Published in Cartesi

Cartesi Foundation Follow
May 16 · 9 min read · Listen



# Guest Post: How OpenCV cross compiles in The Blockchain OS.

Cross-Compiling OpenCV library for the Cartesi Machine RISC-V emulator.


*By Marcus Souza — Locus Custom Software*

**Introduction to the author**

Marcus Souza is from Paudalho, Pernambuco — Brazil, where he works as a Software Engineer at Locus Custom Software. MSc. at the Federal University of

with Cartesi Rollups, showing how to enable OpenCV to be used with the Rollups DApps.

#### What is the Cartesi Machine?

Cartesi Machine is the engine of Cartesi, The Blockchain OS, which is a layer-2 platform for the development and deployment of scalable decentralized applications (DApps). The Blockchain OS offers a Linux operating system coupled with a blockchain infrastructure, which allows DApps to be developed in familiar programming languages like Python without the need to write Solidity code. New to Cartesi? Learn more about The Blockchain OS here.

#### What about OpenCV?

OpenCV is a great tool for image processing and performing computer vision tasks. It is an open-source library that can be used to perform tasks like face detection, objection tracking, landmark detection, and much more. It supports multiple languages including Python, Java, and C++.

> **Some Amazing Applications of OpenCV Library - Analytics Vidhya**
>
> equipped with hundreds of useful functions and algorithms, which are all freely available to us. Some of these...
>
> www.analyticsvidhya.com

#### Why use OpenCV and Cartesi Machine together?

Running OpenCV inside the Cartesi Machine will enable DApps developers to explore, in a decentralized manner, complex subjects such as fingerprint spoof detection in biometrics and to feed machine learning models with features extracted from images for pattern recognition.

This tutorial will guide you through the necessary steps to cross-compile OpenCV for the Cartesi Machine RISC-V emulator using the command line.

#### Premise:

- The toolchain is designed for a target device (Cartesi Machine) and may be customized to meet unique requirements.

## Part 1: The RISC-V GNU Compiler Toolchain

Building the toolchain to cross-compile OpenCV and using it in a Cartesi Machine can be tricky. It is **highly recommended** to get the already built toolchain from: https://github.com/cartesi/image-toolchain, but the user can also build it from scratch. For that, reproduce the steps below.

#### Getting the toolchain from the Cartesi image-toolchain

The safest way is using the already existing toolchain from the cartesi image-toolchain repository. Once the cloning process ends, execute the commands to build and run as mentioned in their readme.

```
$make build
$make run
```

Running this command enables the user to access the docker image in console mode. With that, the user can access /opt/riscv folder, seeing its contents.

```
$root@toolchain-env:/opt/riscv/riscv64-cartesi-linux-gnu# ls
bin build.log.bz2 include lib libexec riscv64-cartesi-linux-gnu
share
$root@toolchain-env:/opt/riscv/riscv64-cartesi-linux-gnu#
```

Now, copy this folder to the host machine, and all the requirements are ready to compile the OpenCV.

#### Building the toolchain

But, if the choice is actually to build the toolchain, clone it from the RISC-V official GitHub. Once the dependencies are installed, it is straightforward to compile.

*1. Prerequisites*


```
$ sudo apt update && sudo apt upgrade
$ sudo apt install gcc g++ git make cmake python python3 gcc-
multilib vim autoconf automake autotools-dev curl libmpc-dev
libmpfr-dev libgmp-dev gawk build-essential bison flex texinfo
gperf libtool patchutils bc zlib1g-dev libexpat-dev pkg-config
libglib2.0-dev cmake-gui
```

### *2. Clone the repository and get the submodules*

The toolchain may be installed in the appropriate directory by specifying the `PREFIX` during the configuration process, and cloning will be done in the `home(~)` directory.

```
# Around 6.65 GB of the disk will be used.

$ cd ~
$ git clone https://github.com/riscv/riscv-gnu-toolchain.git
$ cd riscv-gnu-toolchain
$ git submodule update — init — recursive
```

### *3. Configuring the toolchain*

The configuration compatible with the Cartesi Machine is `— march=rv64ima — mabi=lp64`. So, the configuration is like the following:

```
$ ./configure — prefix=/opt/riscv — with-arch=rv64ima — with-
abi=lp64
```

The GCC documentation includes the options for `—- march` and `—- mabi`.

### *4. Building the toolchain*

## Part 2: Cross-compiling OpenCV

OpenCV runs on many hardware platforms and makes use of the `SIