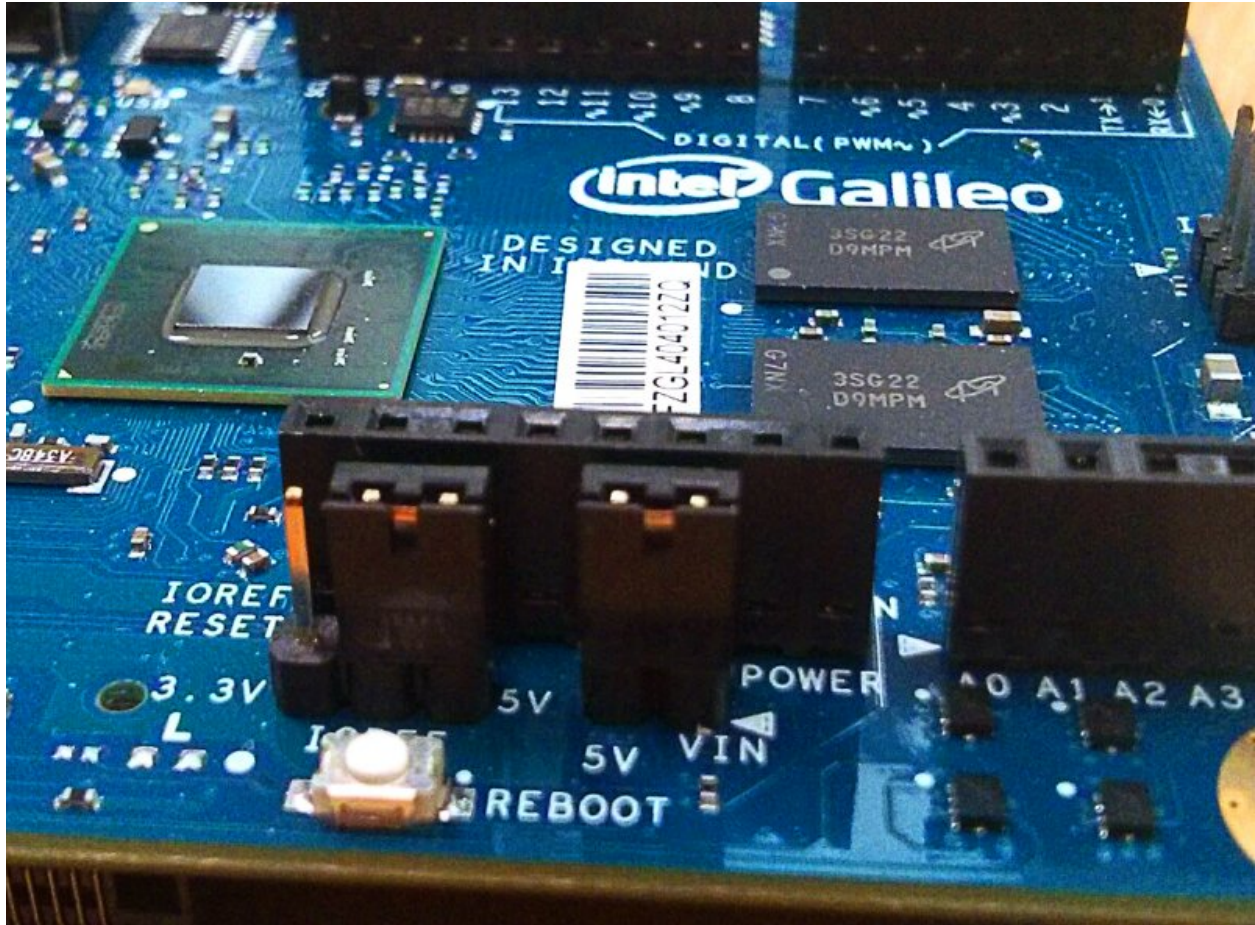
Intel Galileo board comes into this box



Galileo includes a “universal” wall adapter, which should work for just about anyone. If you’re using your own supply, make sure it can supply up to 3A. And remember that 5V is the max and min on that supply input.



Make sure IOREF has this configuration:



Read the *Quick start guide* Chapter to boot the board

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, Galileo 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>

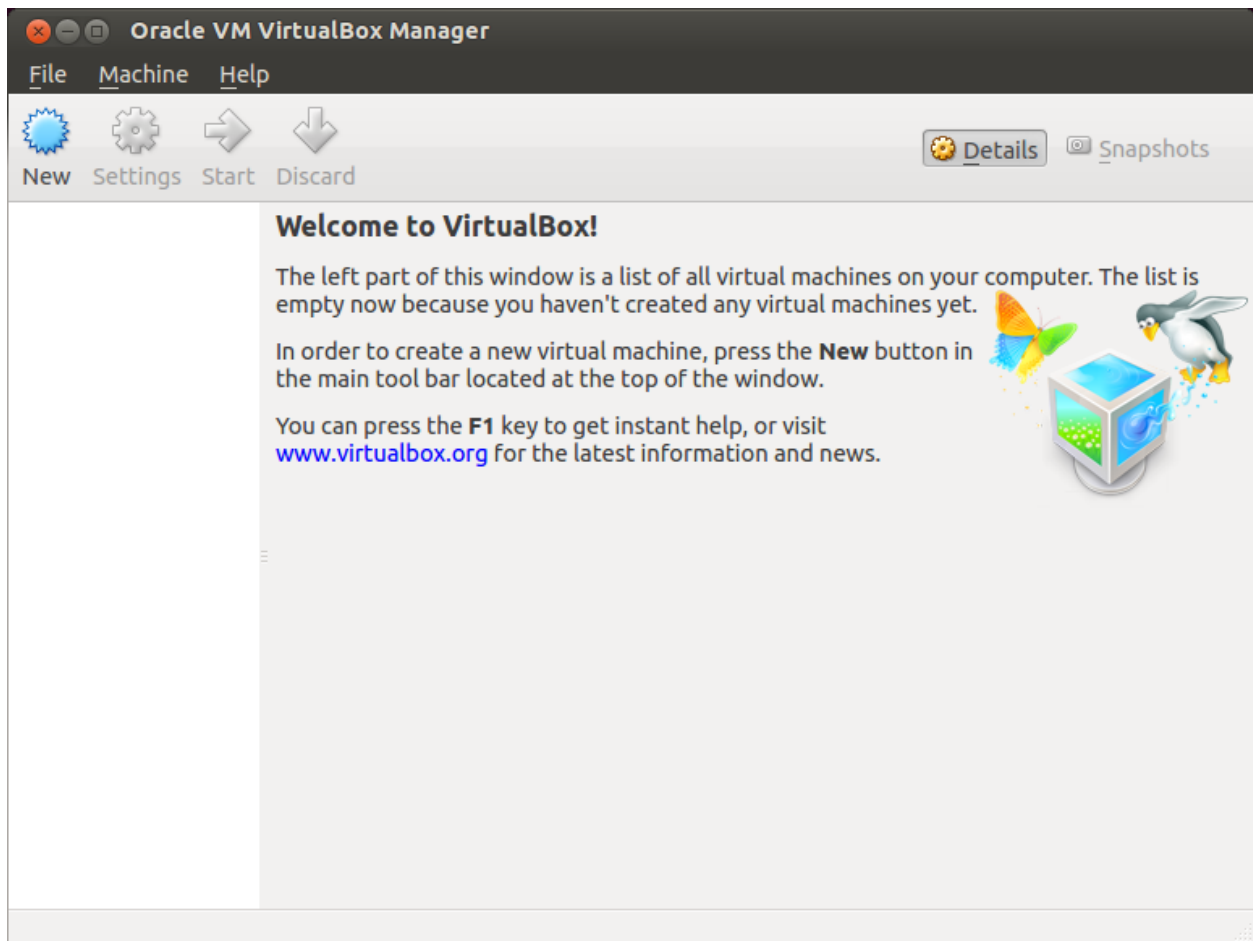
Download the version that suits your host operating system. You need to download and install the **Extension Pack** as well.

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

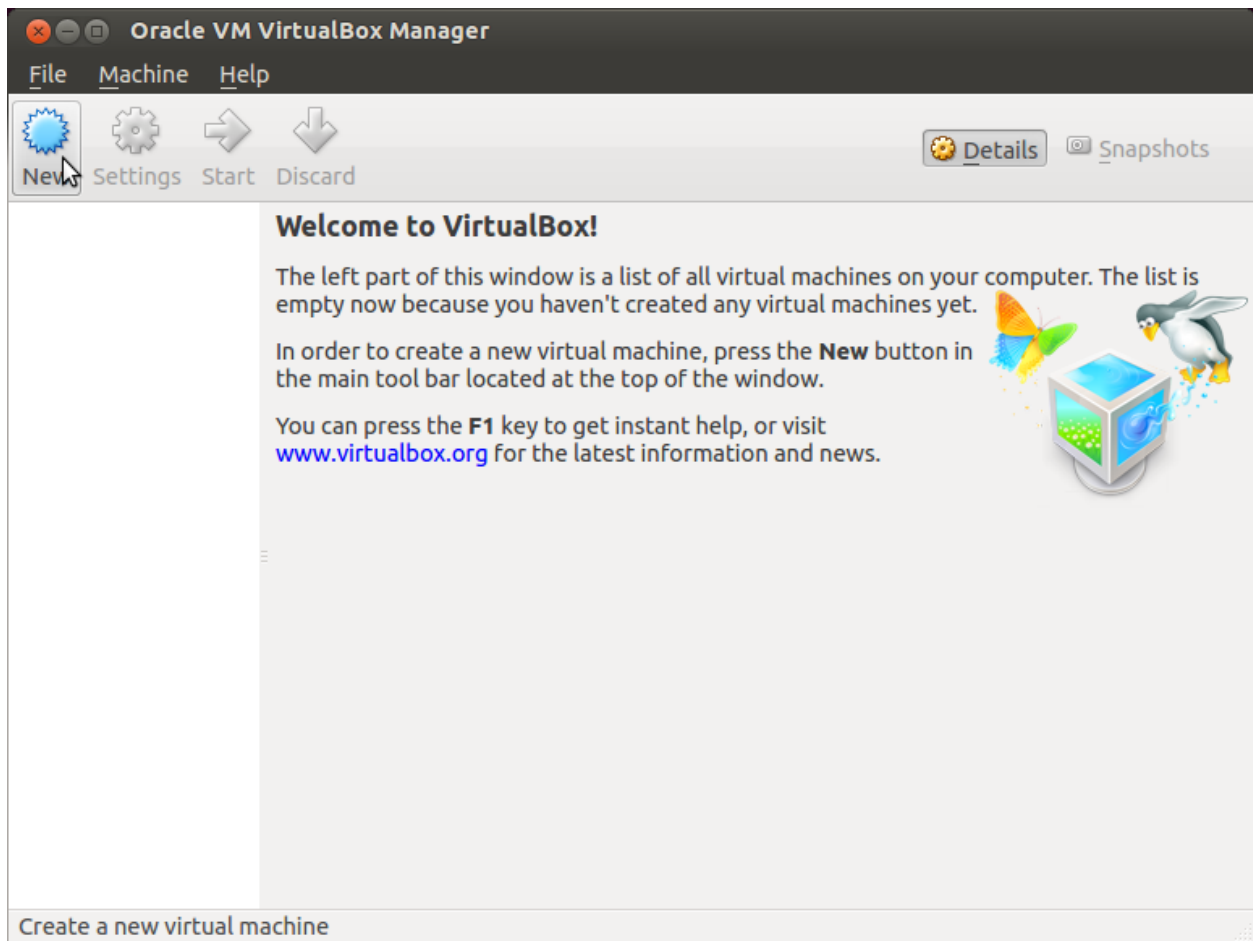
Install the software with all the default options.

Create a new Virtual Machine

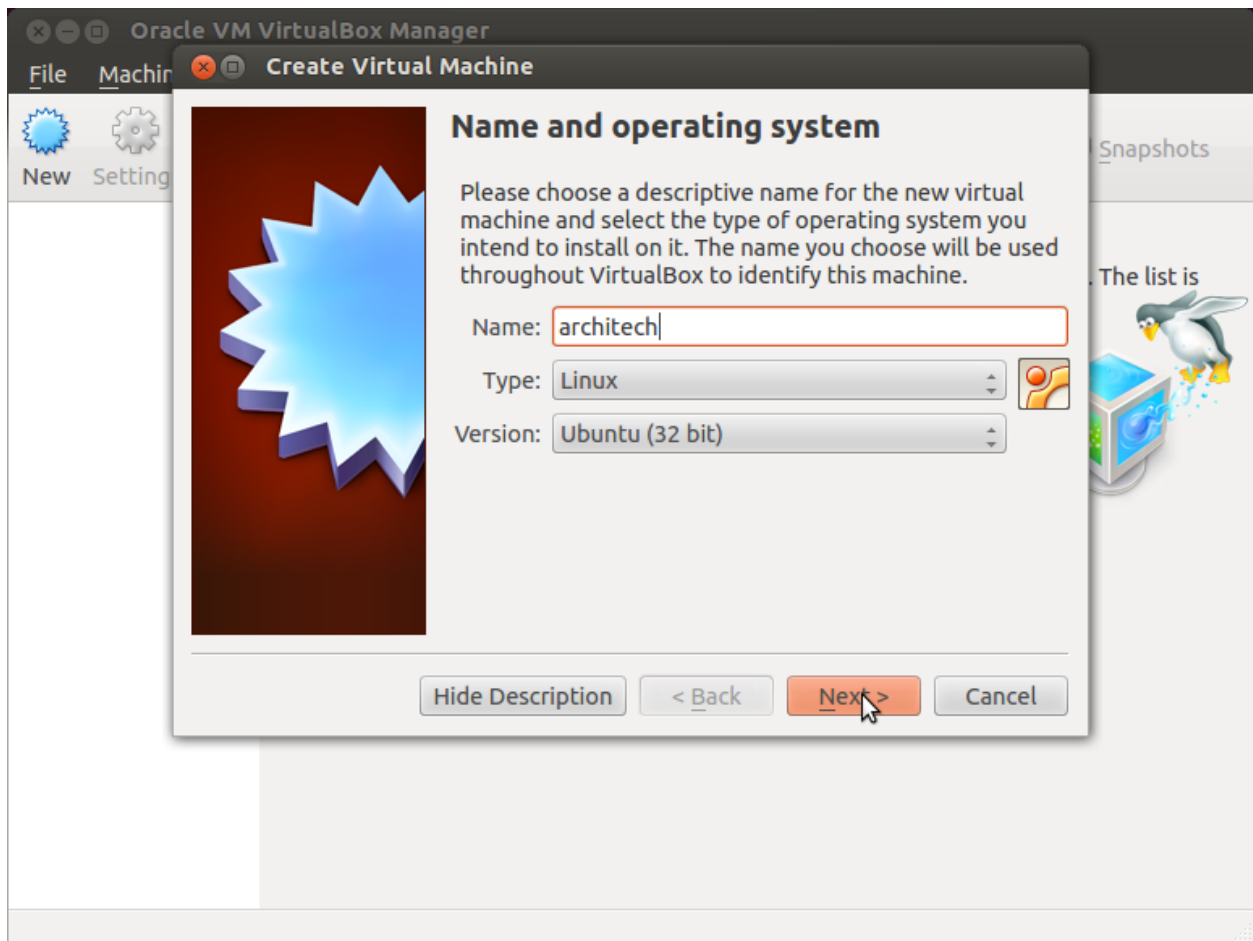
1. Run VirtualBox



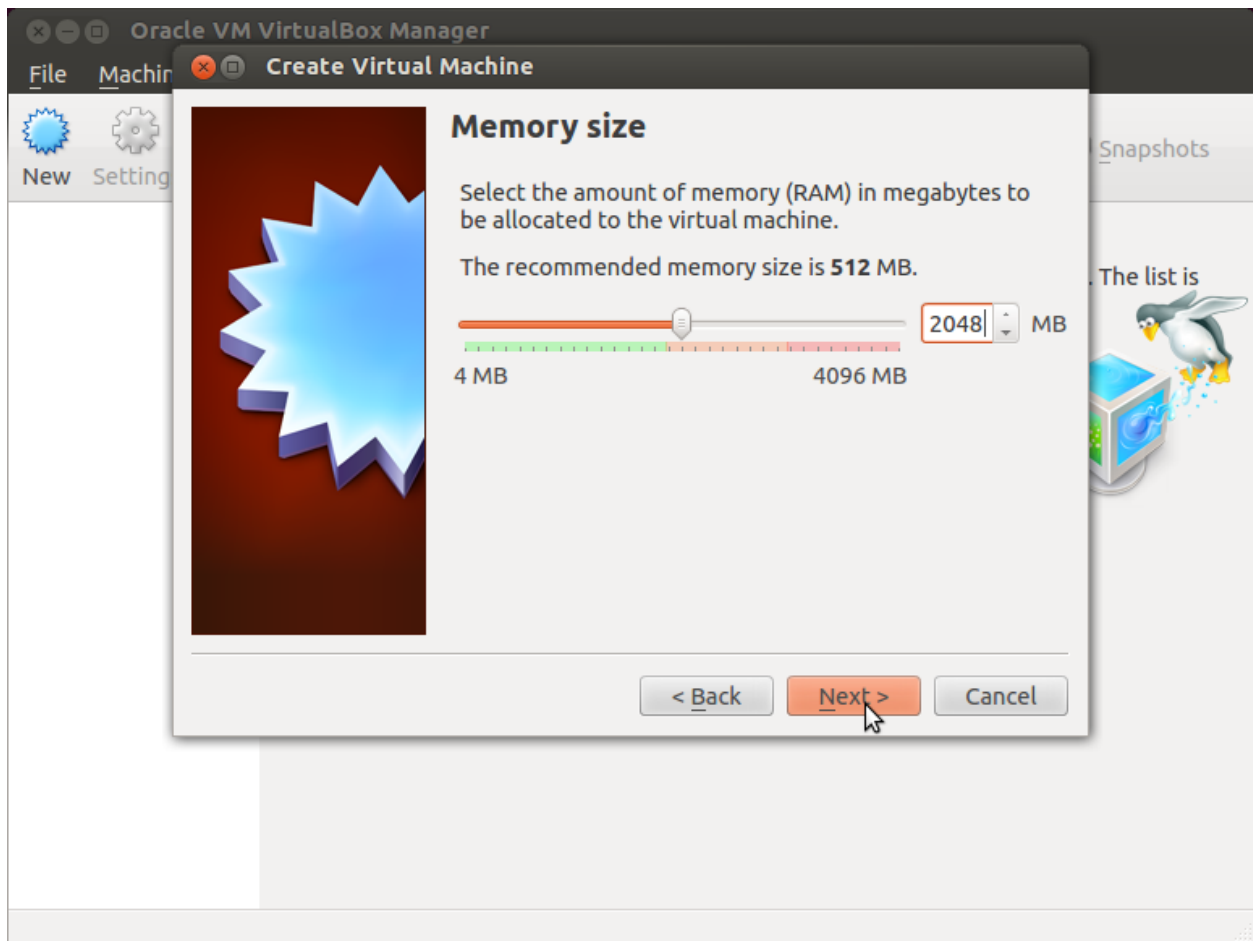
2. Click on *New* button



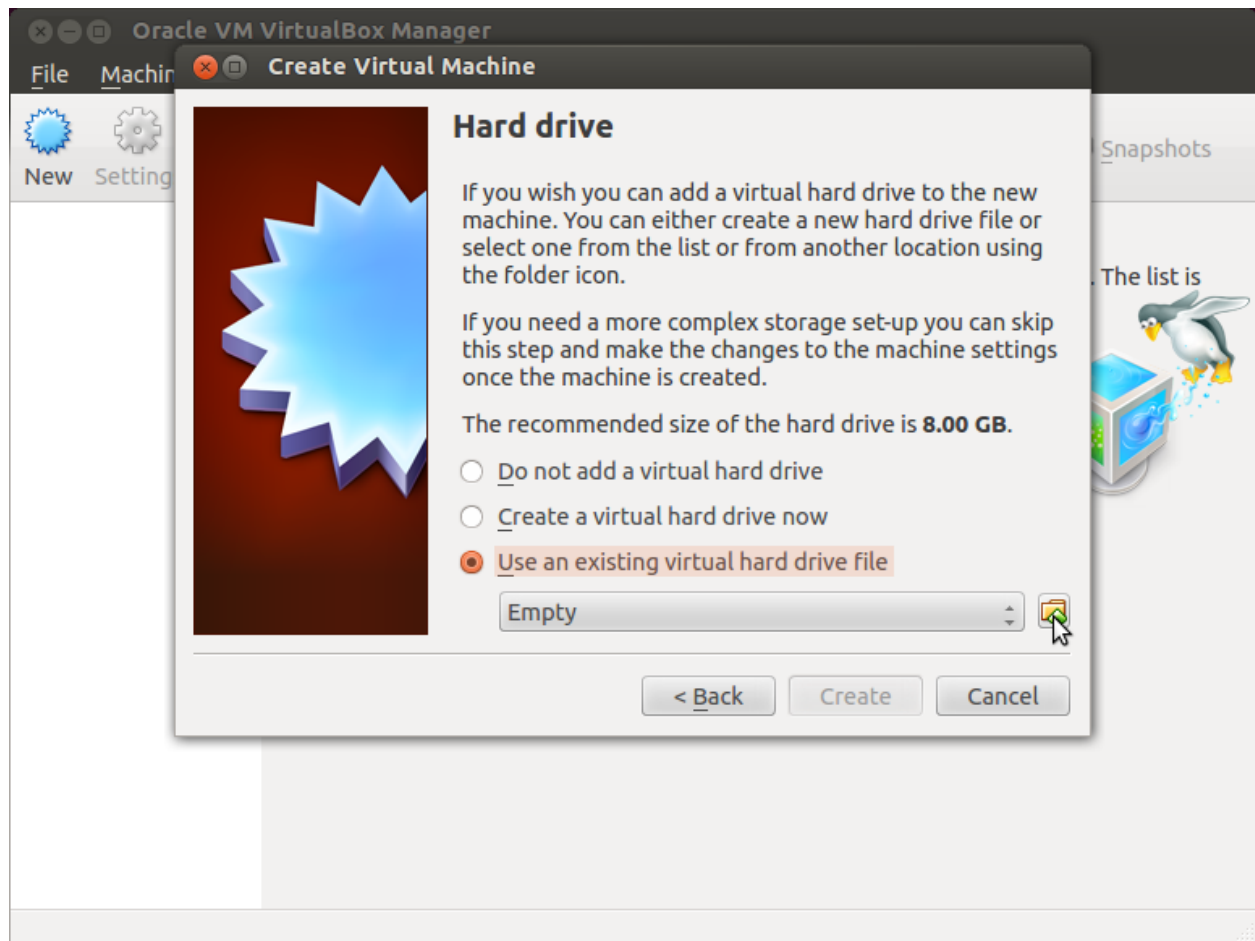
3. Select the name of the virtual machine and the operating system type



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



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

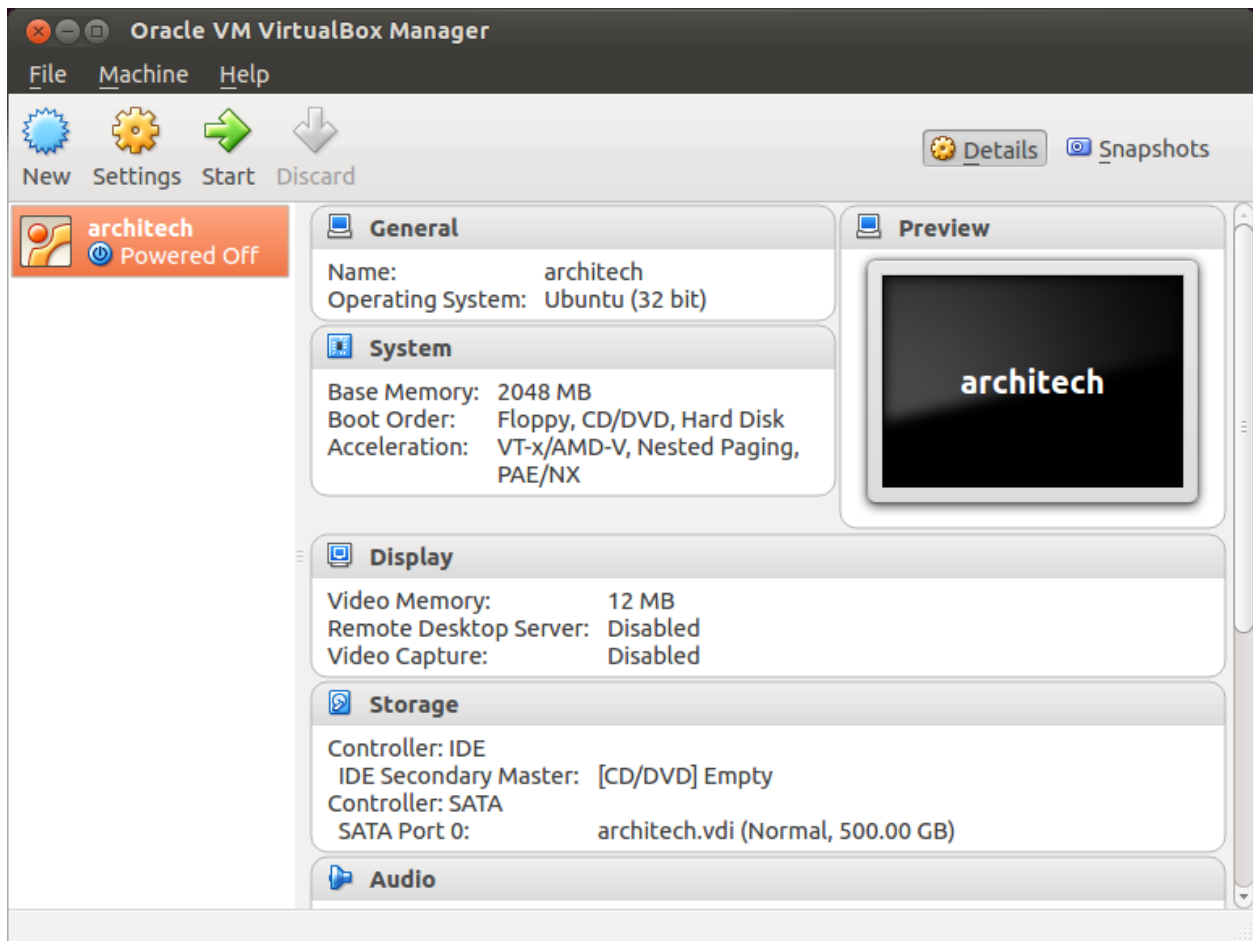


Setup the network

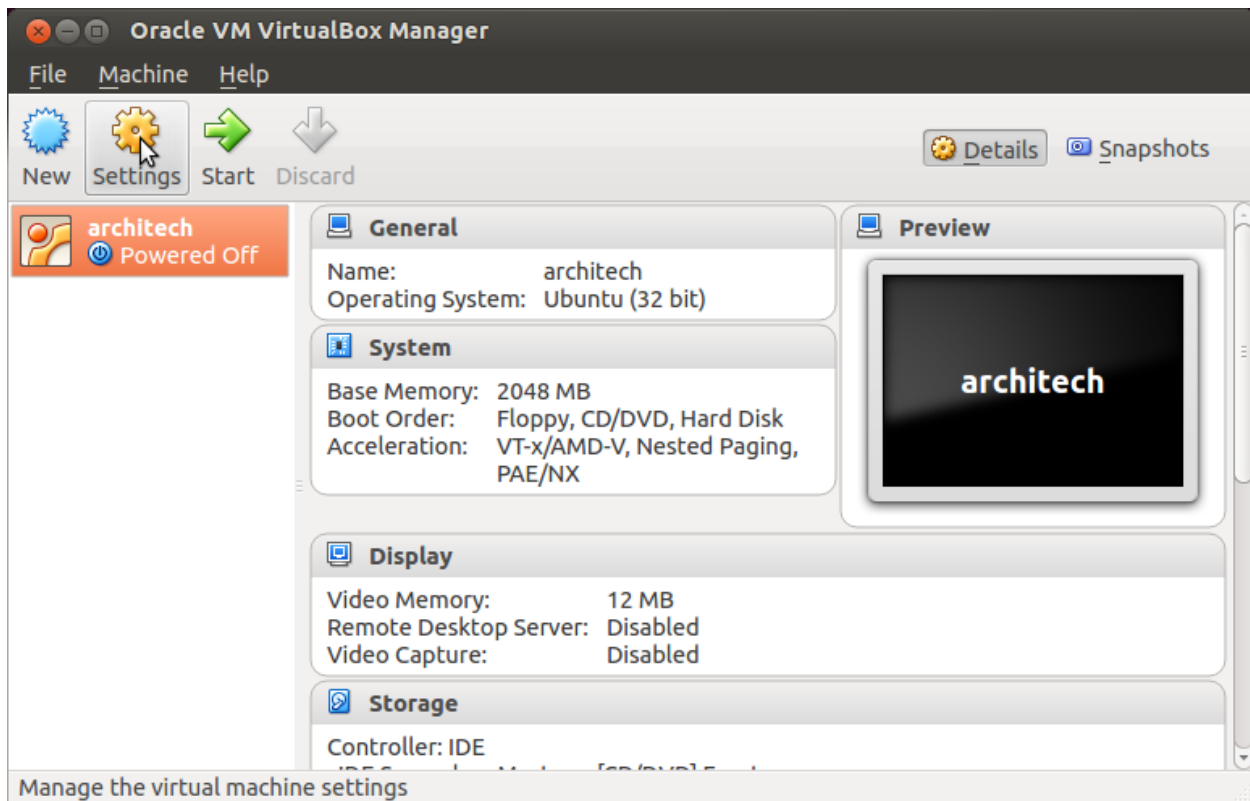
We need to setup a port forwarding rule to let you (later) use the virtual machine as a local repository of packages.

Note: The virtual machine must be off

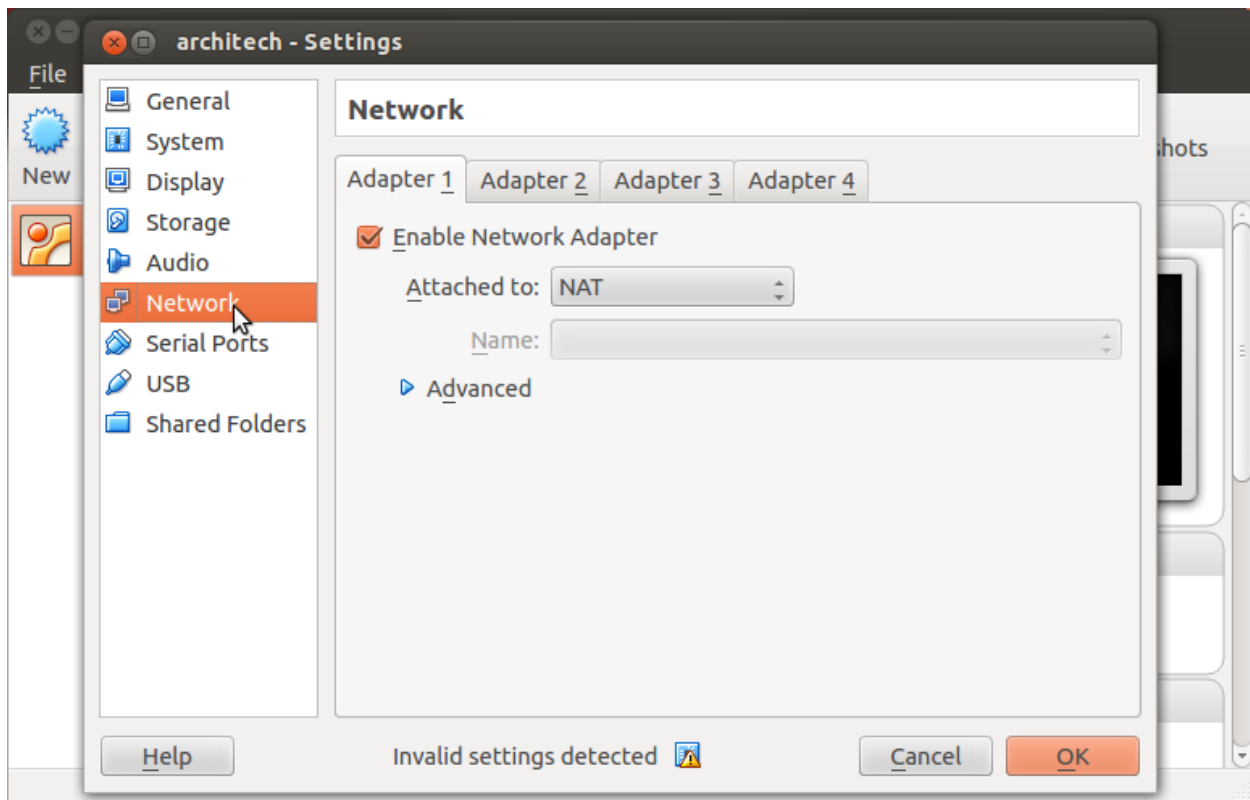
1. Select Architech's virtual machine from the list of virtual machines



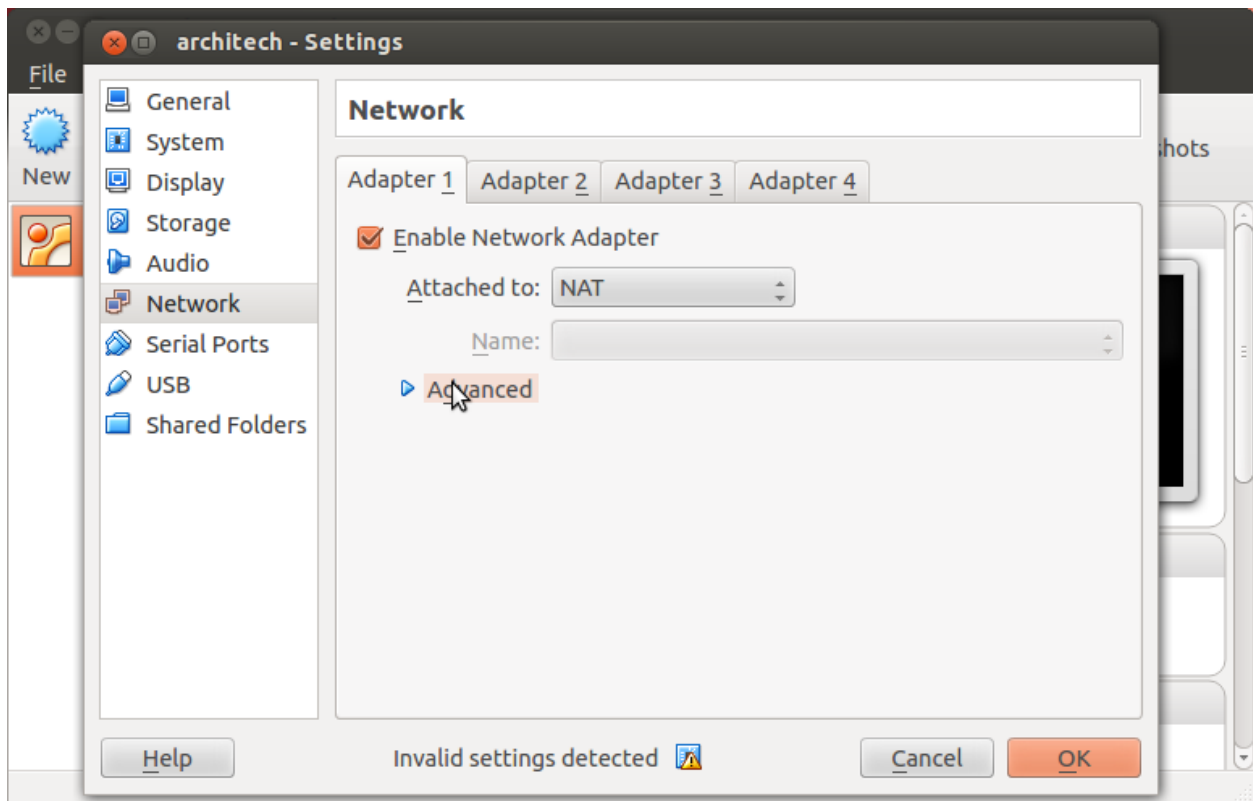
2. Click on *Settings*



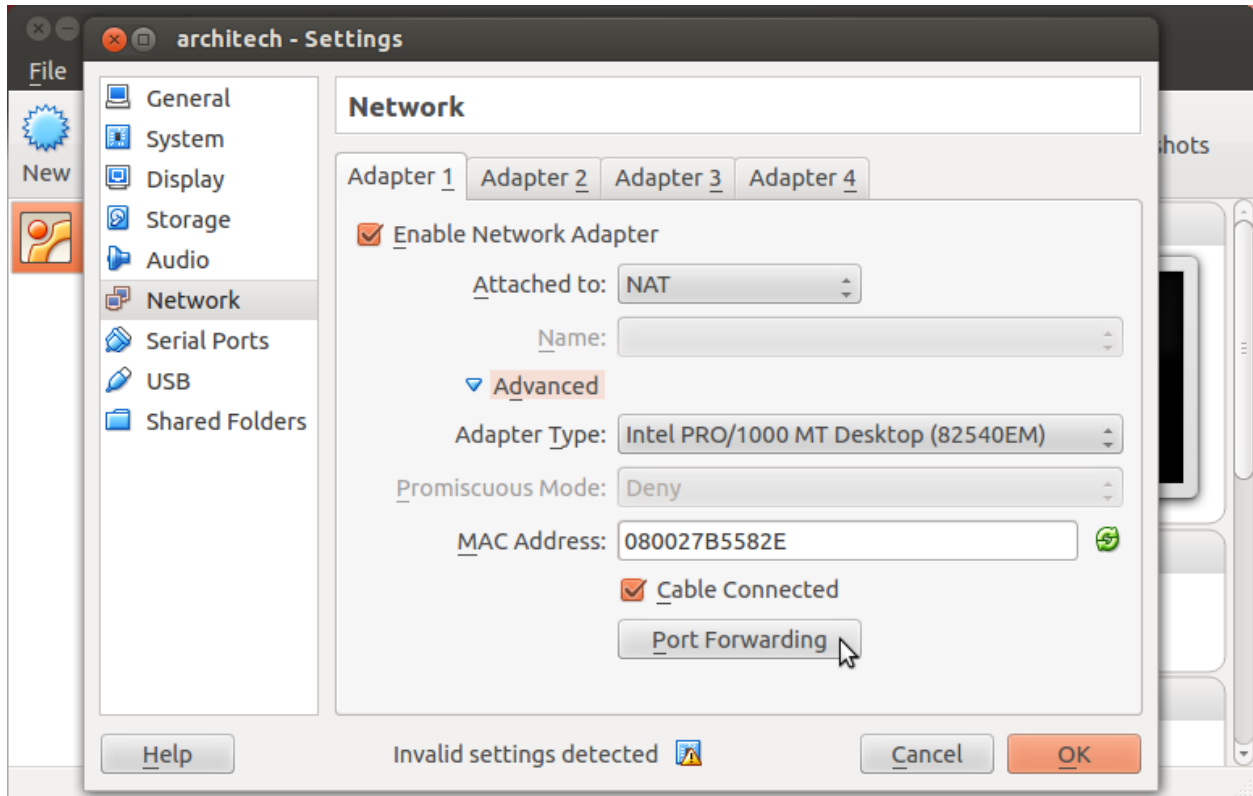
3. Select *Network*



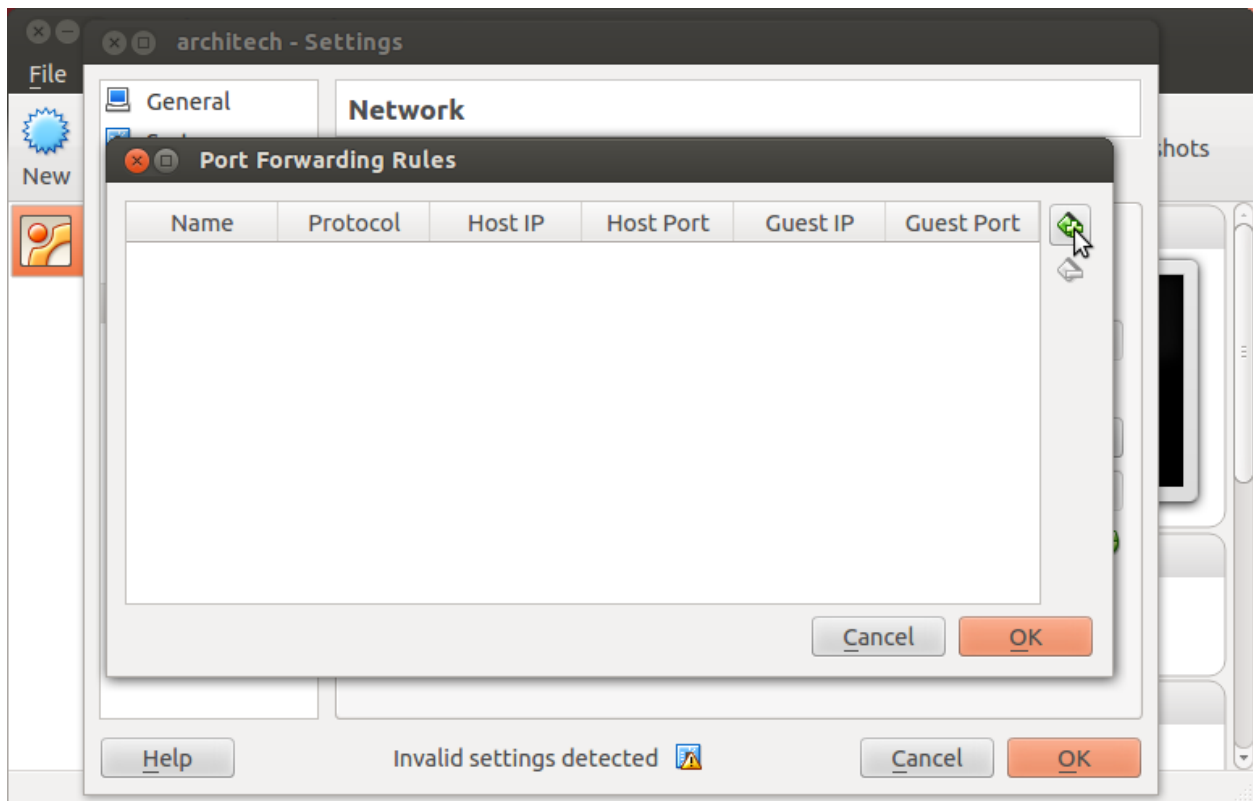
4. Expand *Advanced* of Adapter 1



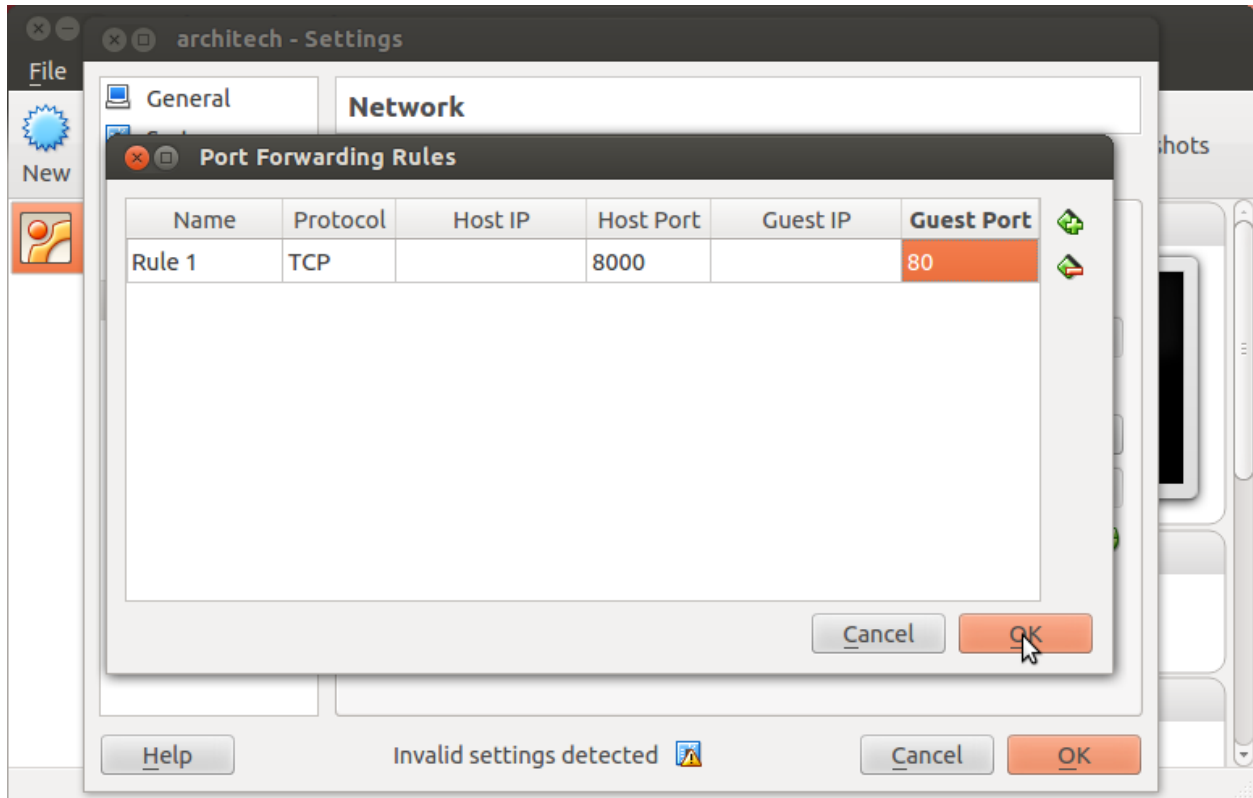
5. Click on *Port Forwarding*



6. Add a new *rule*



7. Configure the *rule*



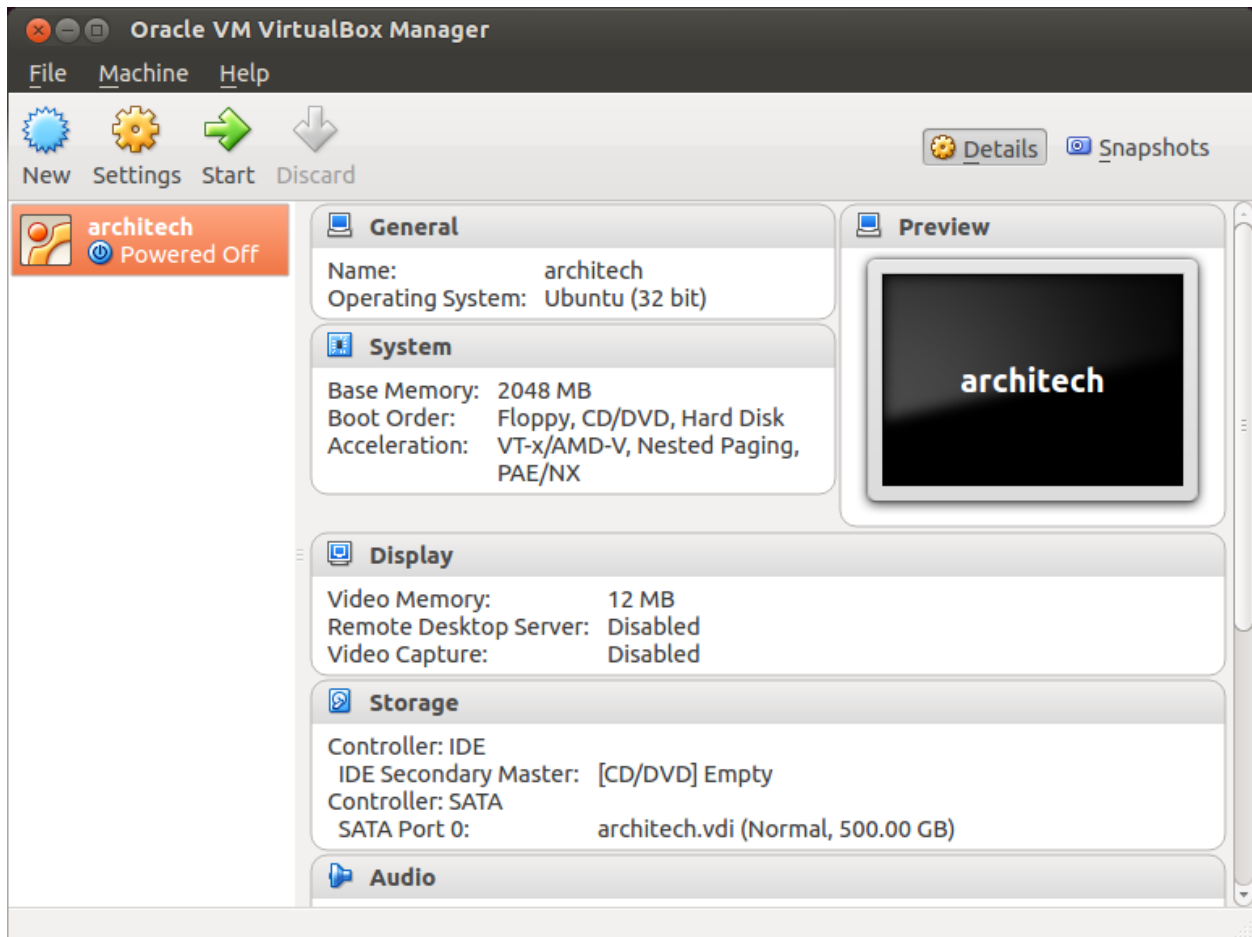
8. Click on *Ok*

Customize the number of processors

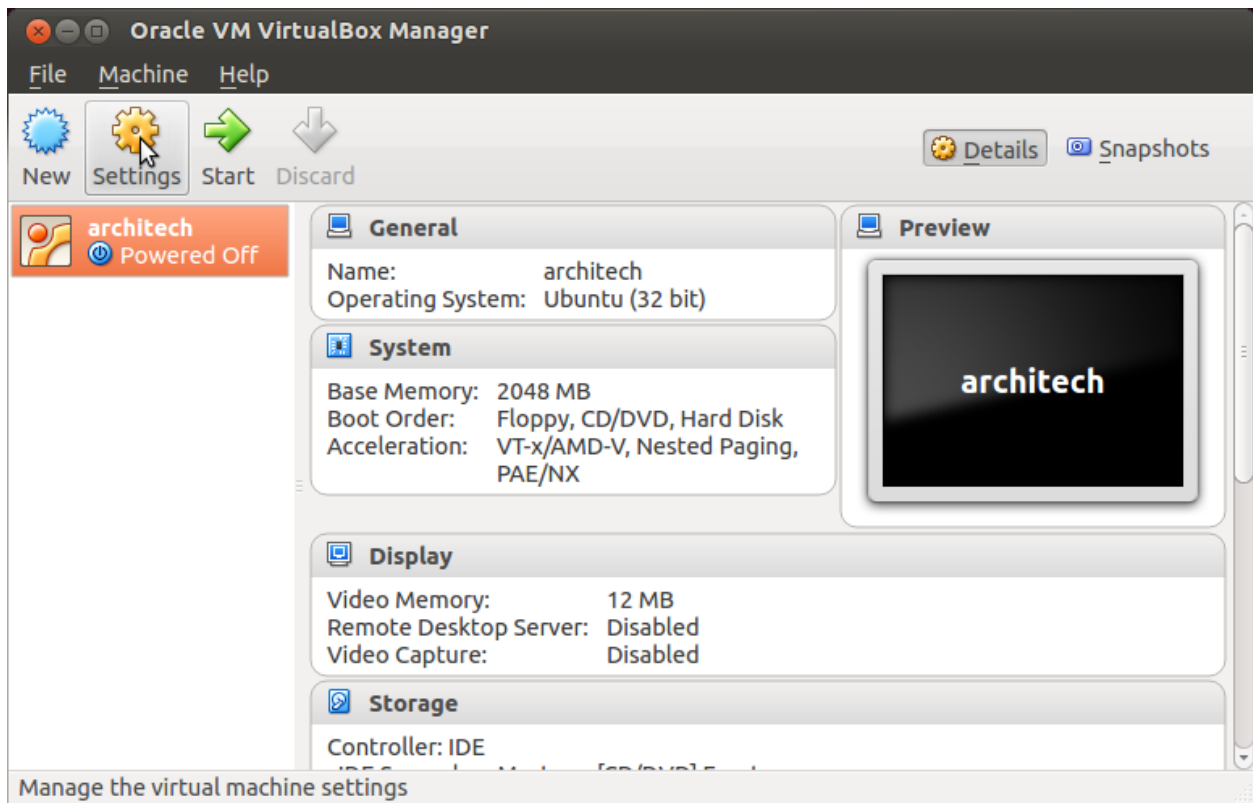
Building an entire system from the ground up is a business that can take up to several hours. To improve the performances of the overall build process, you can, if your computer has enough resources, assign more than one processor to the virtual machine.

Note: The virtual machine must be off

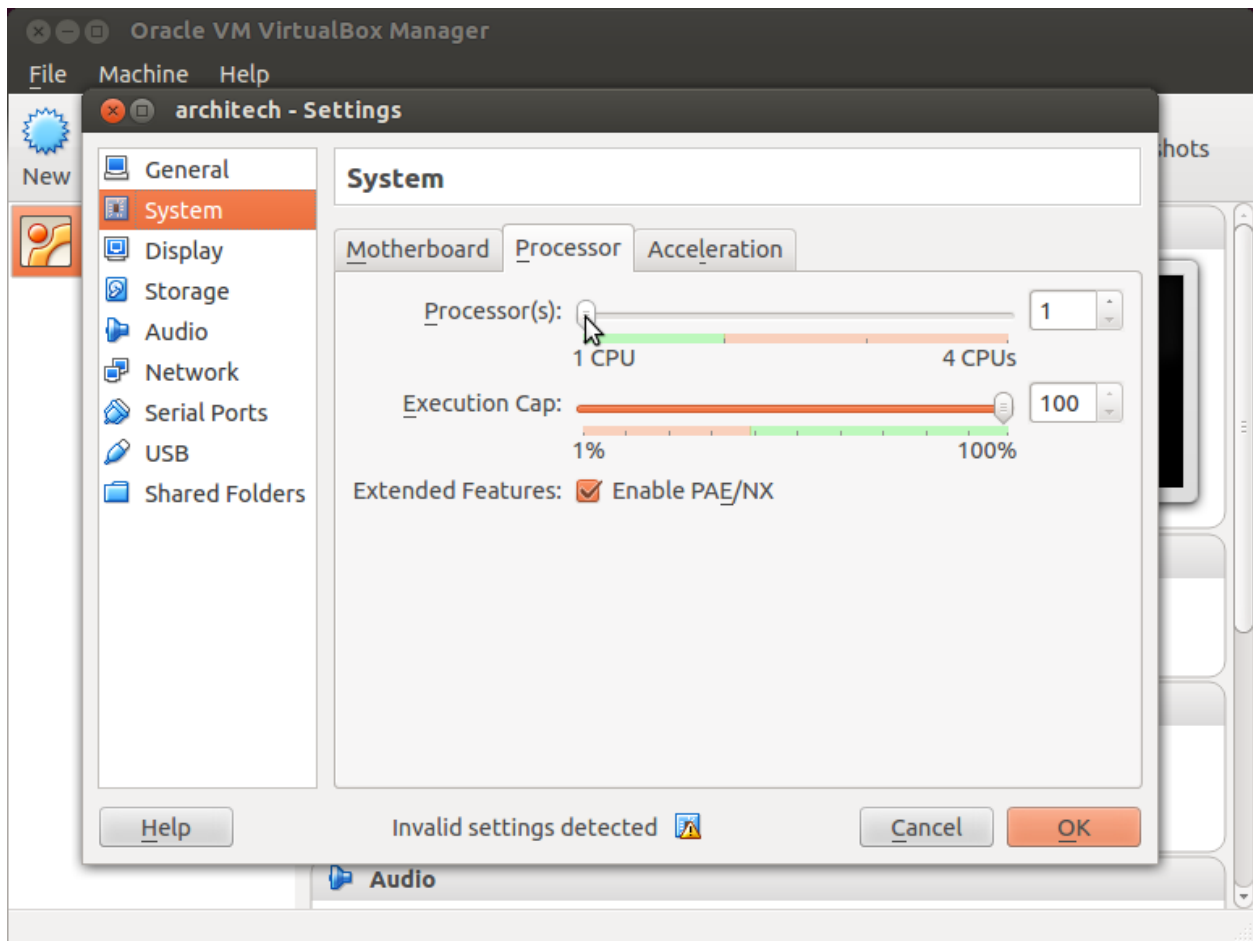
1. Select Architech's virtual machine from the list of virtual machines



2. Click on *Settings*



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

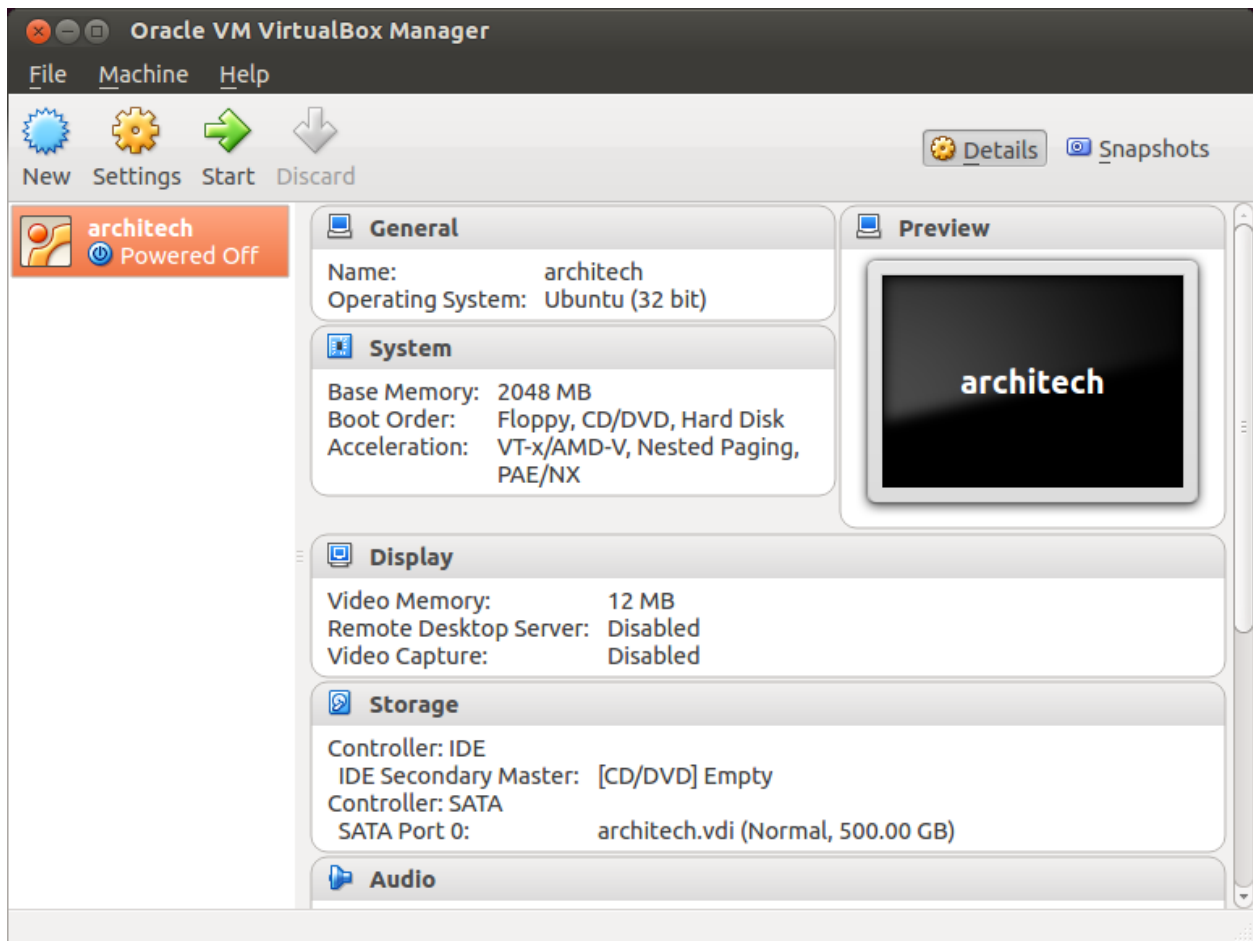


Create a shared folder

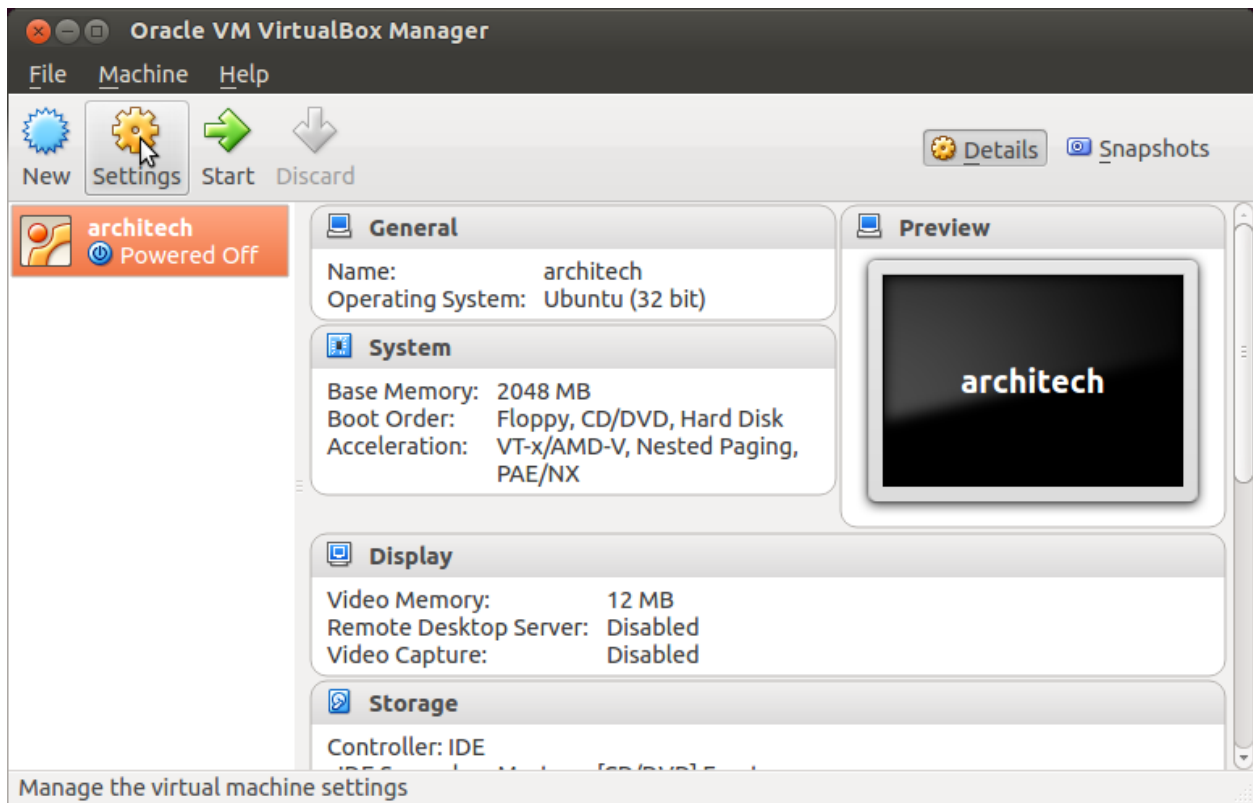
A shared folder is way for host and guest operating systems to exchange files by means of the file system. You need to choose a directory on your host operating system to share with the guest operating system.

Note: The virtual machine must be off

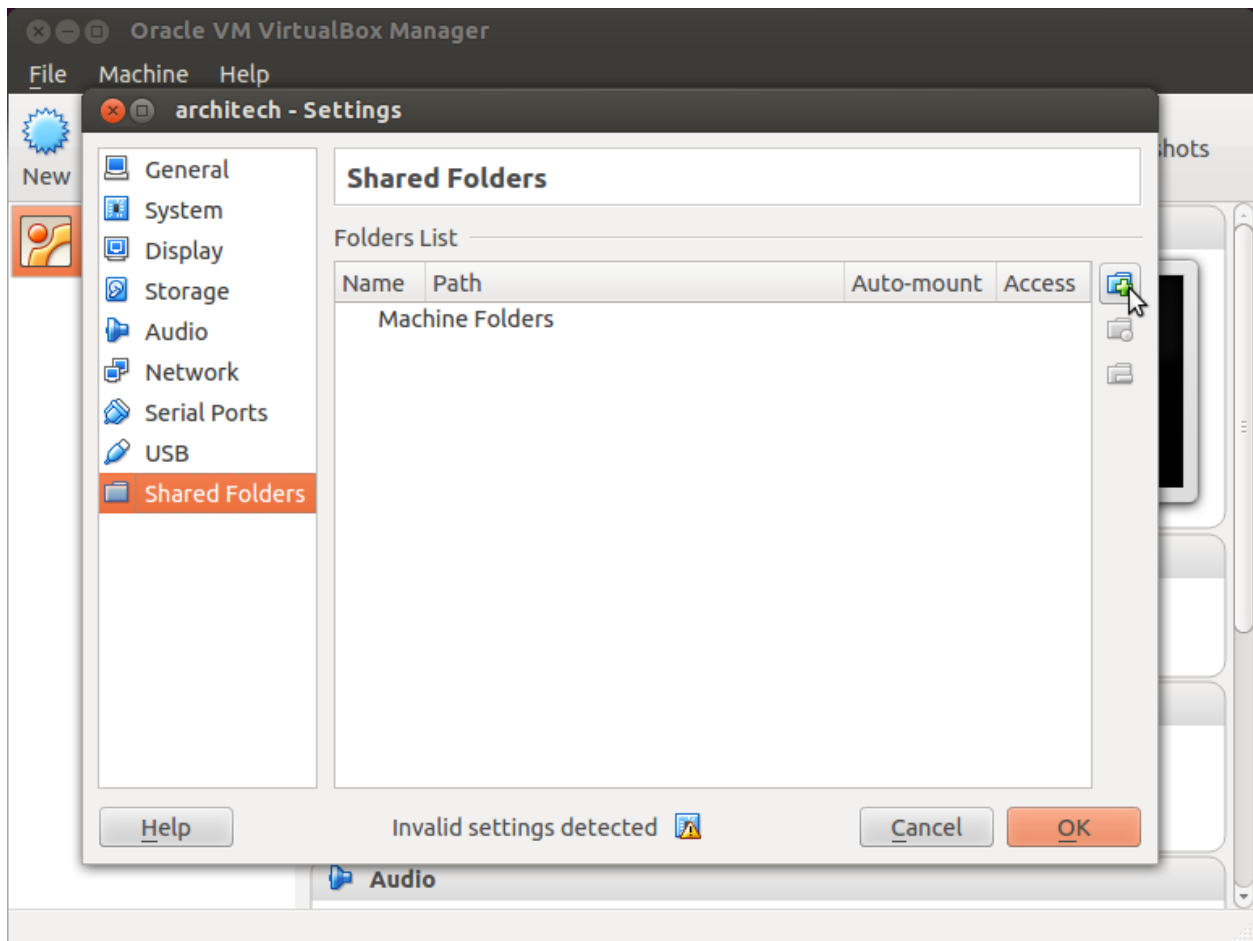
1. Select Architech's virtual machine from the list of virtual machines



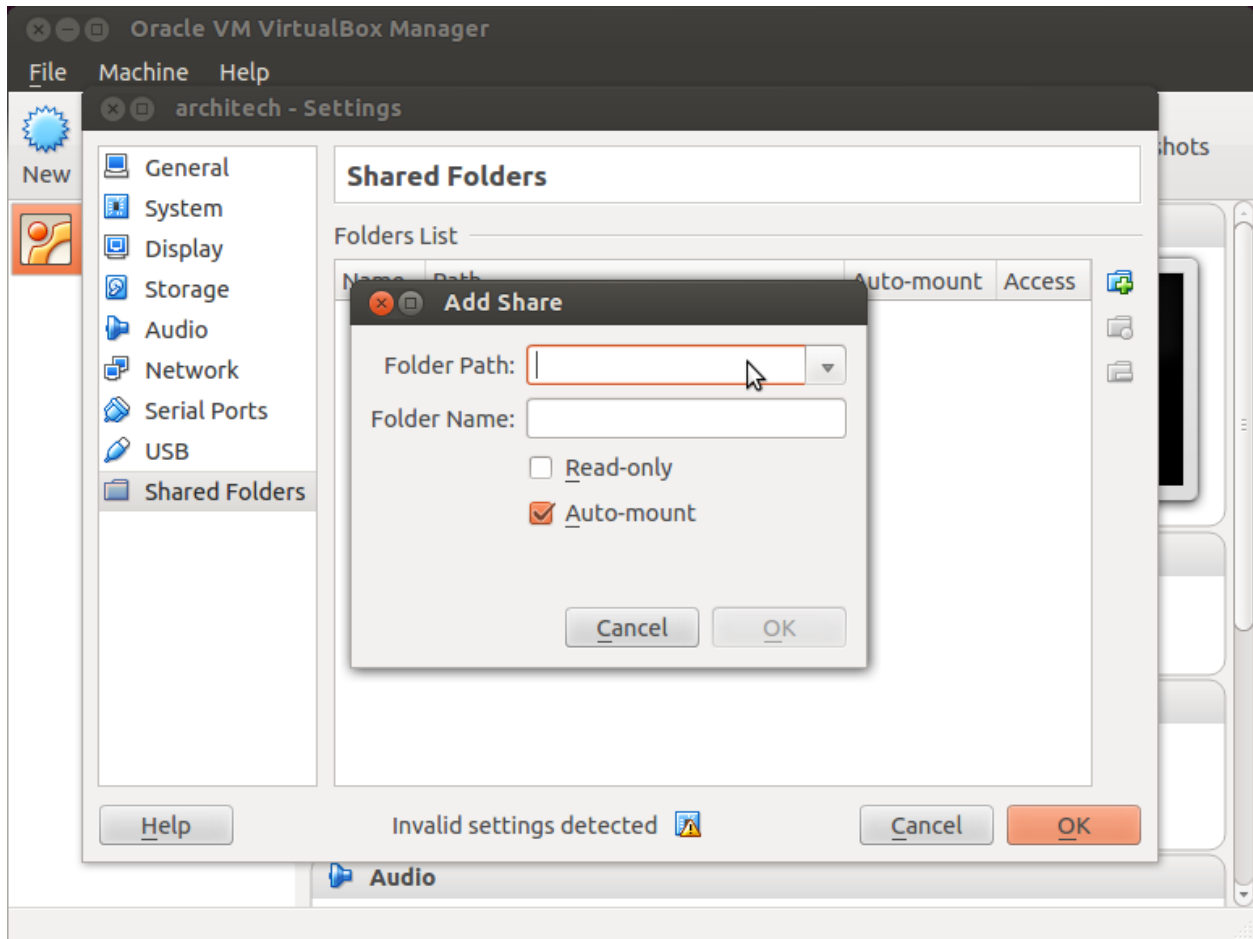
2. Click on *Settings*



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



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.

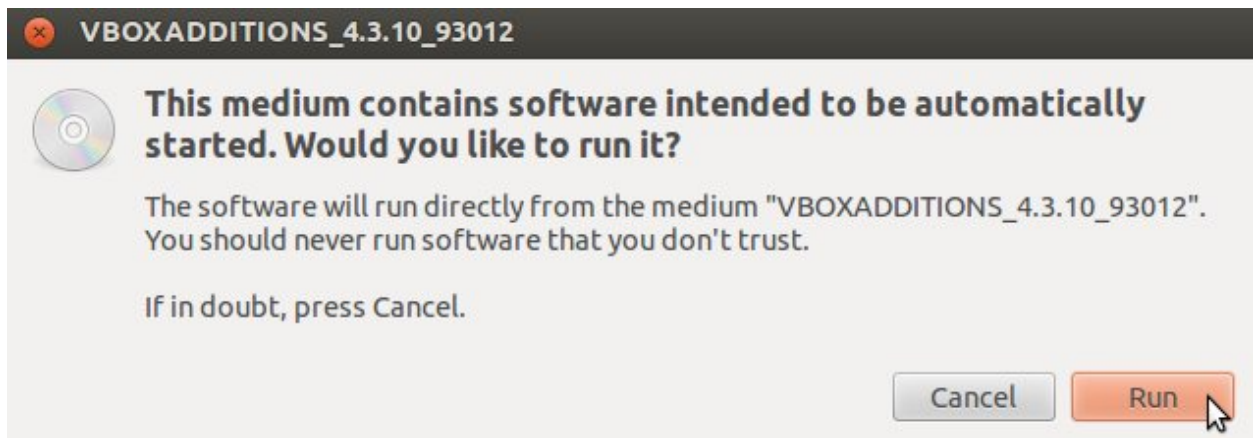
Install VBox Additions

The VBox additions 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.

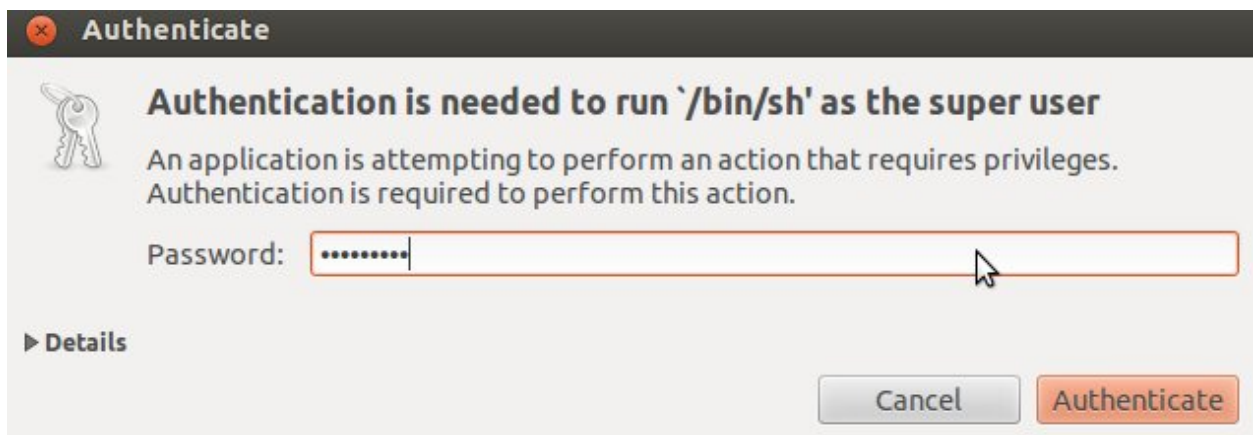
1. Starts the virtual machine



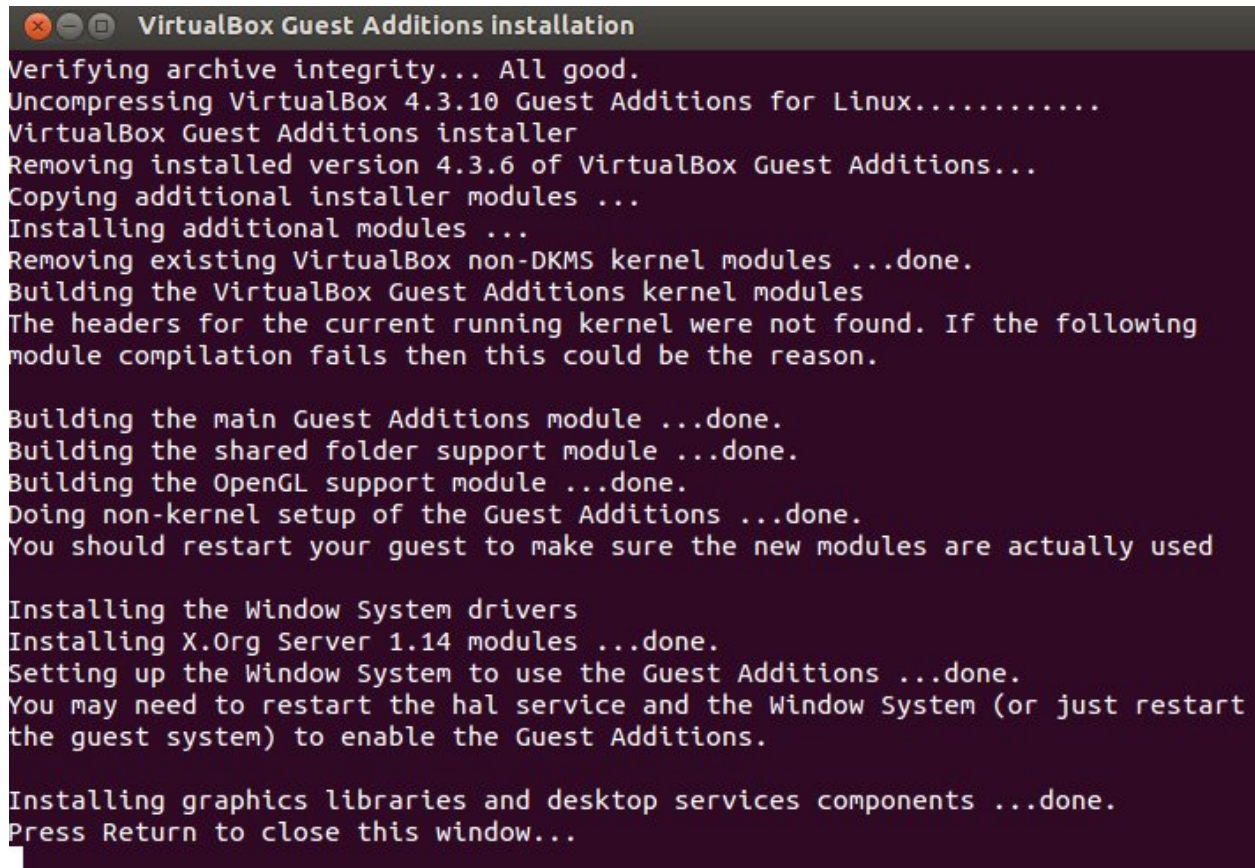
2. Click on the virtual box menu to the voice *Devices* and select *Insert Guest Additions CD Images....* A message box will appear at the start of the installation, click on *run* button



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



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.

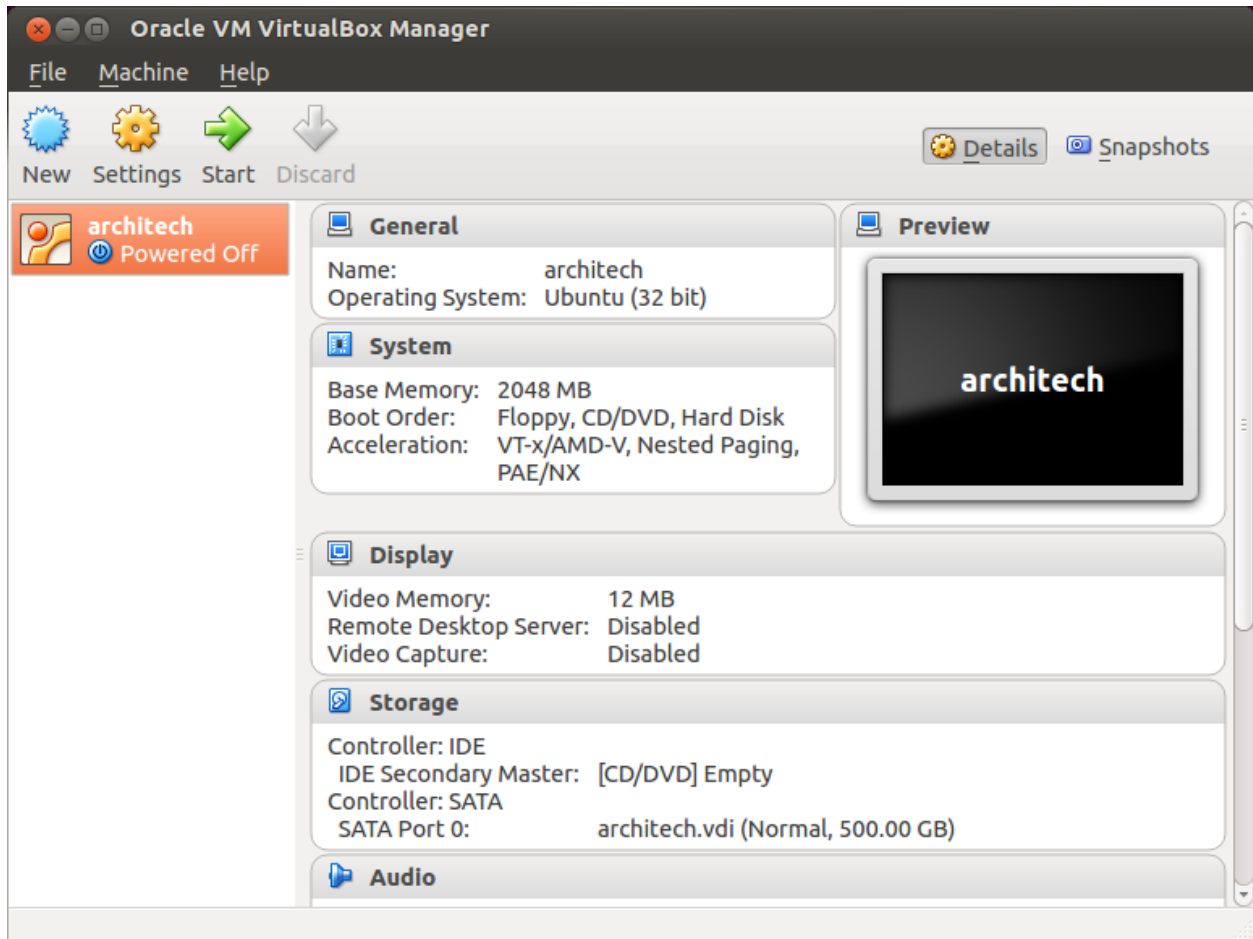
Installing graphics libraries and desktop services components ...done.
Press Return to close this window...
```

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

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



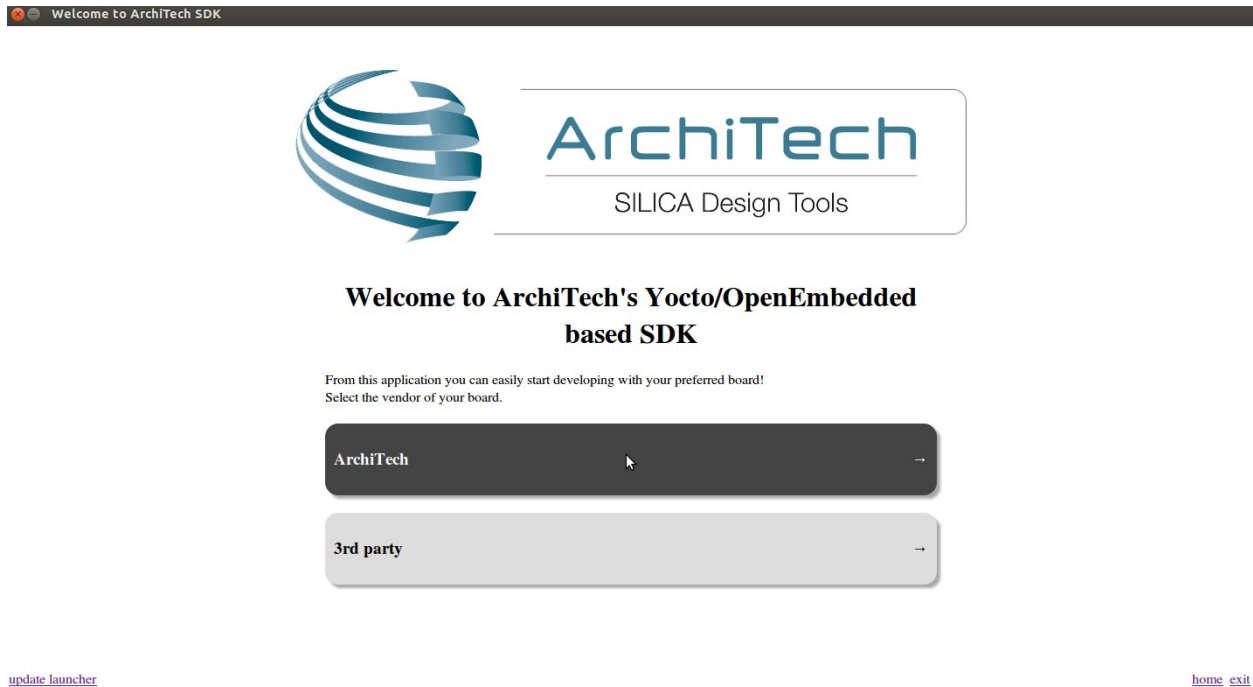
2. Click on the icon *Start* button in the toolbar and wait until the virtual machine is ready



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*.



5. Select Galileo as board you want develop on.



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*.

Now set `BB_NUMBER_THREADS` and `PARALLEL_MAKE` values:

Warning:

where *X* is the number of cores You assigned to the virtual machine multiplied for 2

9. Save the file and close gedit.

10. Build *image-full-galileo* 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**”

Important:

Eclipse needs the *sysroot* directory to compile.

The cross-toolchain looks for the required libraries in *sysroot*.

All the files in *sysroot* needs to be also in the *sdcard*.

Deploy

To deploy the root file system, you need a micro SD card (or USB device).

Warning:

The SD card must meet the following requirements:

- SD card must be formatted as FAT or FAT32.
- SD card size must be 32GB (or smaller) and SDHC format. SDXC format is not supported.

Warning:

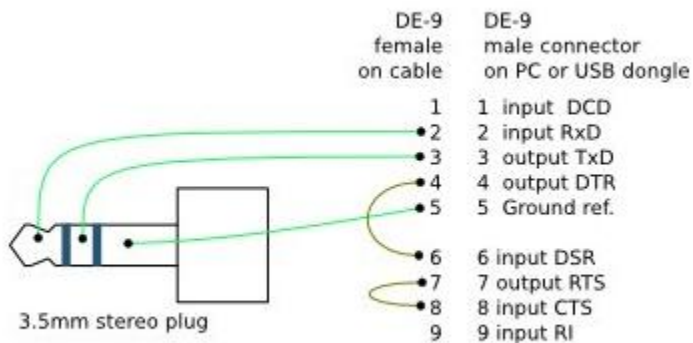
The content of the SD card will be lost forever!

1. Copy the following kernel and root file system files to your SD card
2. Unmount the micro SD card from your computer
3. Interact with the embedded Linux using a cable connected to the minijack UART port.

Important:

For a 3 pin audio jack, the connections are as follows:

- Sleeve = Ground
- Ring = RXD (Input to Clanton)
- Tip = TXD (Output from Clanton)



4. Insert the SD card, then power on the board.
5. Connect the audio jack to UART connector

Important: The first time you boot the board may take several minutes. This is expected behavior due to the SSH component creating cryptographic keys on the first boot.

Warning: To boot from SD/USB, the grub instance embedded in the SPI flash is hardcoded to search for a boot/grub/grub.conf file in partition 1 on the SD/USB card. This is compatible with the factory formatting of most SD/USB devices. By default, the UEFI firmware does not try to boot from SD or USB, it is handled by grub

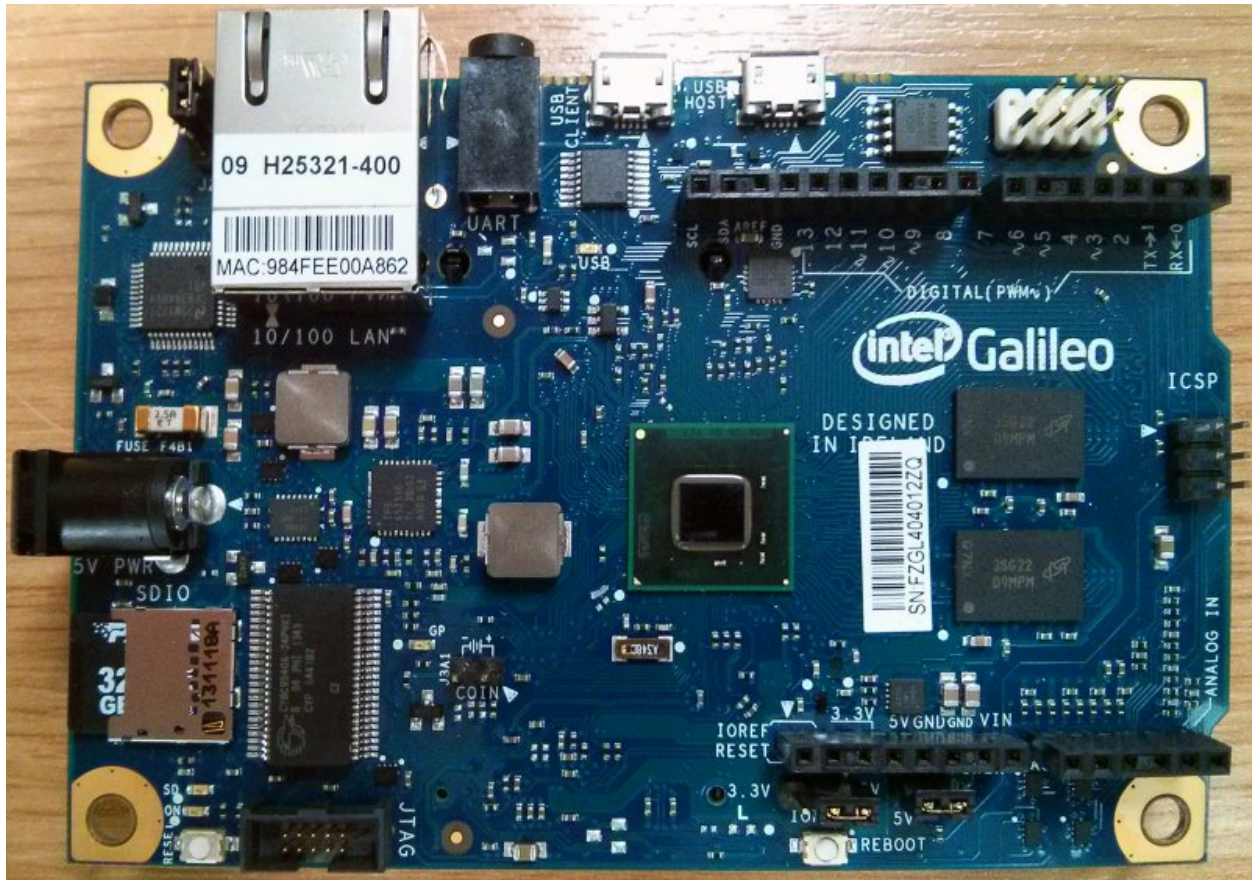
Warning: If you use an SD or USB device that has been reformatted after manufacturing, you might experience problems booting from it. First, try to boot with a different memory device and see if the problem goes away.

Boot

Take the power socket adapter compatible with your country, plug it in the power adapter. When in position, you should hear a slight *click*. Power on the board connecting the external power adapter to Galileo jack **5V PWR**.

Now it's time to start the serial console.

On Galileo there is the dedicated serial console connector **UART** (audio mini-jack)



which you can connect, by the serial to mini-jack UART port 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 `ttyUSBX` or `ttySX` device 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/ttyS0`.

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/ttyS0`, just replace `/dev/ttyS0` 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

galileo 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.

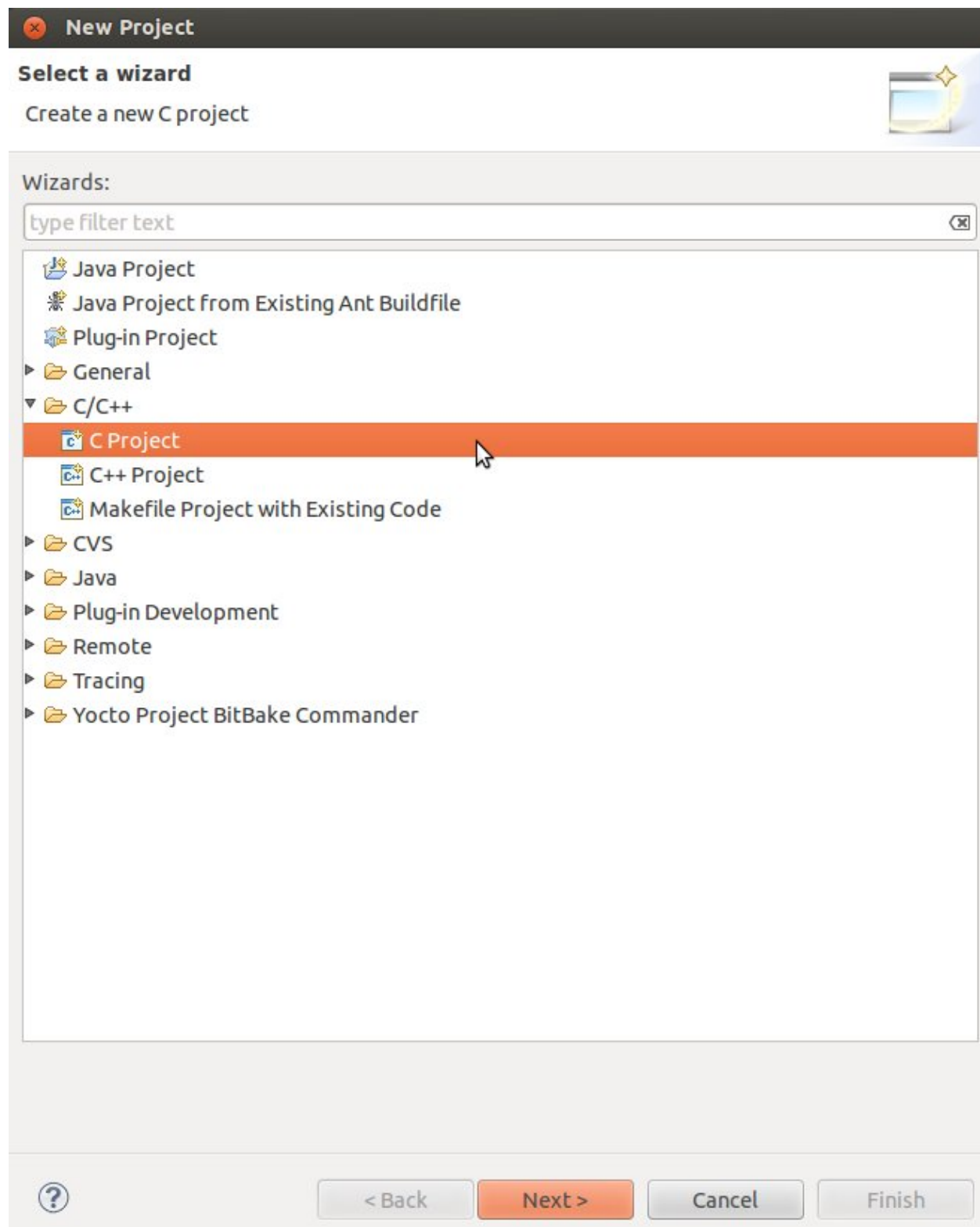
Code

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

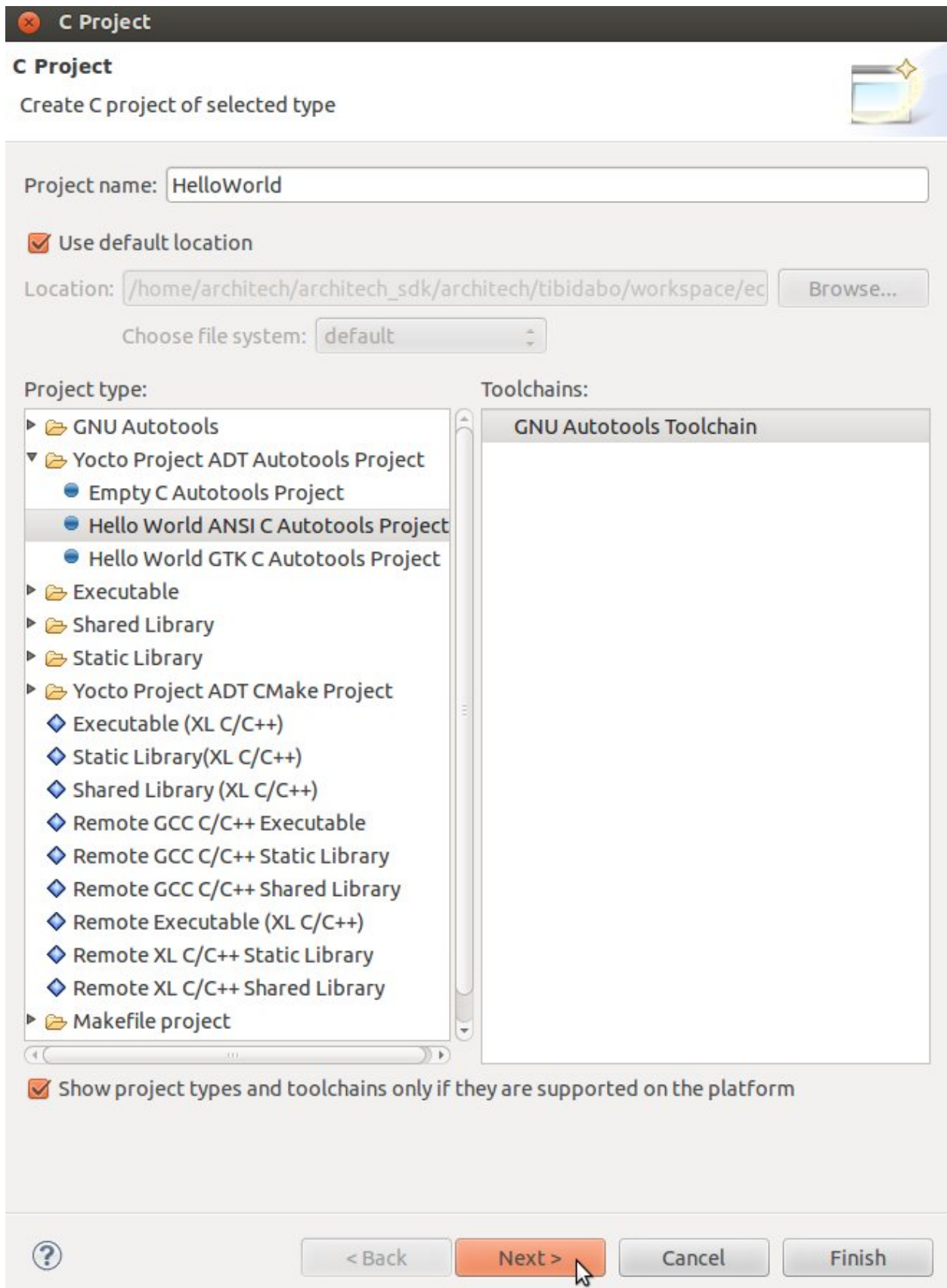
1. Return to the **Splashscreen**, which we left on Galileo board screen, and click on *Develop with Eclipse*.



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.



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:

Hello world greeting:

Source:

License:

? < Back Next > Cancel Finish

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

type filter text

- Resource
- Autotools
- Builders
- C/C++ Build
- C/C++ General
- Project References
- Run/Debug Settings
- Yocto Project Settings**

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 10/100 LAN) to your PC. Configure your workstation ip address as 192.168.0.100. Make sure the board can be seen by your host machine:

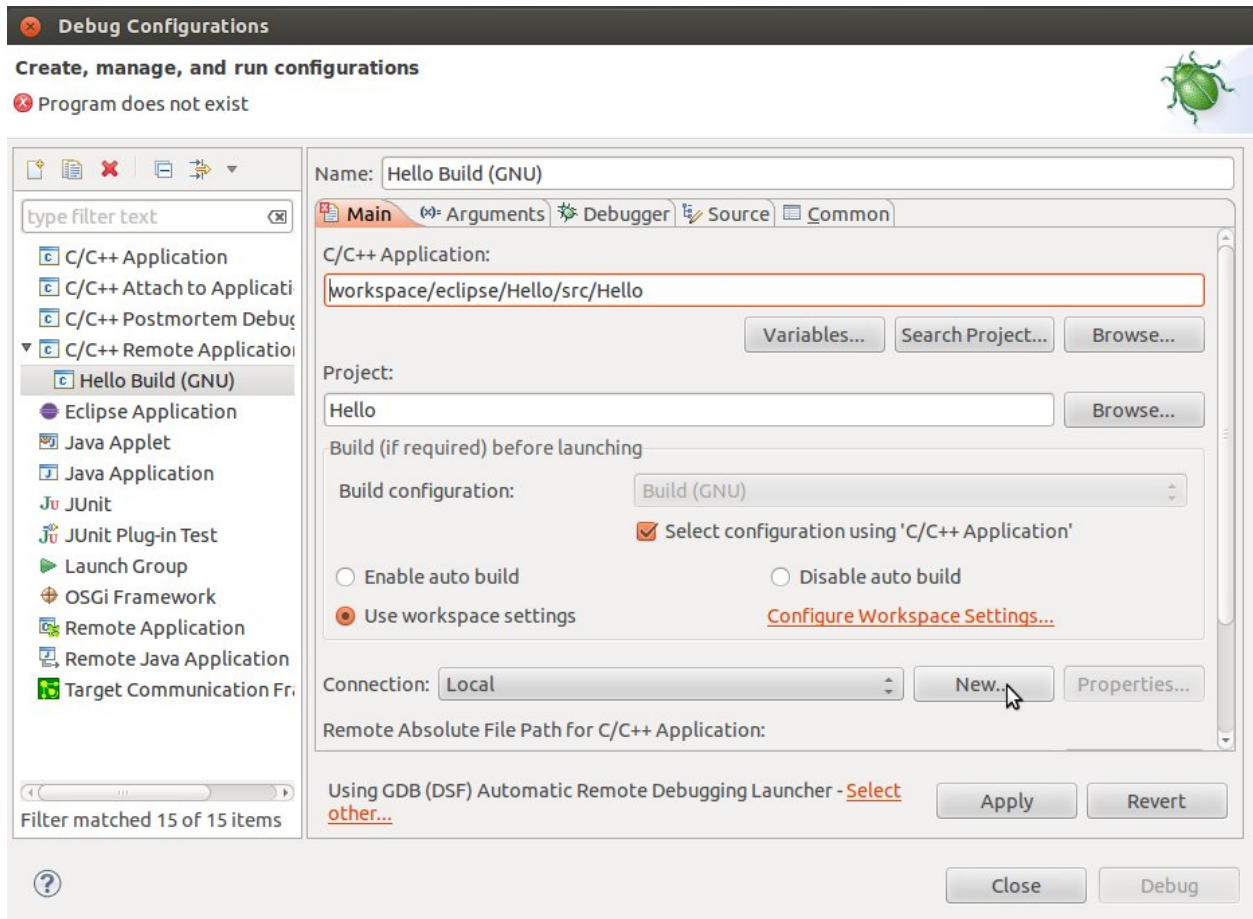
If the output is similar to this one:

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

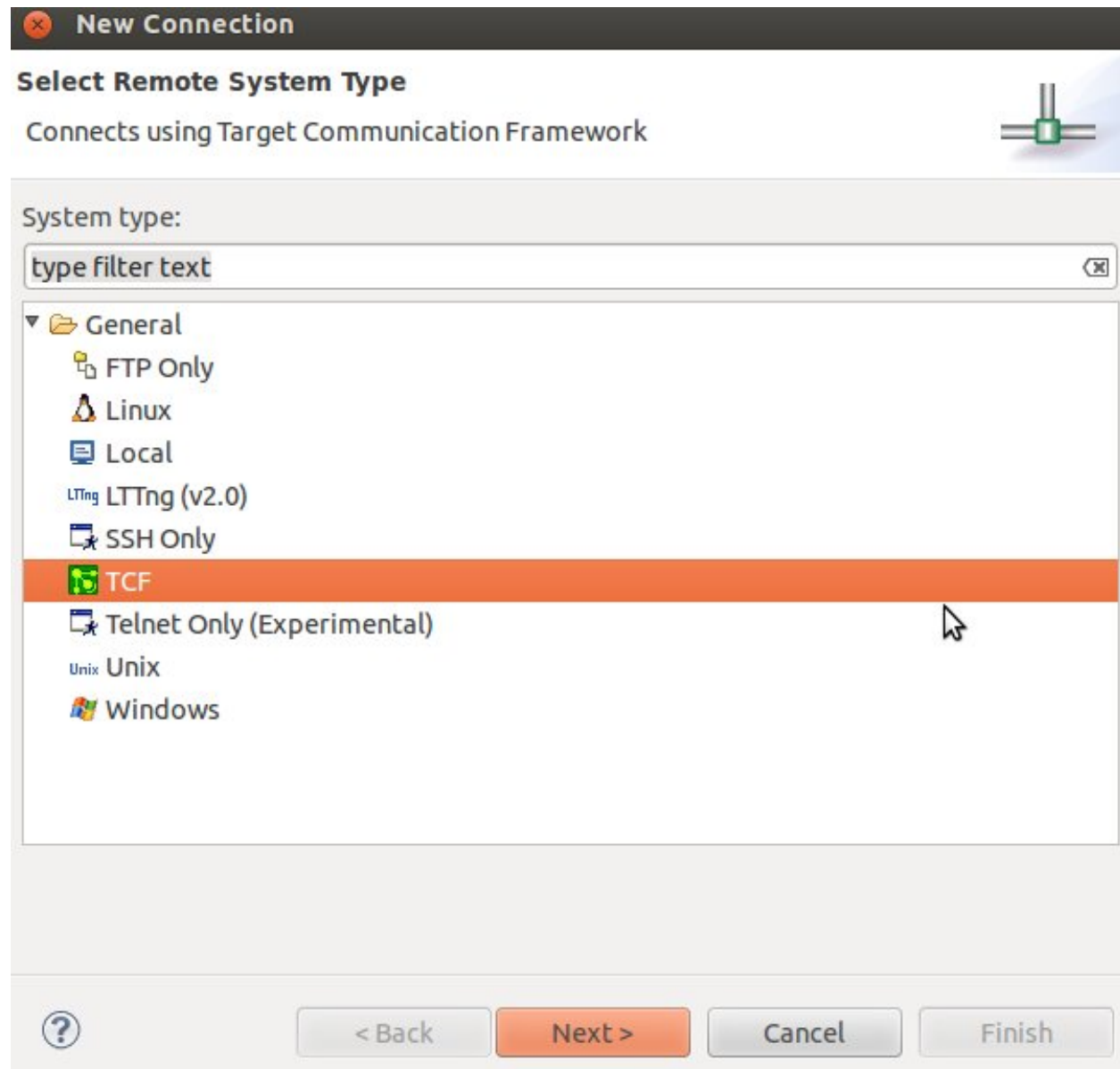
On the Host machine, follow these steps to let **Eclipse** deploy and debug your application:

- 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.



- 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

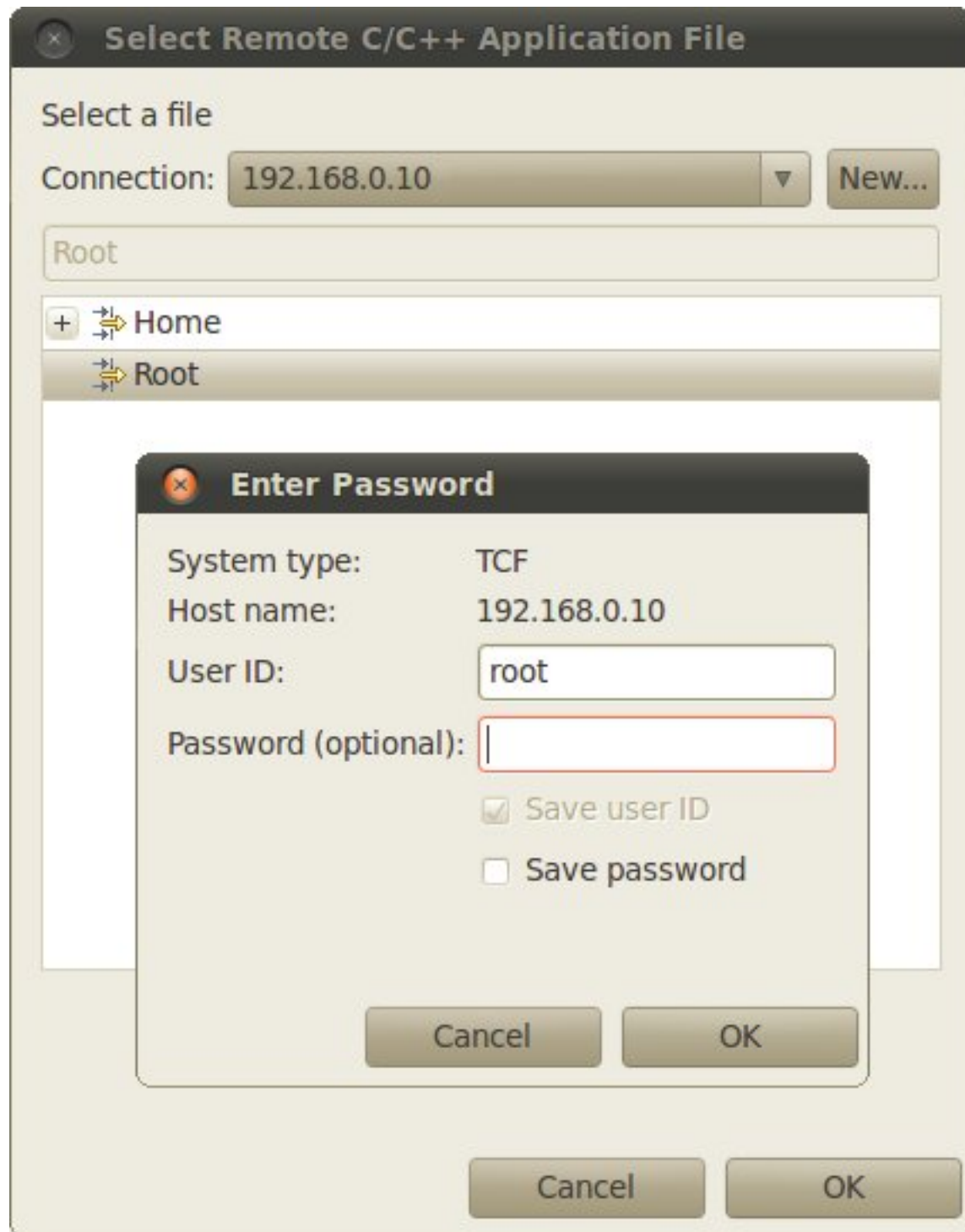
Description: Commentary description of the connecti

☒ Verify host name

[Configure proxy settings](#)

? < Back Next > Cancel Finish

- 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.

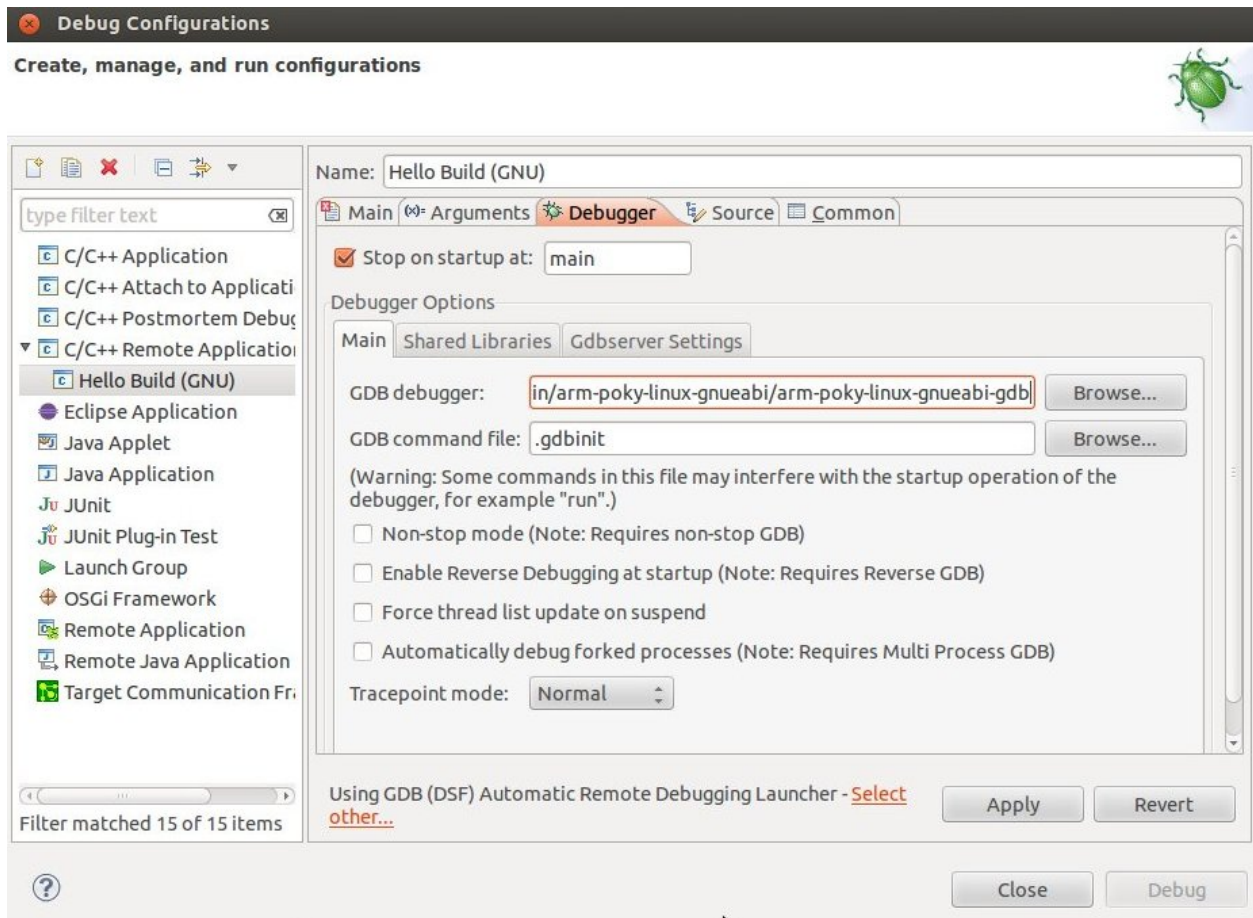


- Enter also in the path the name of the application you want to debug. (e.g. HelloWorld)

Connection: 192.168.0.10

Remote Absolute File Path for C/C++ Application:
/home/root/HelloWorld

- Select *Debugger* tab



- 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 Galileo 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, Galileo 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>

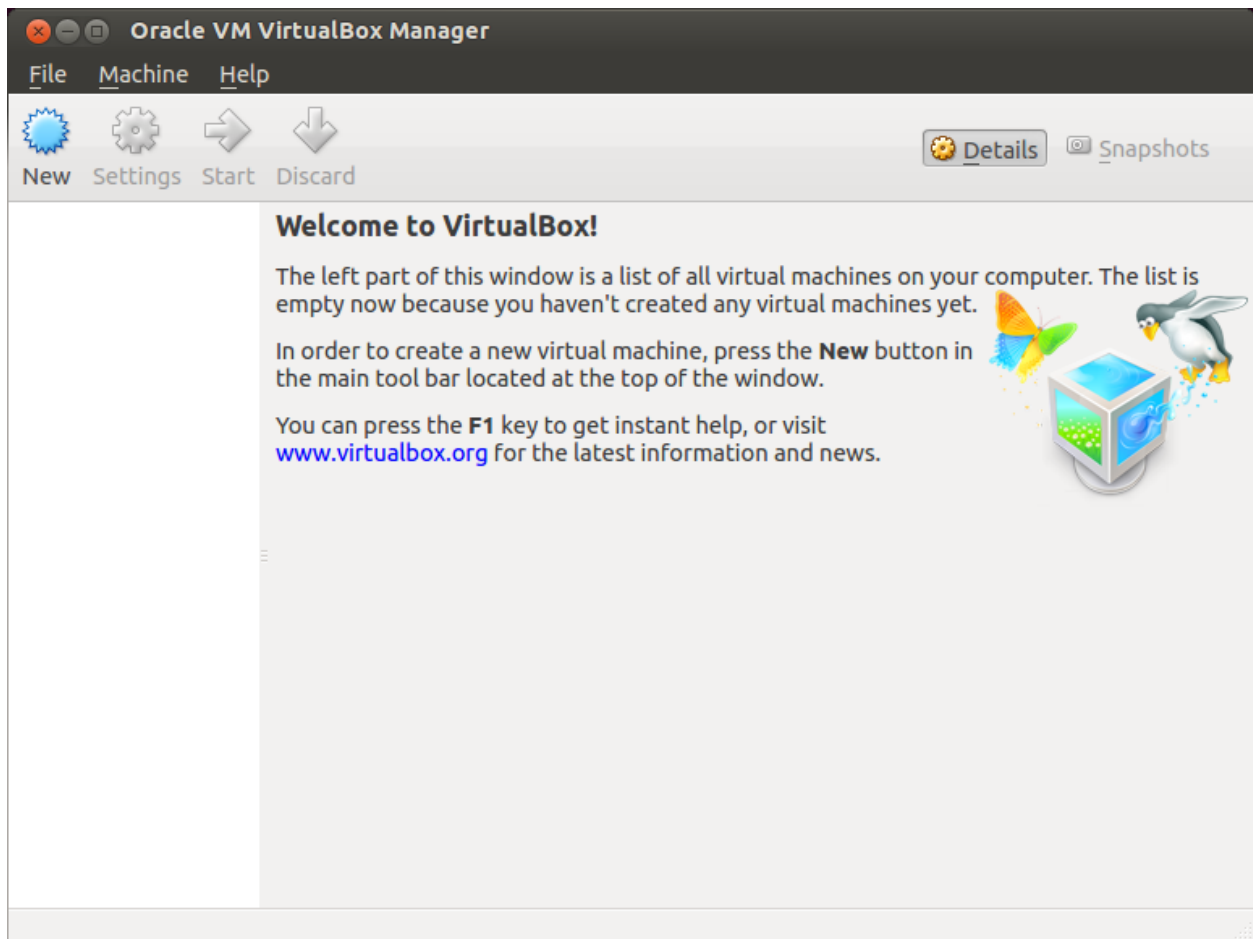
Download the version that suits your host operating system. You need to download and install the **Extension Pack** as well.

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

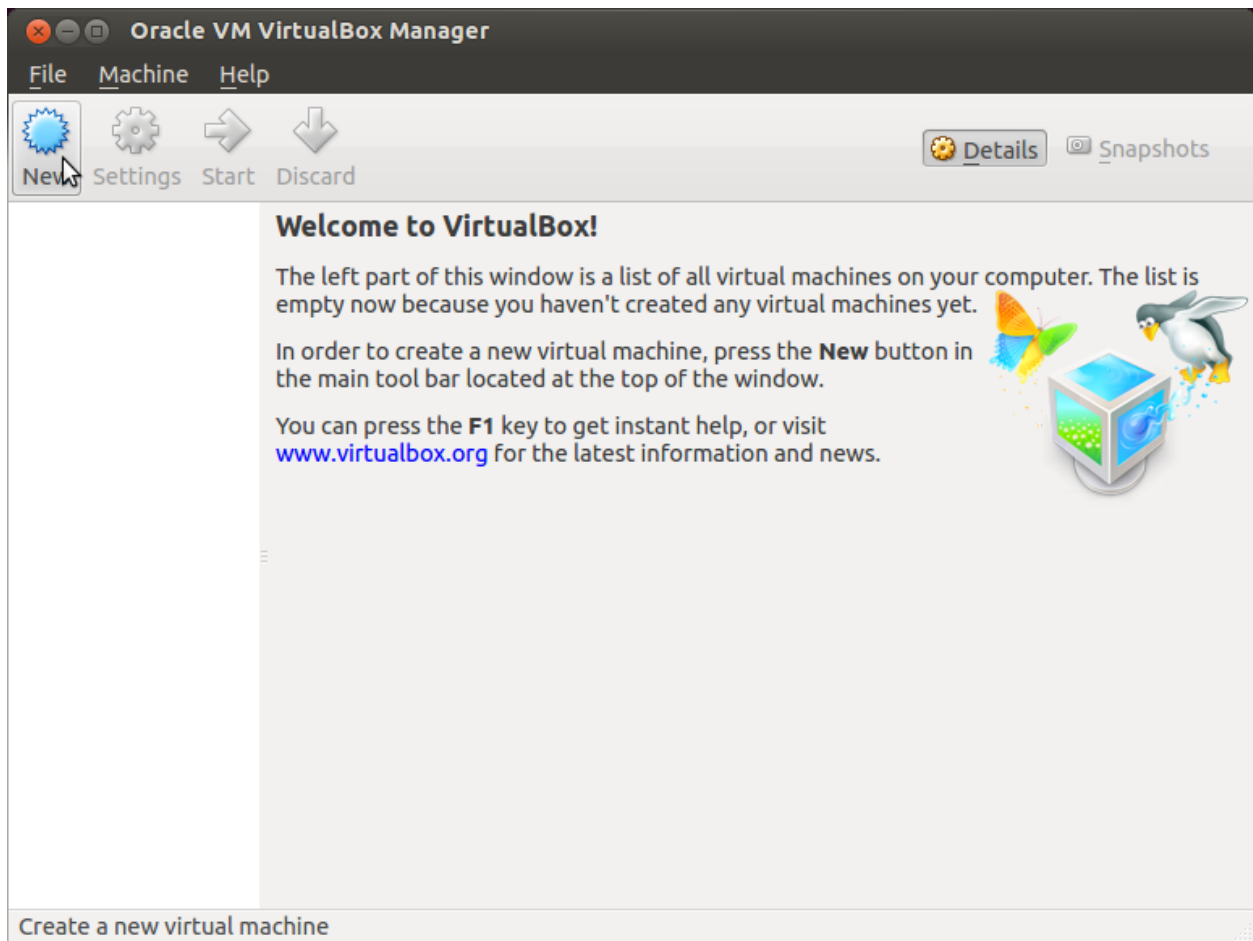
Install the software with all the default options.

Create a new Virtual Machine

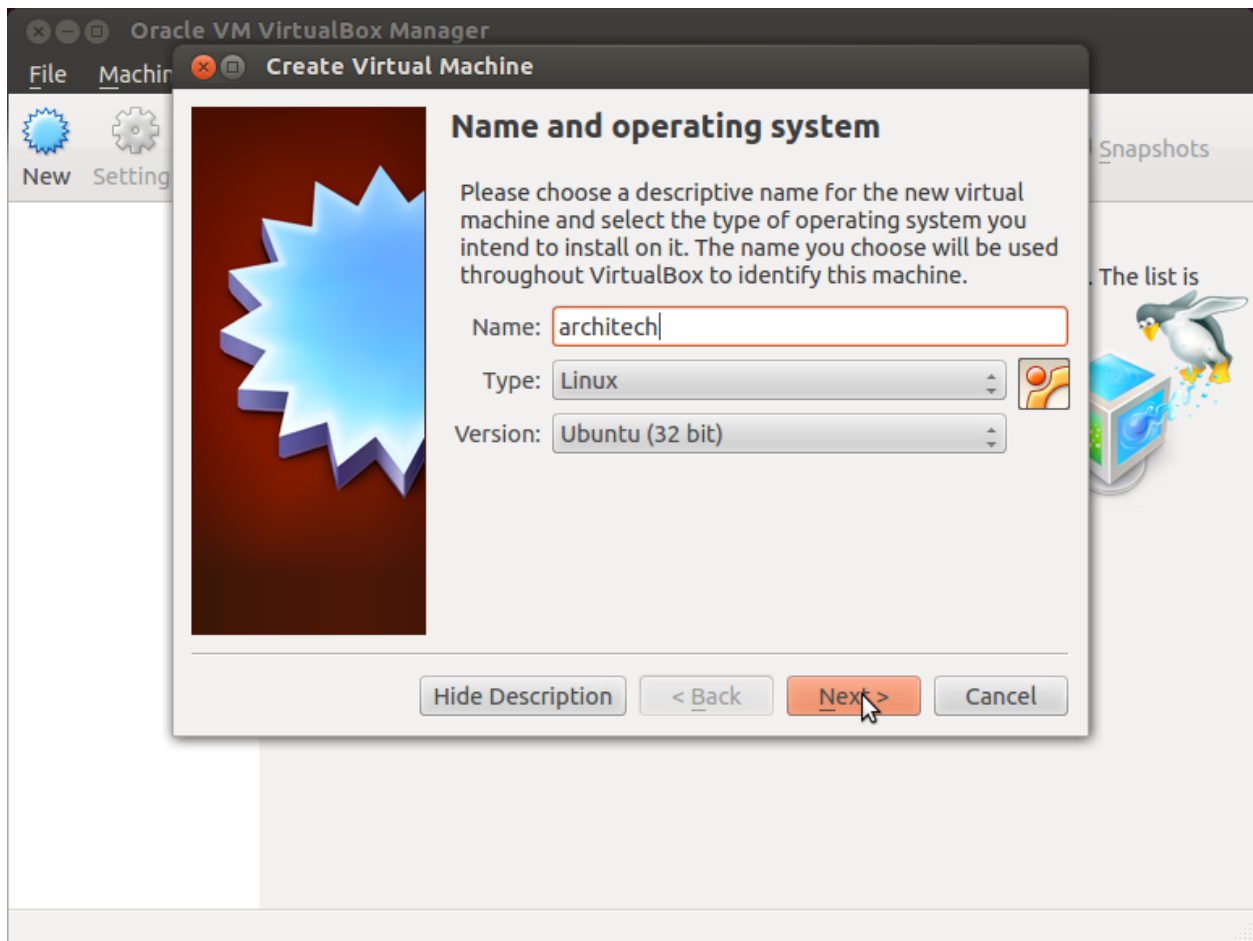
1. Run VirtualBox



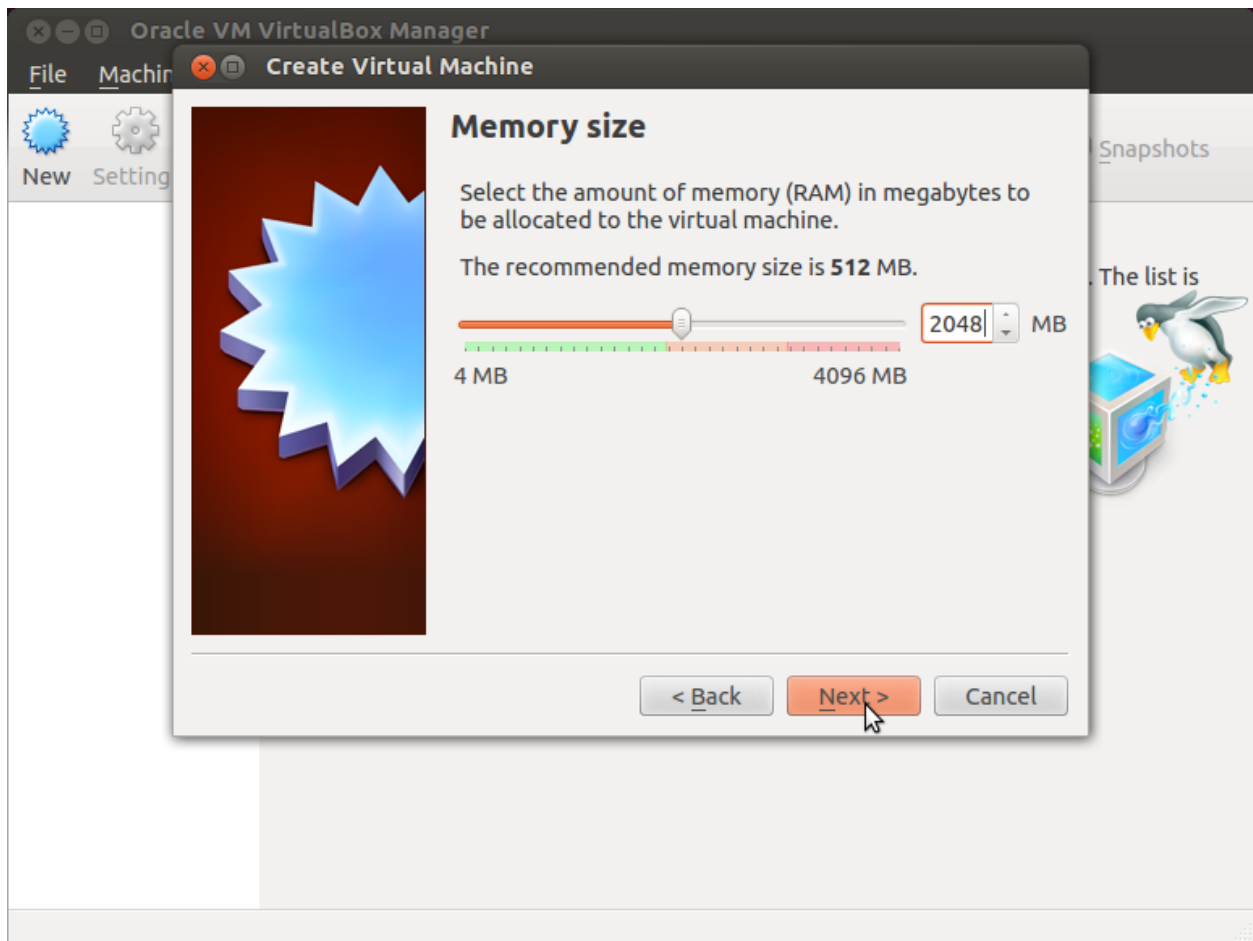
2. Click on *New* button



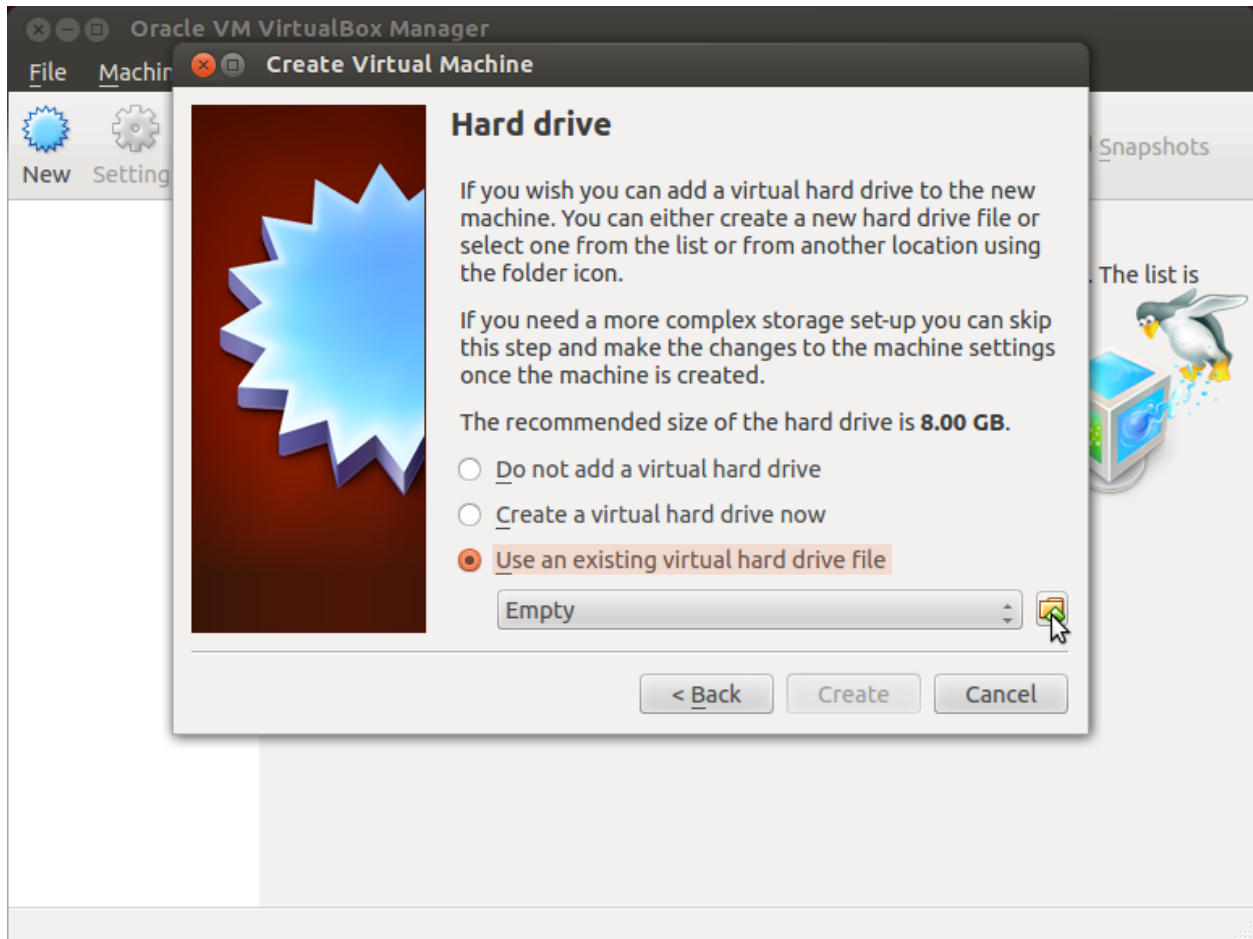
3. Select the name of the virtual machine and the operating system type



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



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

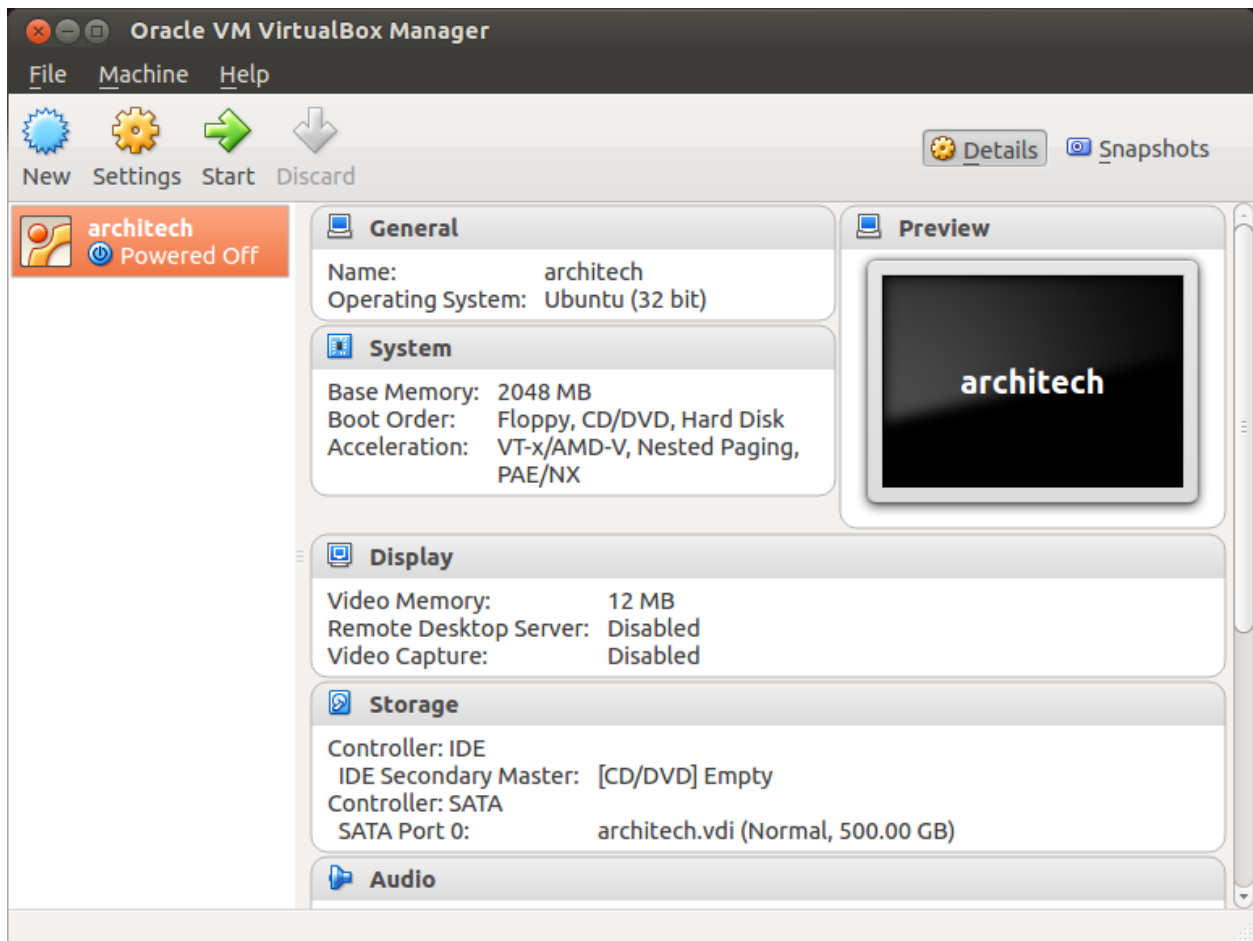


Setup the network

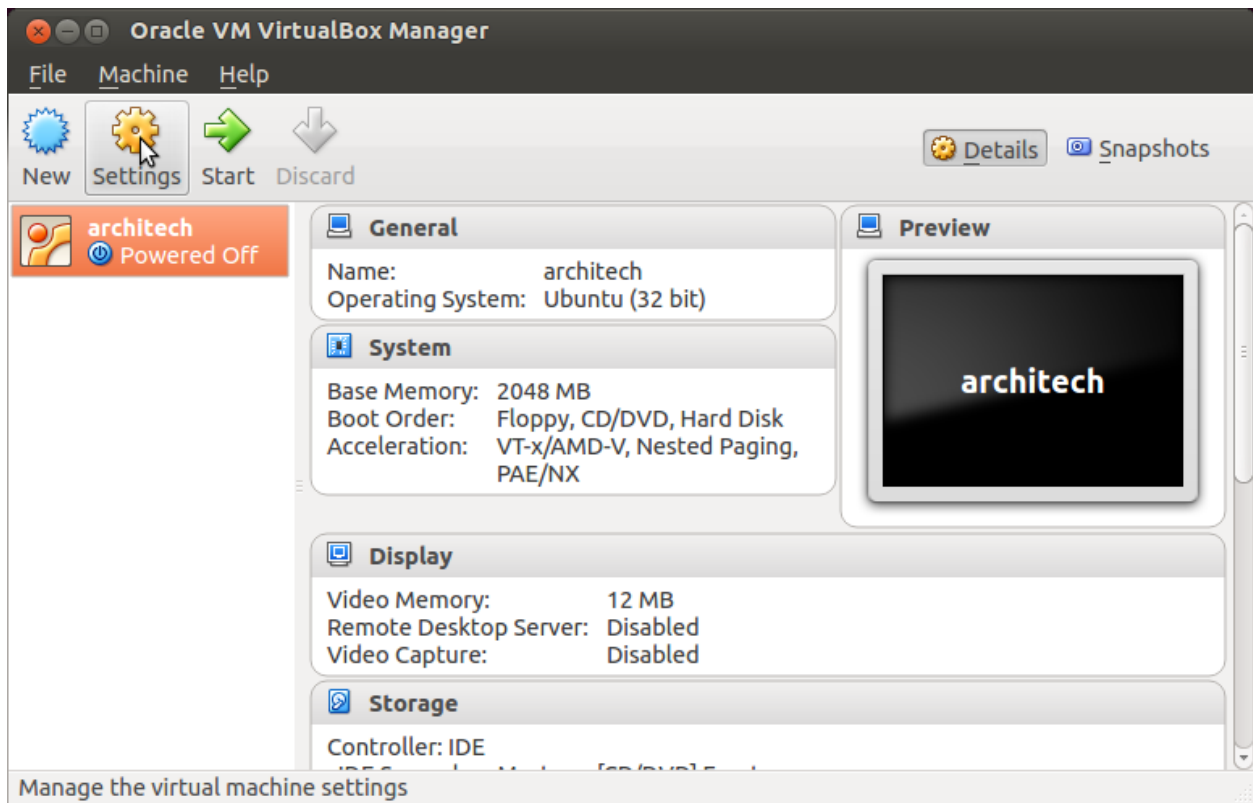
We need to setup a port forwarding rule to let you (later) use the virtual machine as a local repository of packages.

Note: The virtual machine must be off

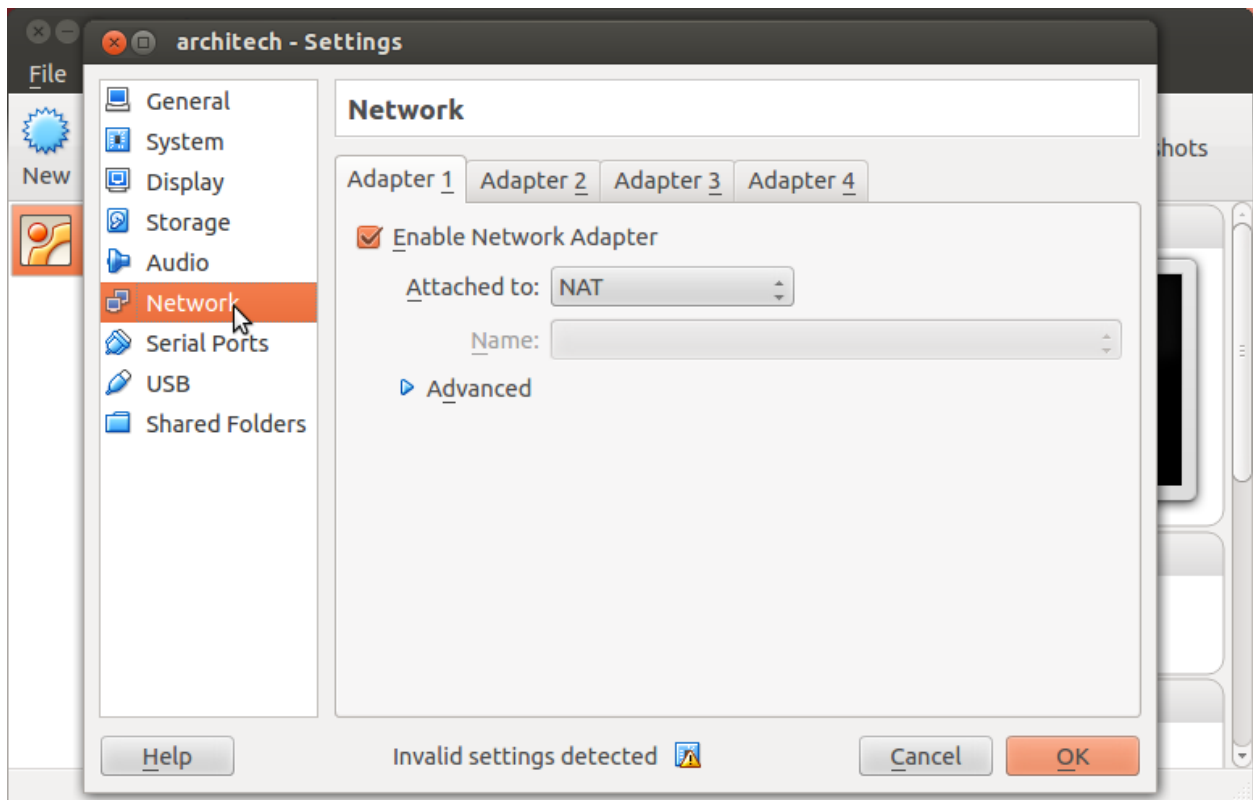
1. Select Architech's virtual machine from the list of virtual machines



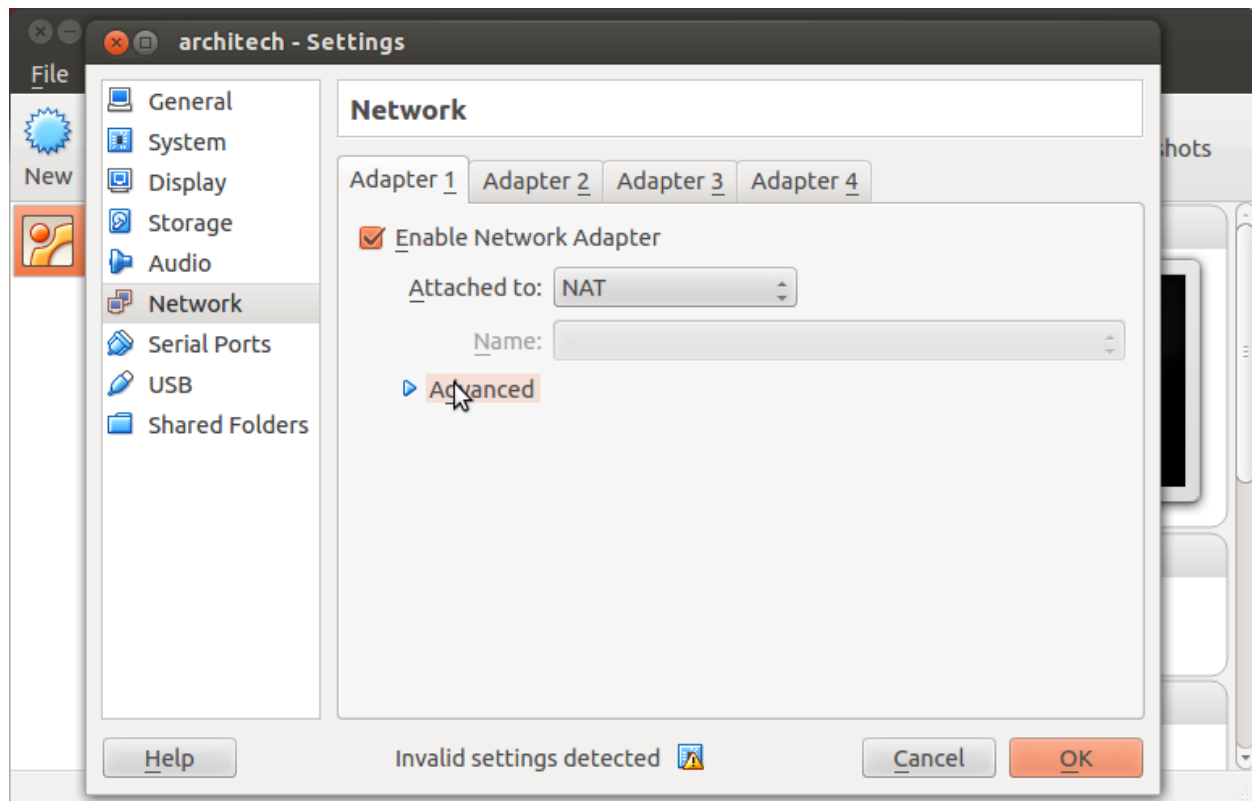
2. Click on *Settings*



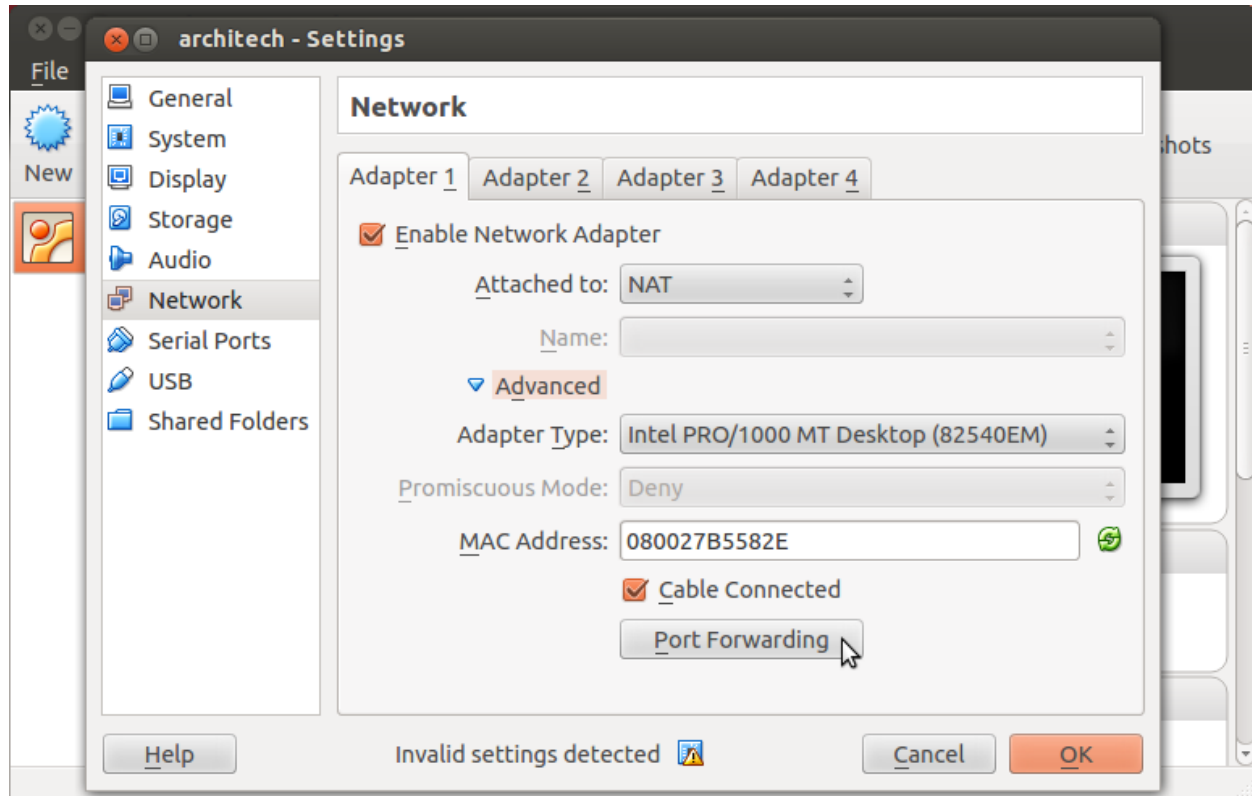
3. Select *Network*



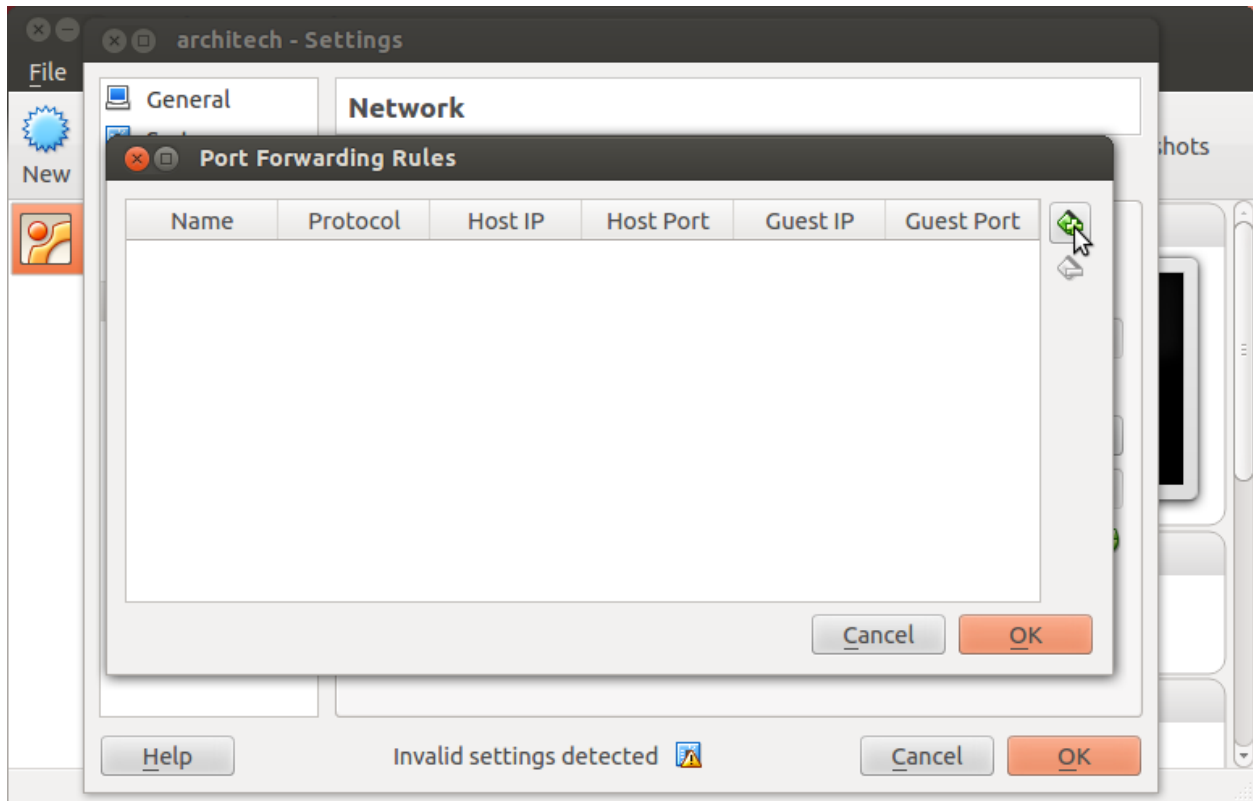
4. Expand *Advanced* of Adapter 1



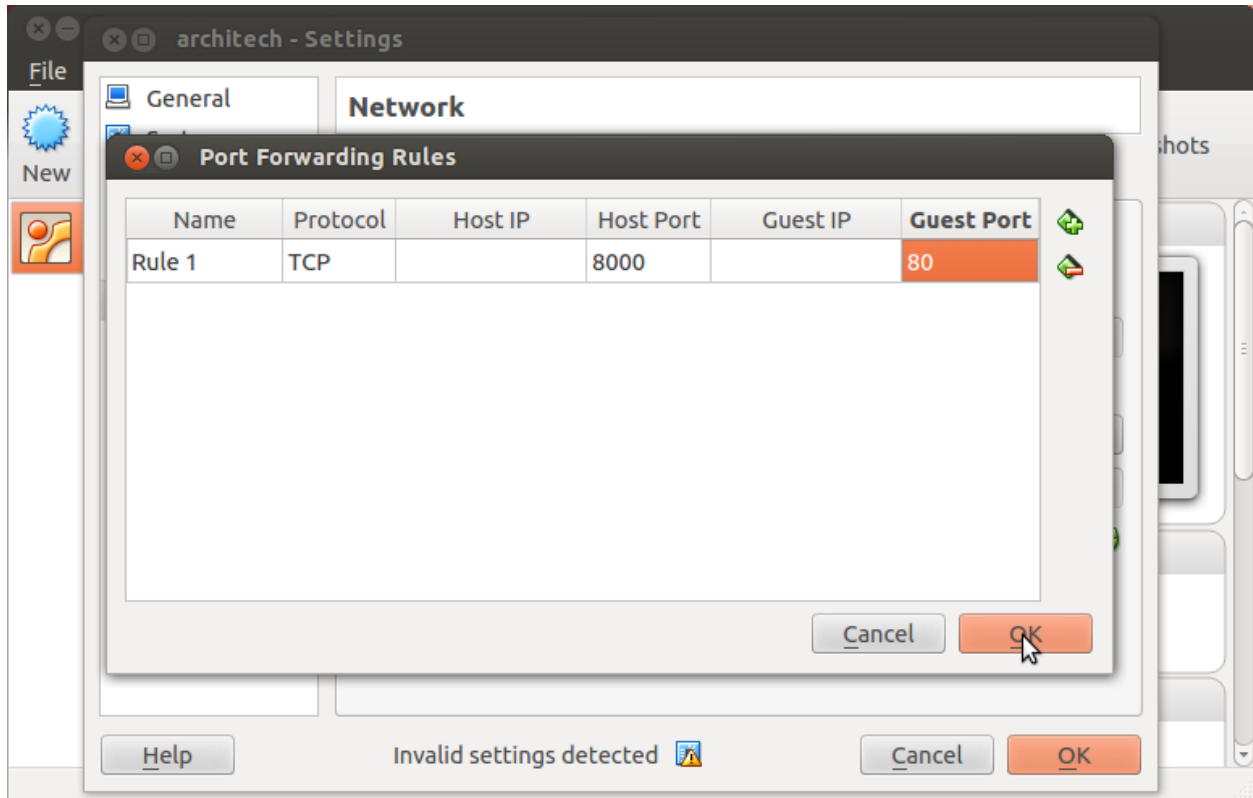
5. Click on *Port Forwarding*



6. Add a new *rule*



7. Configure the *rule*



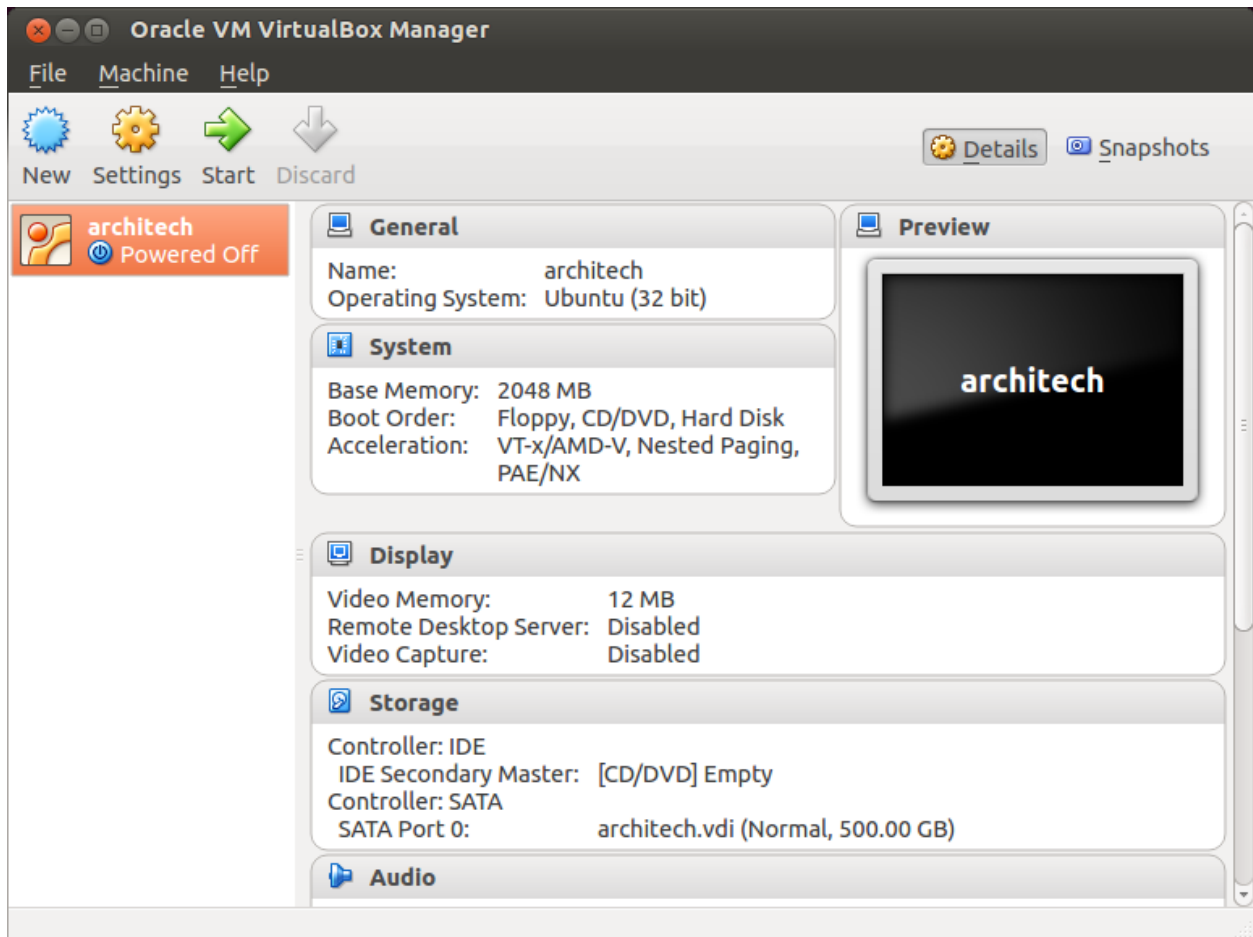
8. Click on *Ok*

Customize the number of processors

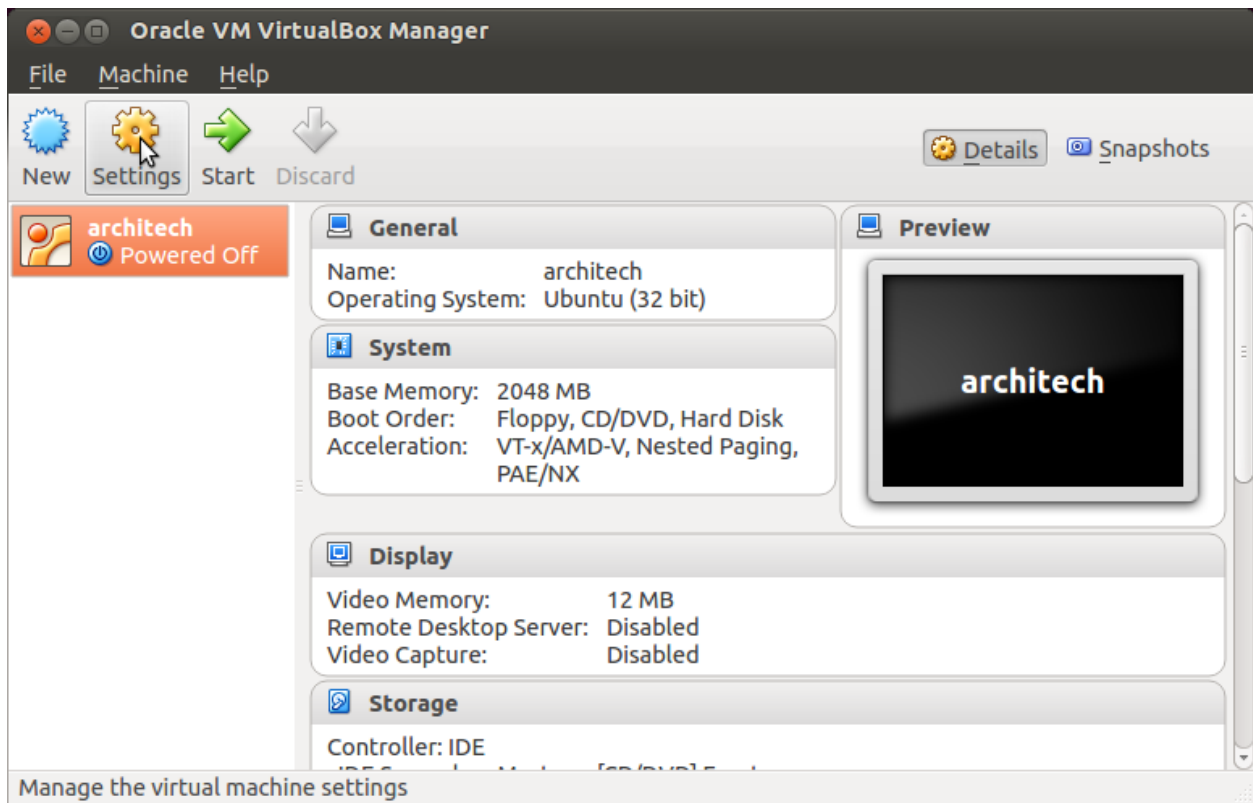
Building an entire system from the ground up is a business that can take up to several hours. To improve the performances of the overall build process, you can, if your computer has enough resources, assign more than one processor to the virtual machine.

Note: The virtual machine must be off

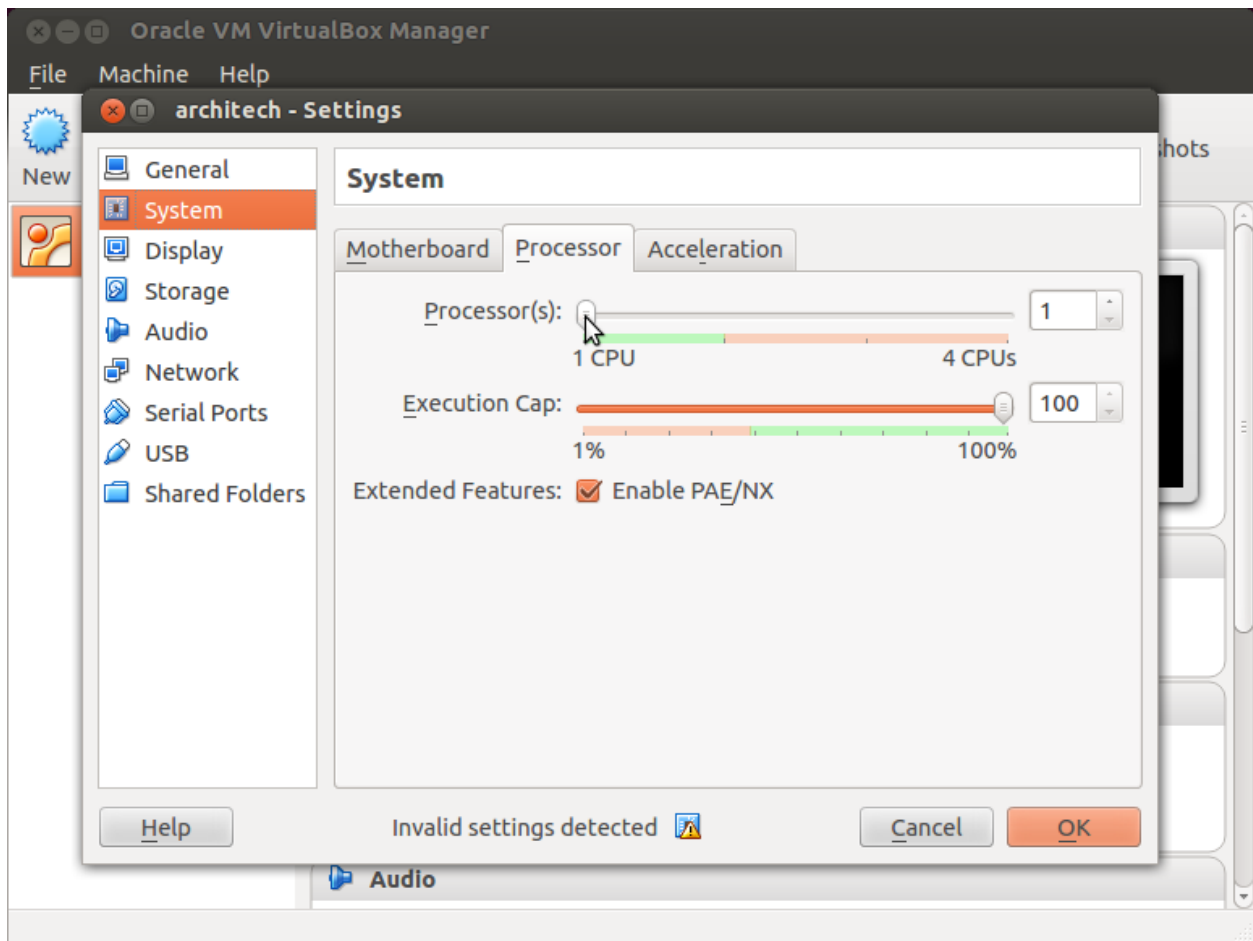
1. Select Architech's virtual machine from the list of virtual machines



2. Click on *Settings*



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

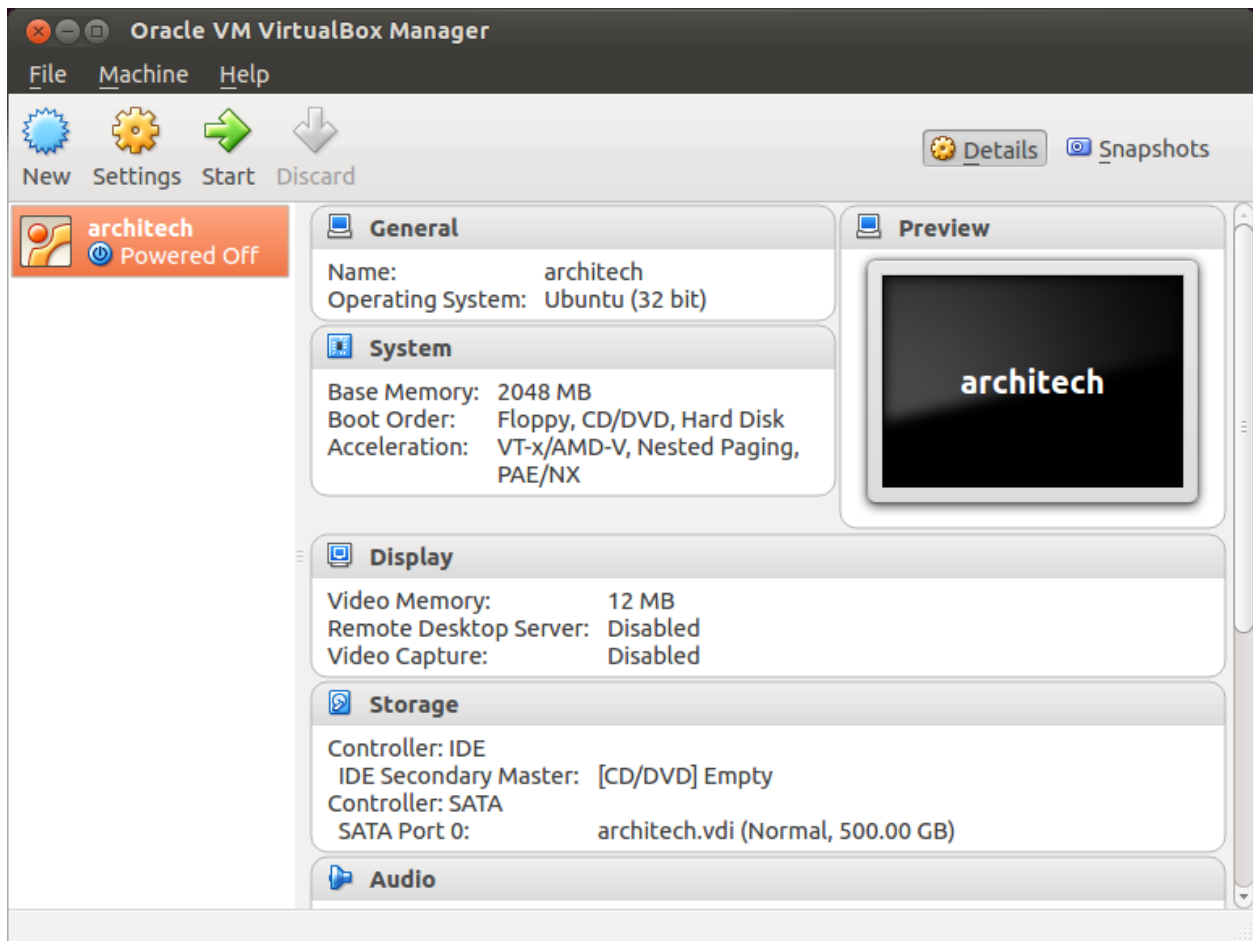


Create a shared folder

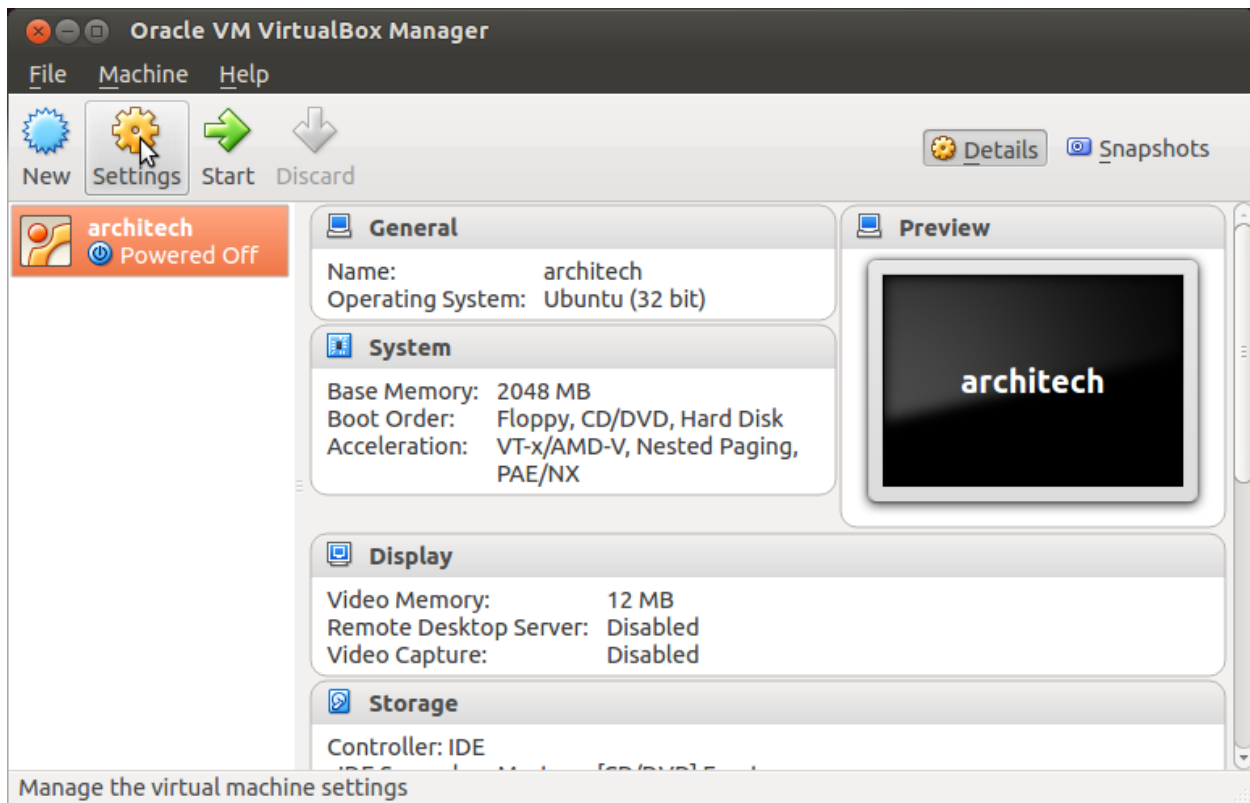
A shared folder is way for host and guest operating systems to exchange files by means of the file system. You need to choose a directory on your host operating system to share with the guest operating system.

Note: The virtual machine must be off

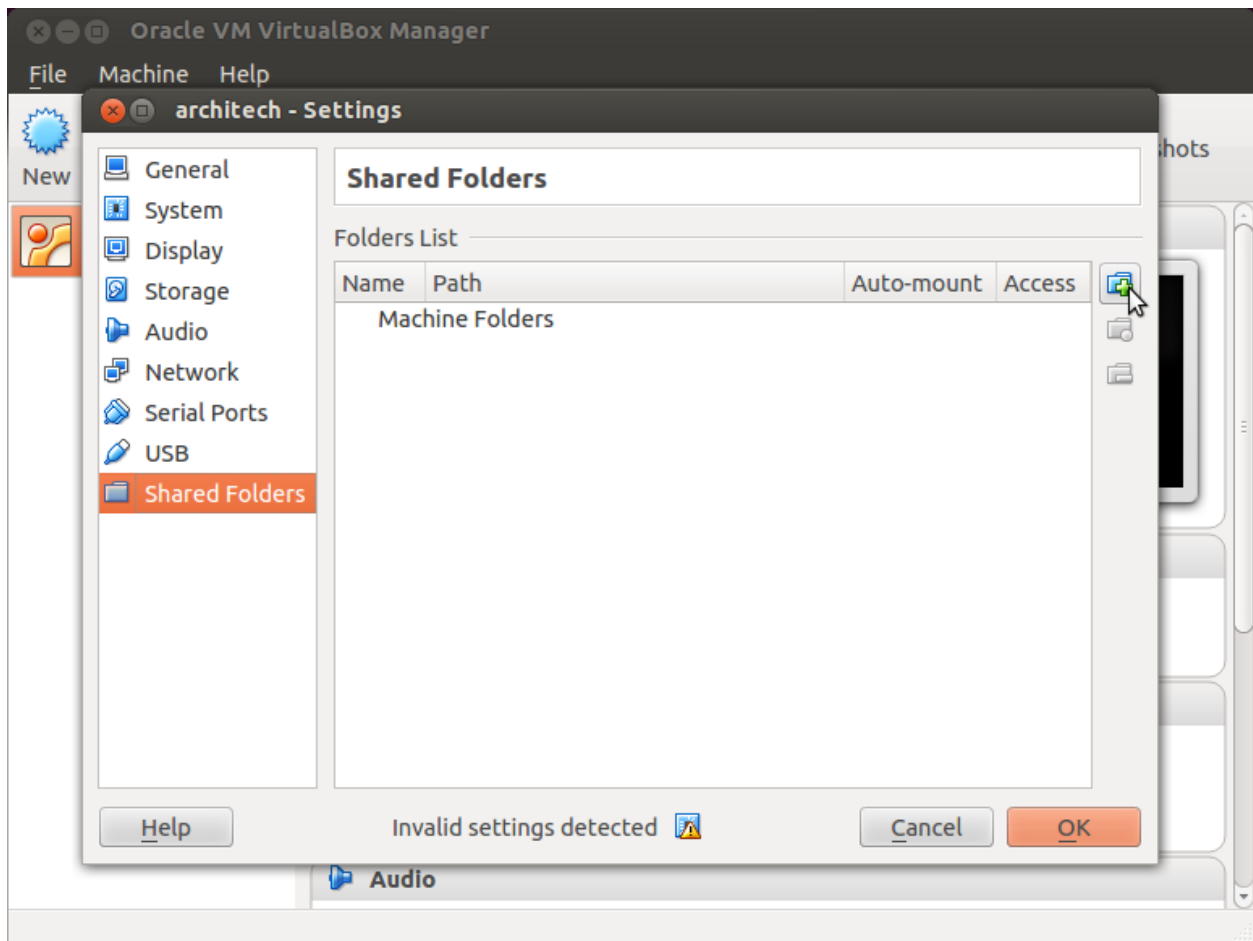
1. Select Architech's virtual machine from the list of virtual machines



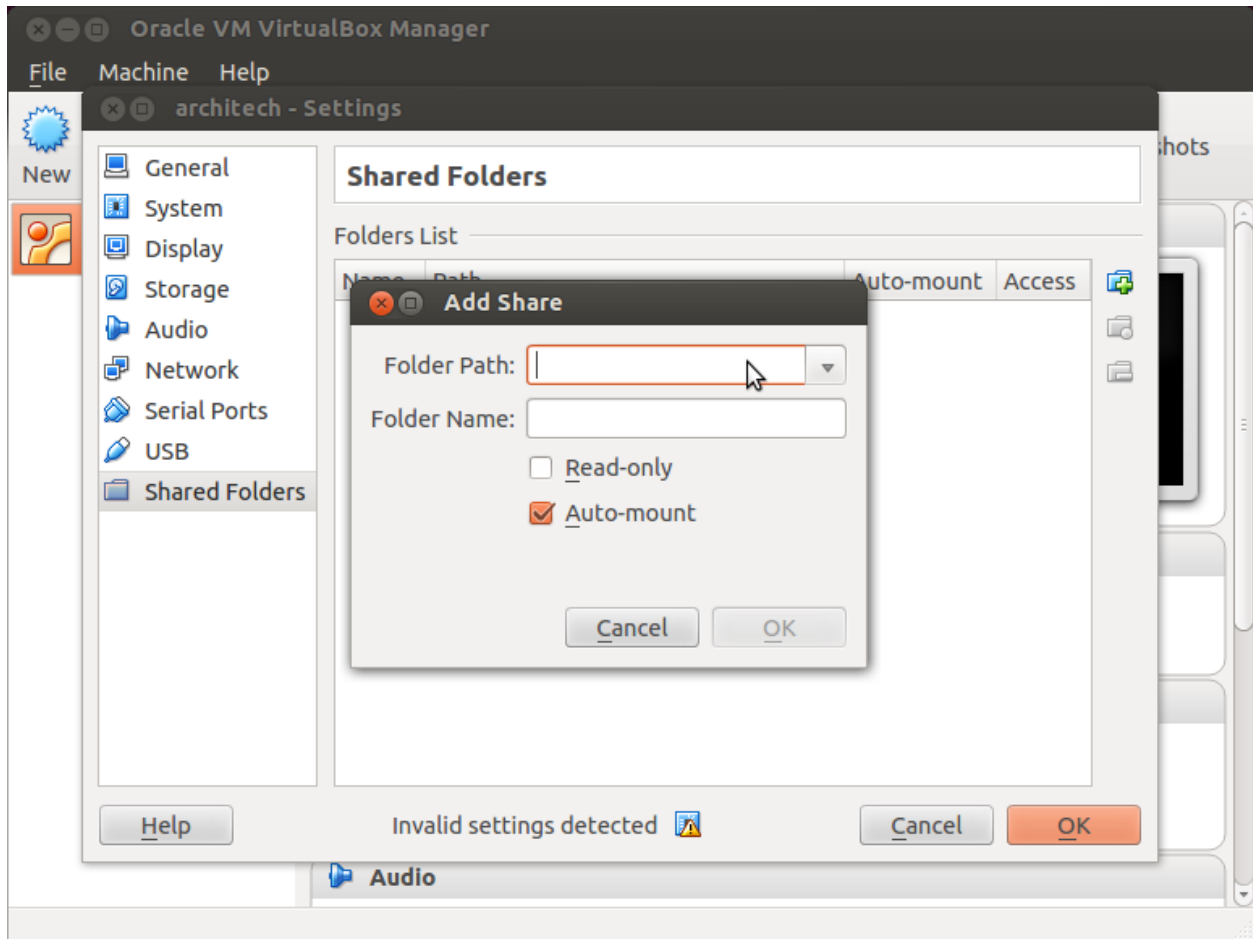
2. Click on *Settings*



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



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.

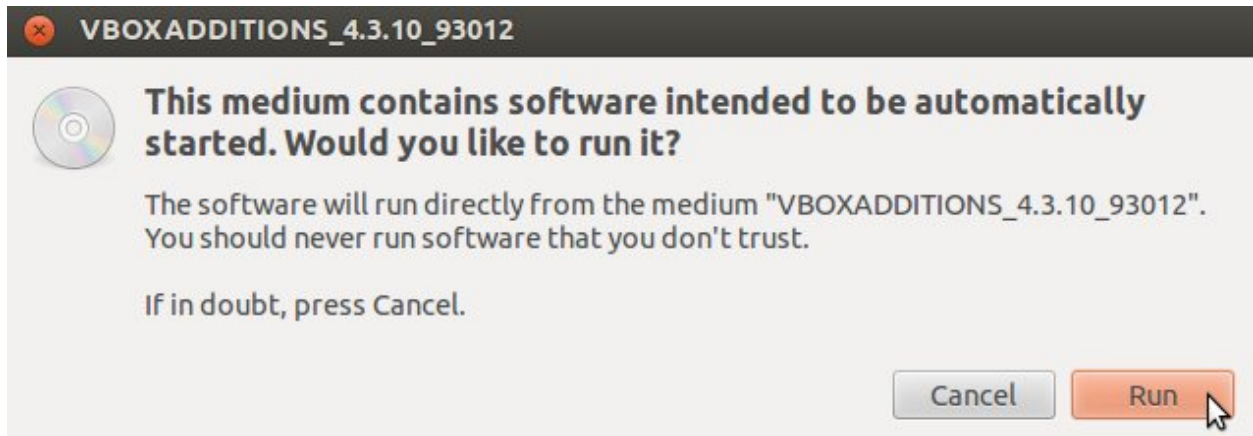
Install VBox Additions

The VBox additions 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.

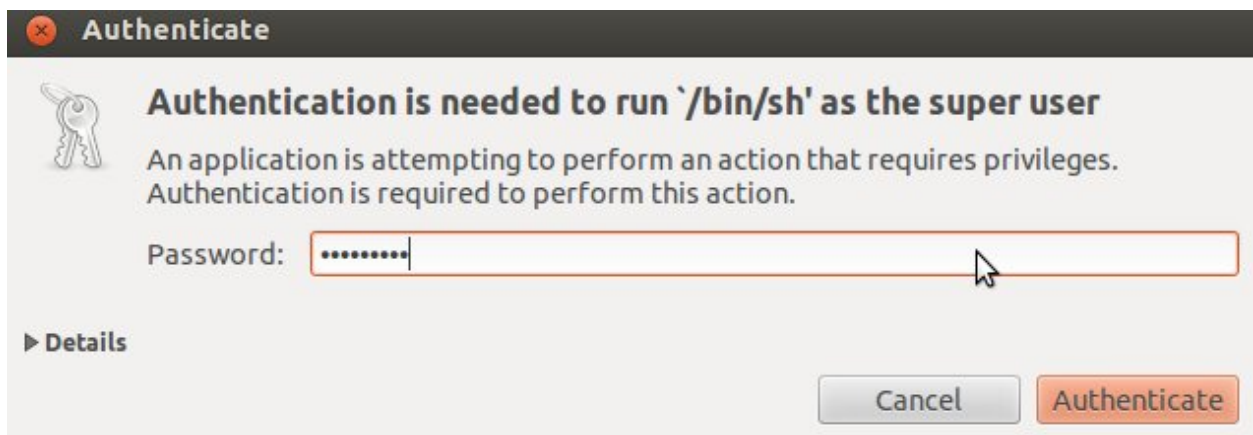
1. Starts the virtual machine



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



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



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:

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

- *eclipse* directory is where Eclipse IDE has been installed
- *java* directory is where the Java Virtual Machine has been installed (needed by Eclipse)
- *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,

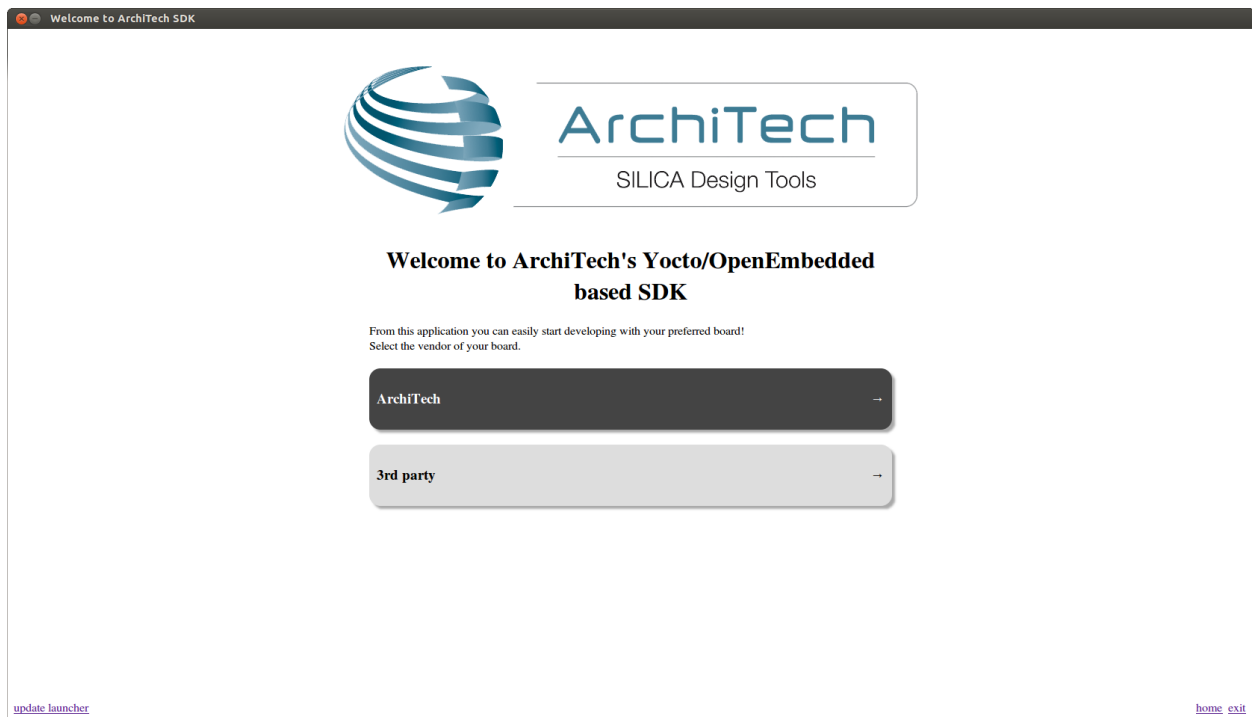
- *toolchain* is where the cross-toolchain has been installed installed
- *workspace* contains the the workspaces for Eclipse and Qt Creator IDEs
- *yocto* is where you find all the meta-layers Galileo requires, along with Poky and the build directory

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 choose if you want to work with Architech's boards or with partners' ones. For Galileo, choose **ArchiTech**.



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

A list of actions related to Galileo 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 Galileo 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 Galileo:

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 understanding 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 galileo 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 *dora* of *meta-clanton_v1.0.1*:

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.

Bootloader

On power on the boot sequence steps of Galileo are:

- BIOS starts
- grub loads the kernel and ramdisk
- the kernel loads the ramdisk drivers
- the kernel loads the sdcard rootfs

The bootloader used by Galileo is **Quark EDKII**. If You need to build the open source EDKII firmware for the Intel® Quark™ SoC download the Intel® Build and Software User Guide:

http://downloadmirror.intel.com/23962/eng/Quark_BSP_BuildandSWUserGuide_329687_007.pdf

The building guide starts at chapter 4.

If You want only to download the latest firmware go to:

<https://communities.intel.com/docs/DOC-22226>

Linux Kernel

The first step to compile the linux Kernel is to get the sources. You can get them from the Internet by cloning the proper repository and checking out the proper hash commit:

The second step is to properly patching the sources:

The third step is to copy the configuration:

and the next step is to compile them:

Warning:

X isn't a parameter. It's the number of processor's threads

By the end of the build process you will get **bzImage** under *arch/i386/boot*.

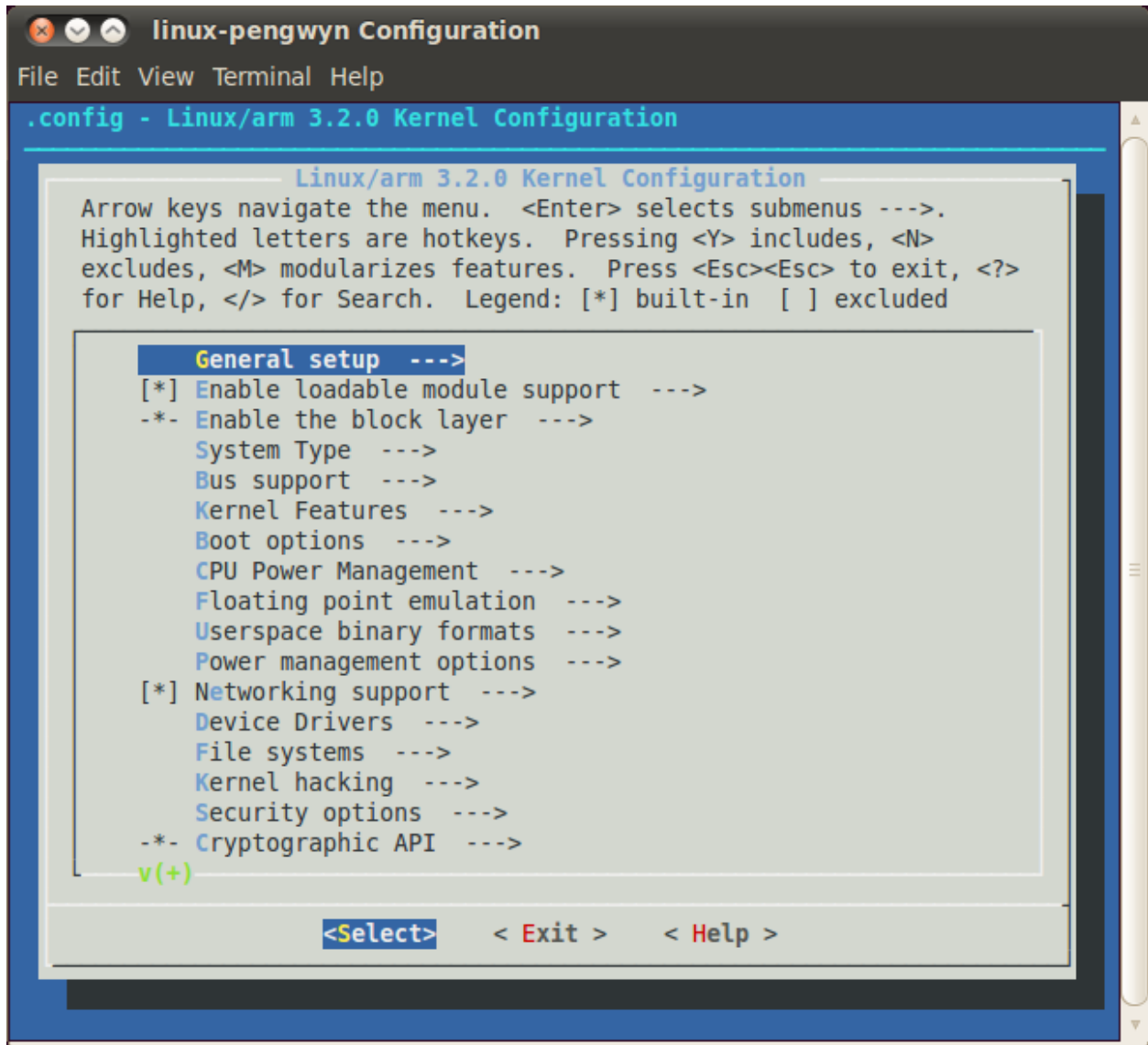
Now you need to download and unzip the RAMdisk rootfs from the Internet:

Then you have to copy the compiled modules to RAMdisk rootfs:

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:



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 (bzImage), along with the built kernel modules, will be placed under *tmp/deploy/images/galileo/* 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/galileo/yocto/meta-clanton_v1.0.1/yocto_build/tmp/deploy/images/*.

Root FS

By default, Galileo's Yocto/OpenEmbedded SDK will generate this files when you build an image:

- image-full-galileo-clanton.ext3*,
- core-image-minimal-initramfs-clanton.cpio.gz*

To open .ext3 file use:

now you can read and edit the files. At the end of your changes you must umount the folder

This files and the bootloader are to be copied in your final medium partition (on SD card or USB stick) on your host development system and used for build purposes with the Yocto Project.

The SD card (or USB stick) must meet the following requirements:

- SD card must be formatted as FAT or FAT32.
- SD card size must be 32GB (or smaller) and SDHC format. SDXC format is not supported.

Now you are rady to copy the files to the final device:

Warning:

The content of the media will be lost forever!

After the copy completes, run:

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 Galileo is located under:

and the splash screen has a facility (a button located under Galileo'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 Galileo is `galileo`:

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, bblayers, 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. *image-full-galileo* 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.

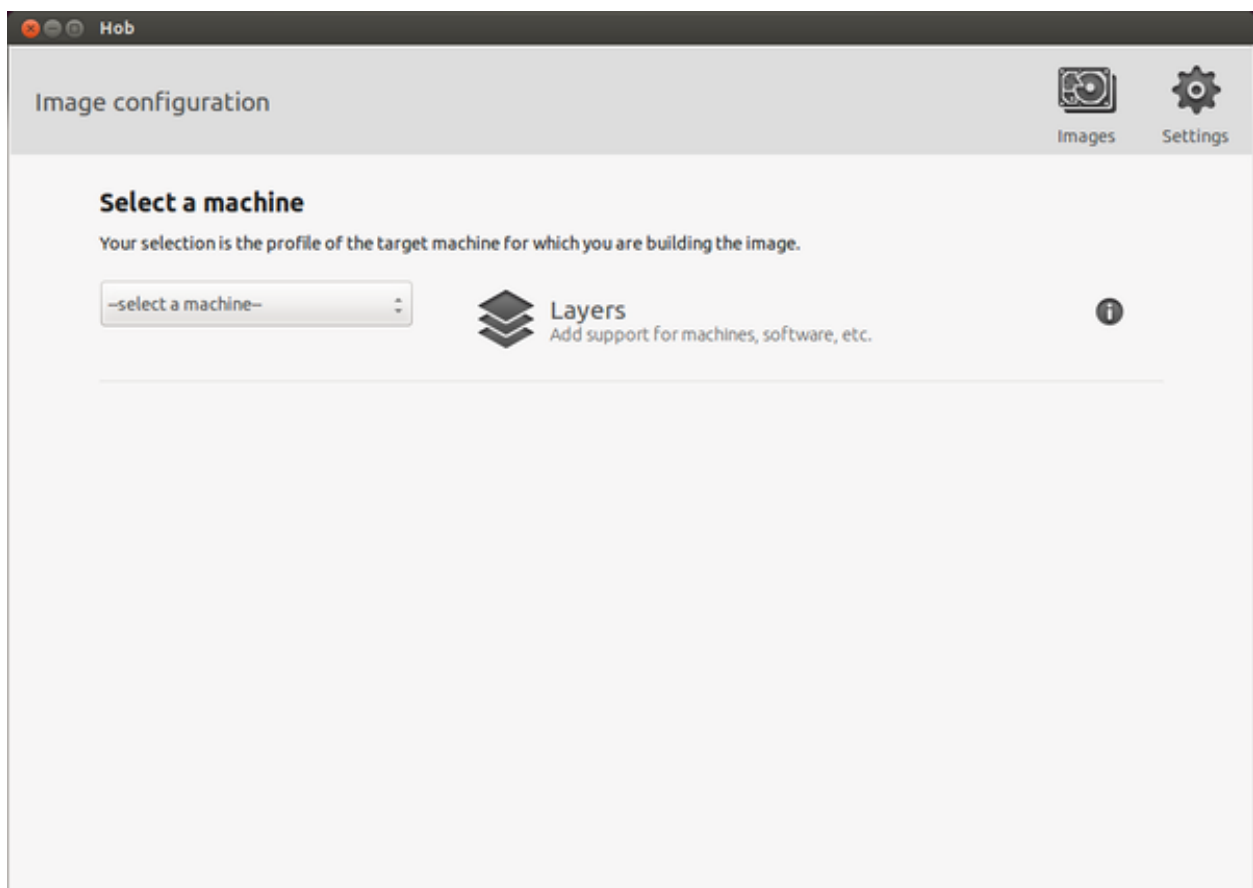
Hob

Hob is a graphical interface for Bitbake. It can be called once Bitbake environment has been setup (see [Bitbake](#)) like this:

Host

hob

once open, you are required to select the machine you want to compile against



after that, you can select the image you want to build and, of course, you can customize it.

Eclipse

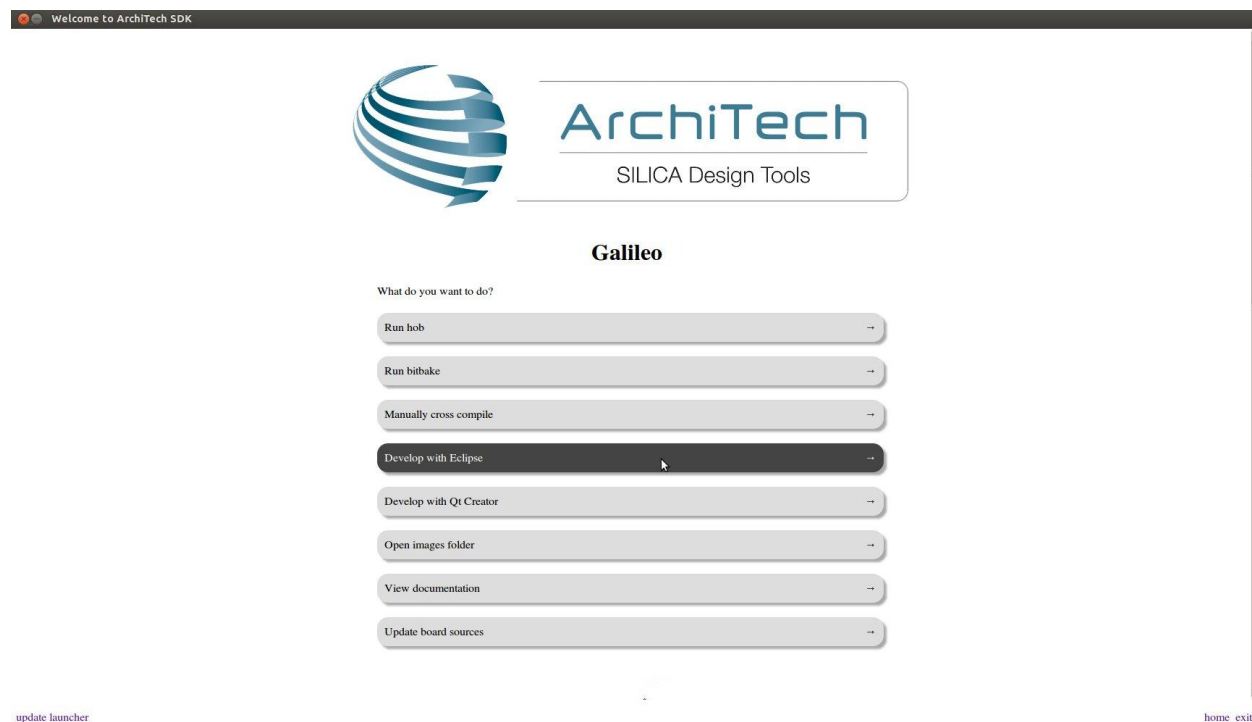
Eclipse is an integrated development environment (IDE). It contains a base workspace and the Yocto plug-in system to compile and debug a program for Galileo. Hereafter, the operating system that runs the IDE/debugger will be named host machine, and the board being debugged will be named target machine. 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
- Galileo *powered up* with the aforementioned root file system
- a working *serial console* terminal
- a working *network* connection between your workstation and the board (connector *10/100 LAN*), 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.

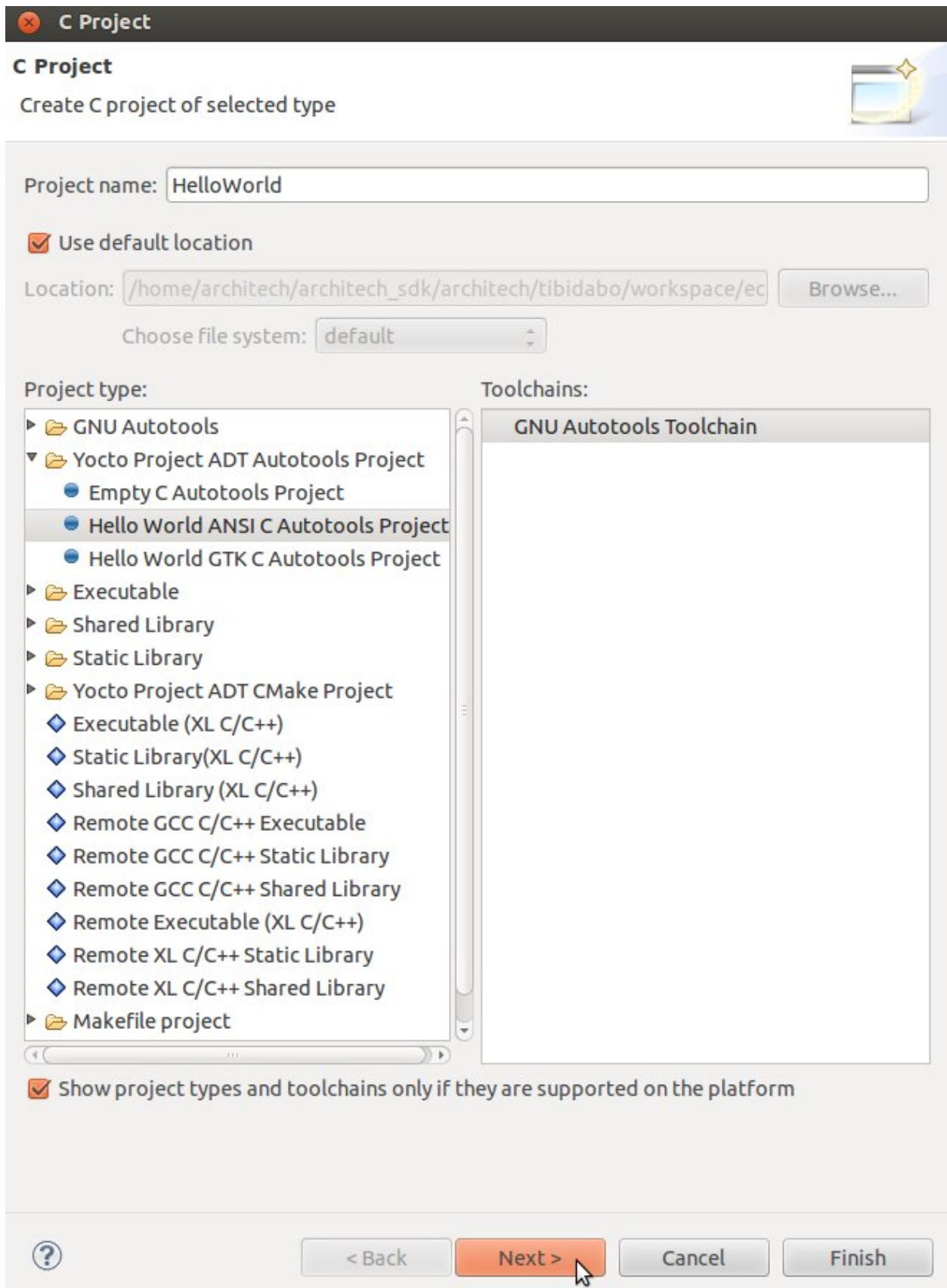
Creating the Project

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.

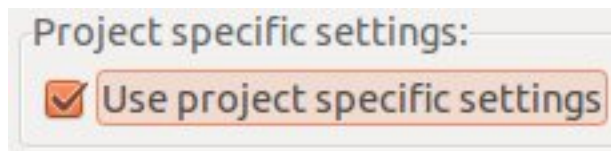


- Put a name in the Project *name:* field. Do not use hyphens as part of the name.
- Click *Next*.
- Add information in the *Author* and *Copyright* notice fields.
- Be sure the *License* field is correct.
- Click *Finish*.

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*

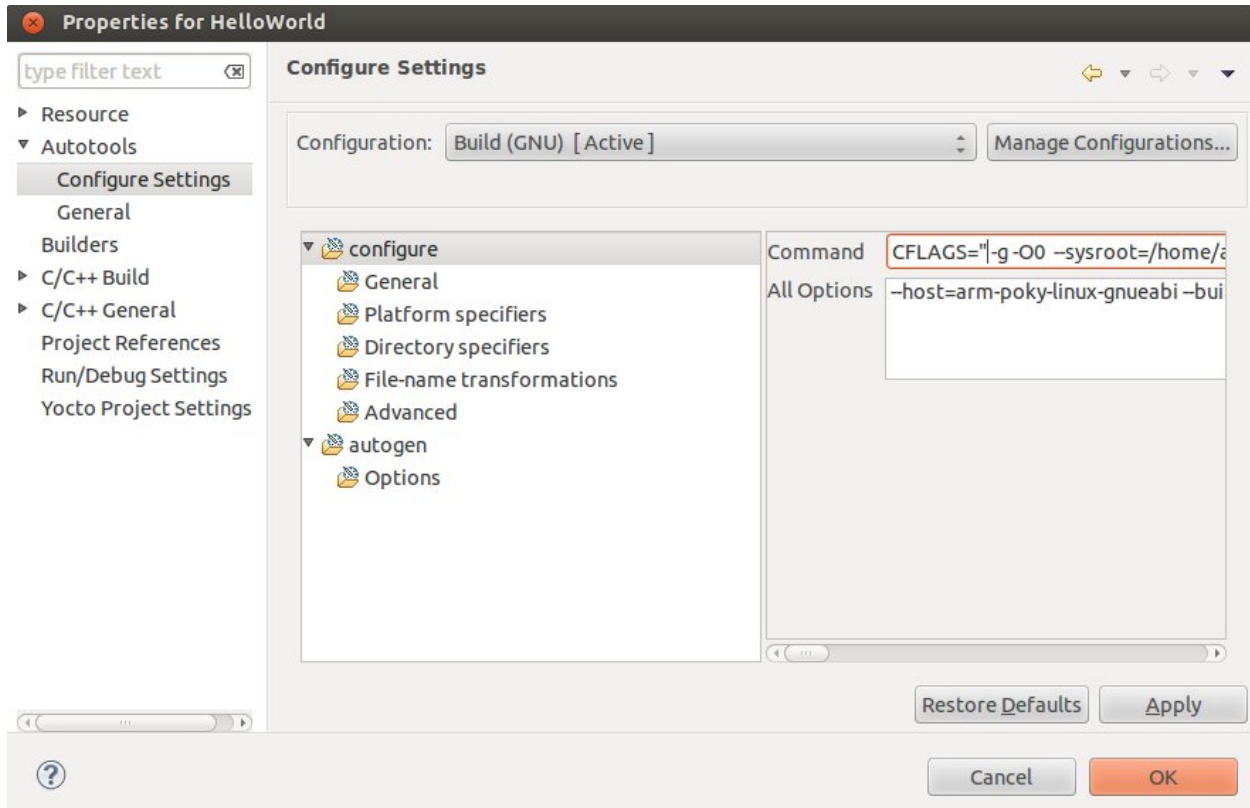


Building the Project

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/galileo/sysroot` subdirectories.

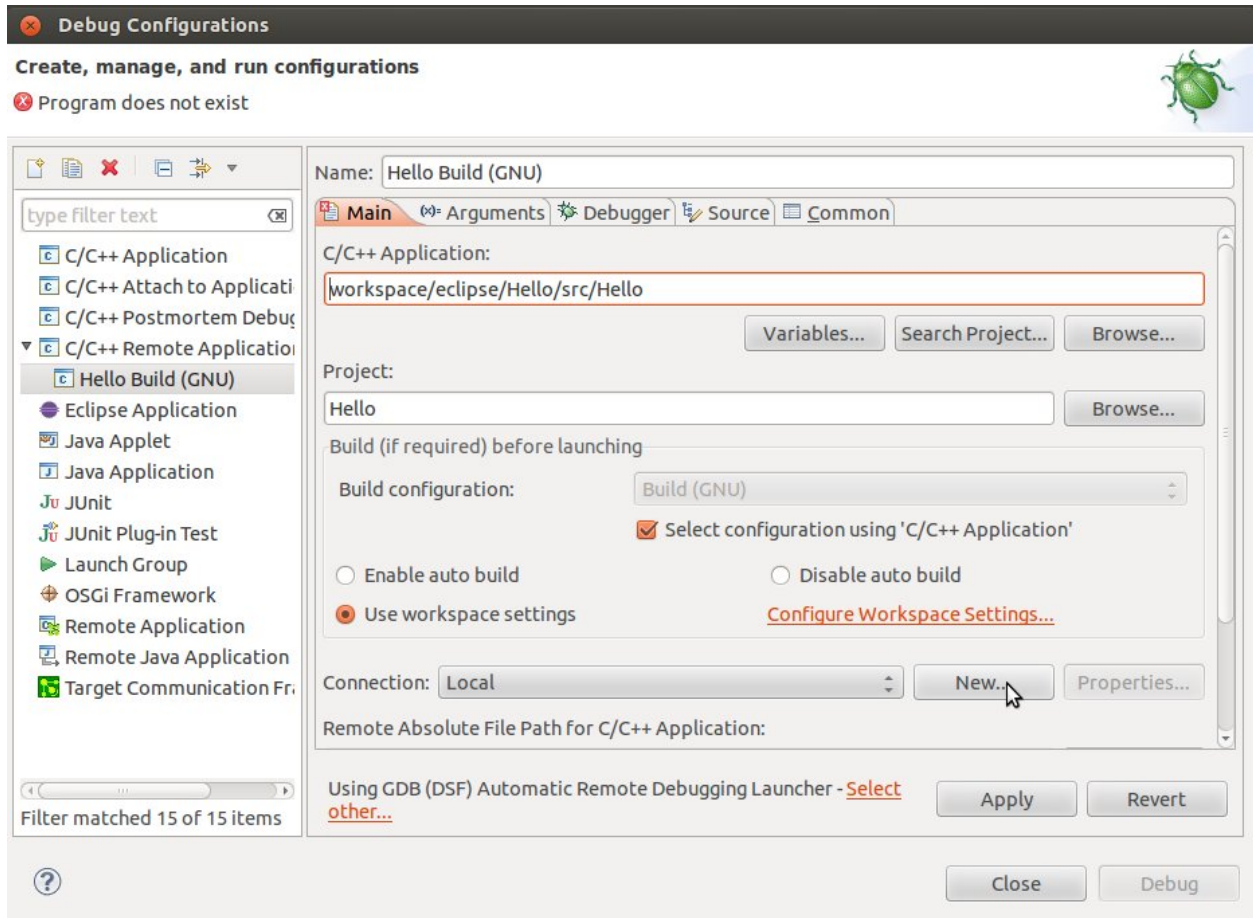


Deploying and Debugging the Application

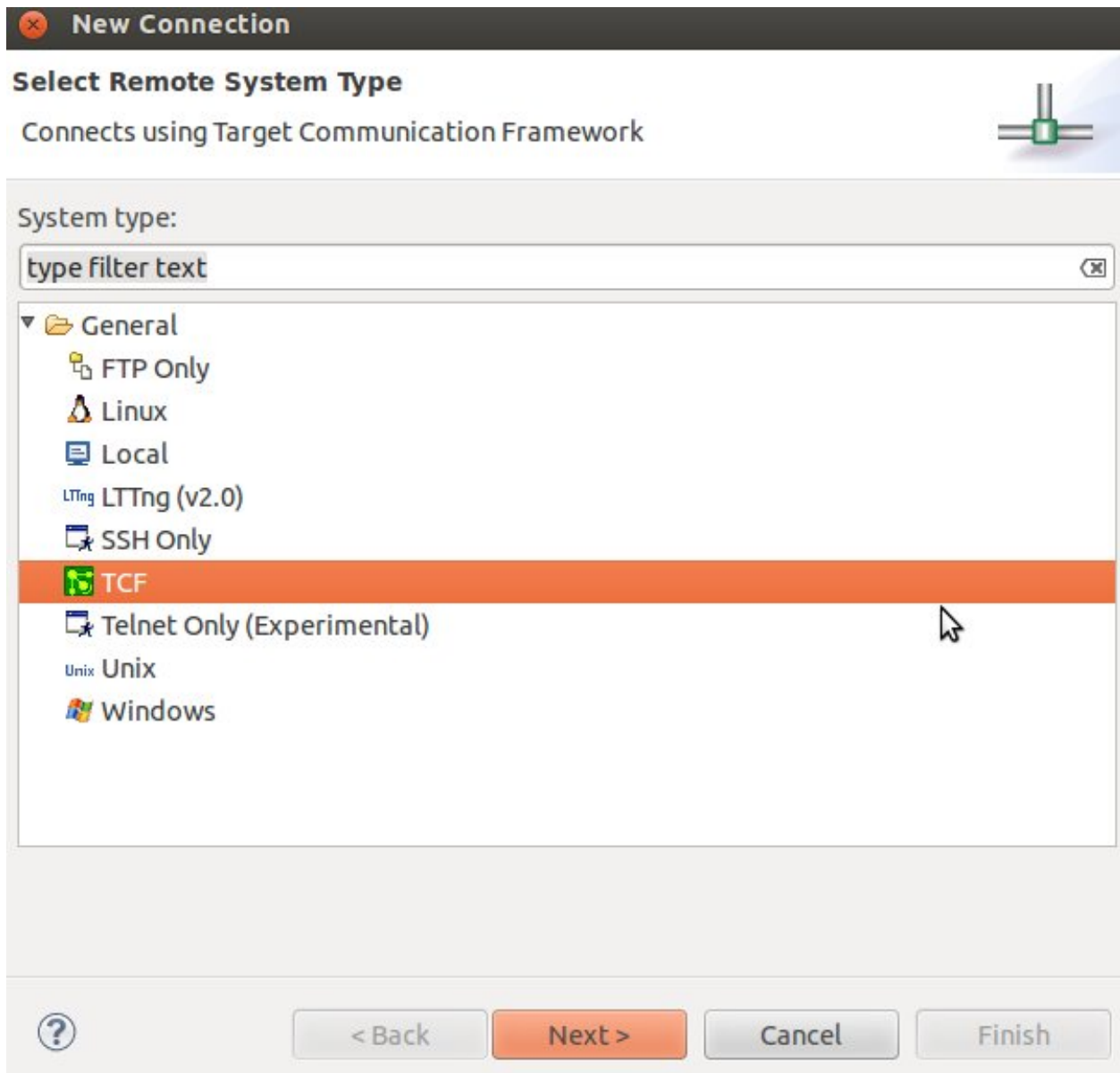
Connect Galileo console to your PC and power-on the board. Once you built the project and the board is running the image, use minicom to run **tcf-agent** program in target board:

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.



- 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)

New Connection

Remote TCF System Connection

Define connection information

Parent profile: architech

Host name: 192.168.0.10

Connection name: 192.168.0.10

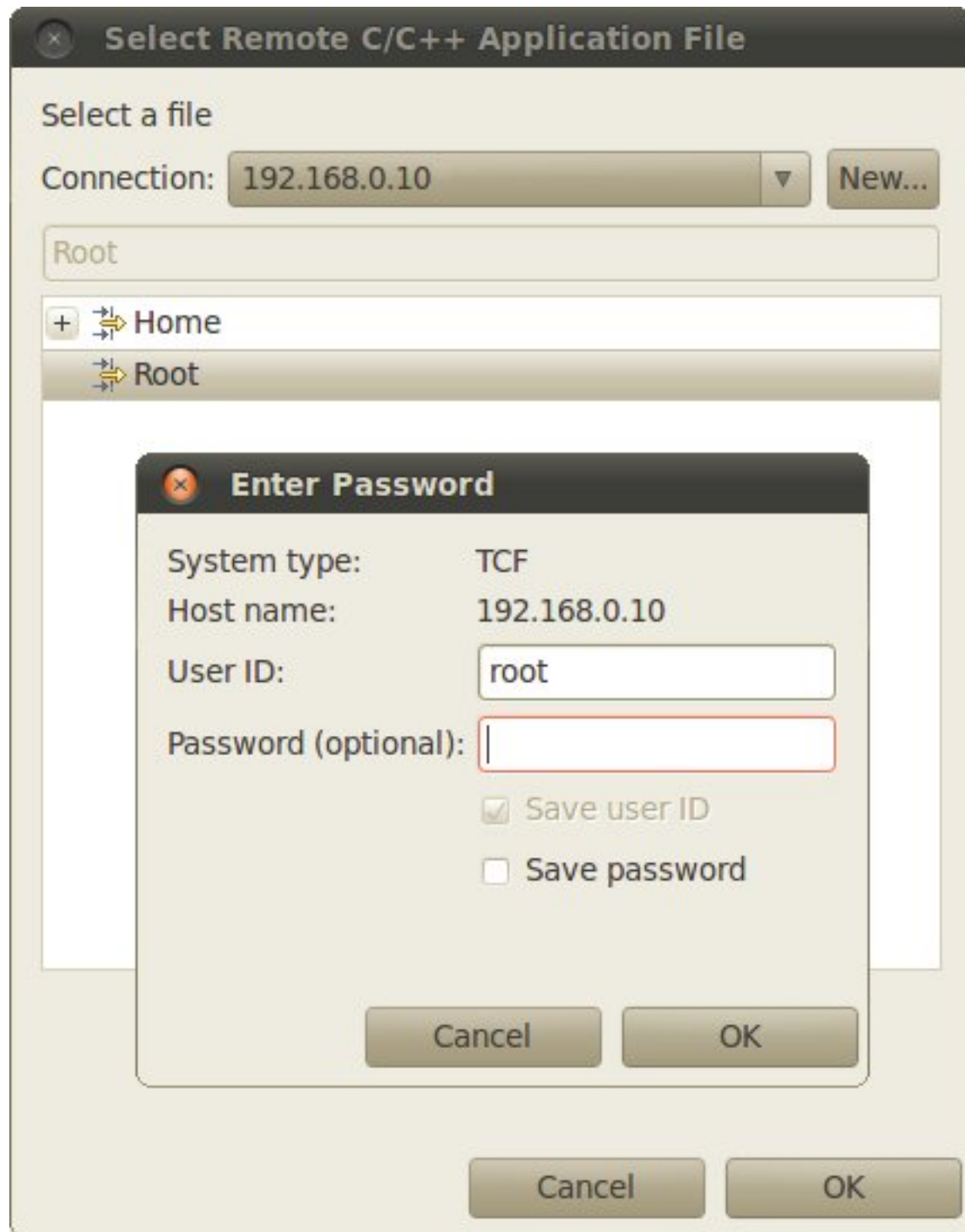
Description:

☒ Verify host name

[Configure proxy settings](#)

? < Back Next > Cancel Finish

- 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.

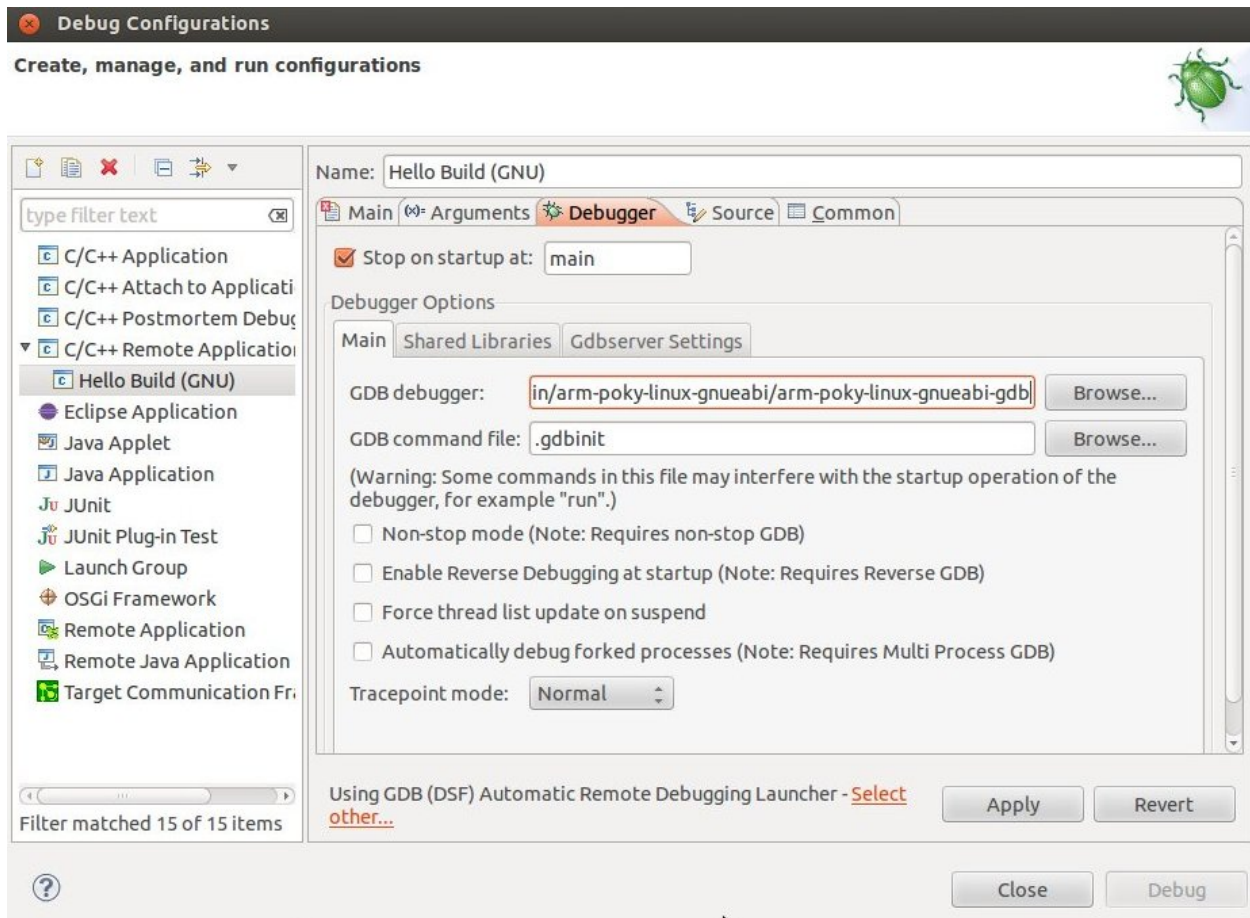


- Enter also in the target path the name of the application you want to debug. (e.g. HelloWorld)

Connection: 192.168.0.10

Remote Absolute File Path for C/C++ Application:
/home/root/HelloWorld

- Select *Debugger* tab



- 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.

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 Galileo can be found here:

<https://communities.intel.com/docs/DOC-21822>

<https://communities.intel.com/docs/DOC-21824>

Power-On

Galileo takes the power from connector **5V PWR**. The board is shipped with an external power adapter.



To assemble it, take the power socket adapter compatible with your country, plug it in the power adapter.



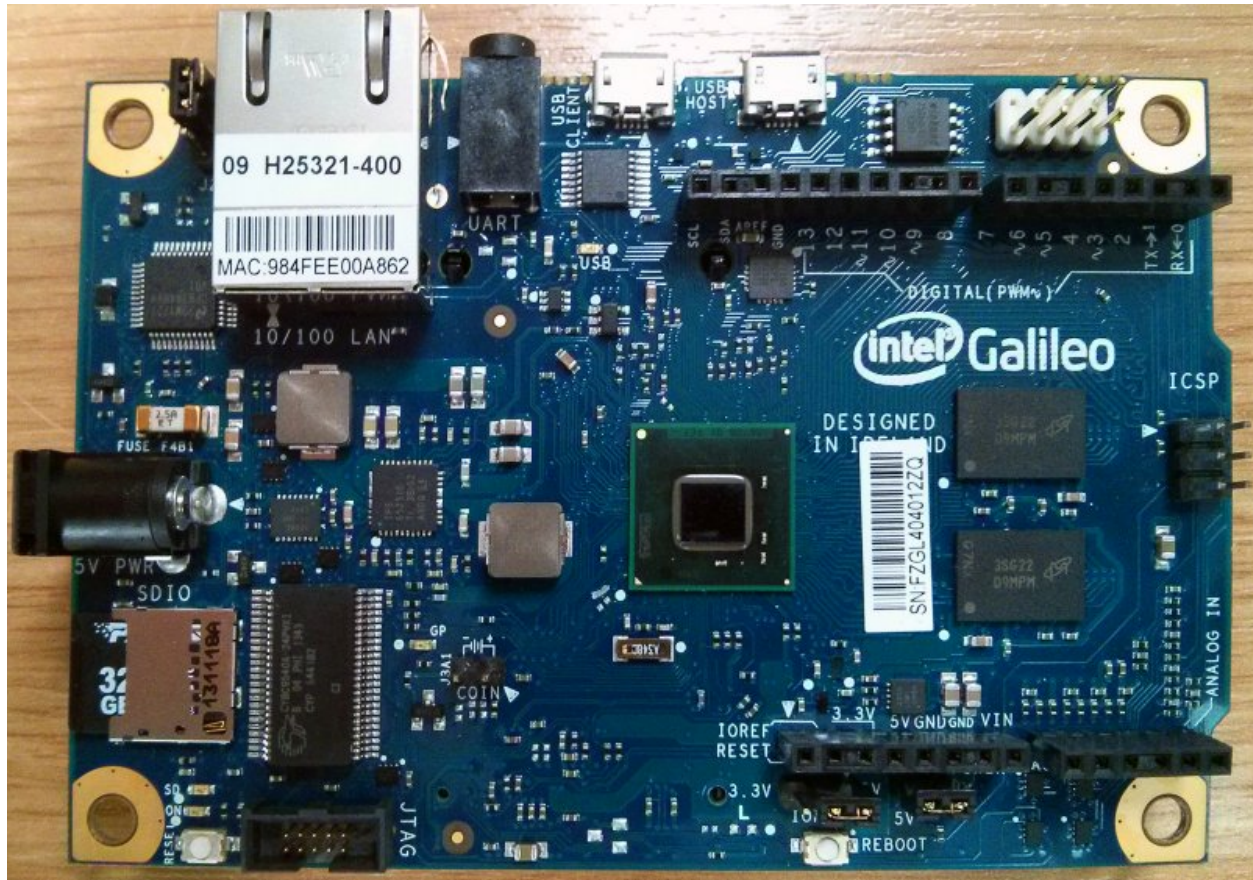
When in position, you should hear a slight *click*.



To power-on the board, just connect the external power adapter to Galileo connector **5V PWR**.

Serial Console

On Galileo there is the dedicated serial console connector **UART** (audio mini-jack)



which you can connect, by the serial to mini-jack UART port 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 `ttyUSBX` or `ttySX` device 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/ttyS0`.

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/ttyS0`, just replace `/dev/ttyS0` 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 Galileo board boots from SD card (or USB stick).

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 bandwidth. 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.

Galileo

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 preceded 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*” contains 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 typically 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 easily 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 simplify 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.

D

Debug, [73](#)

P

Project, [69](#)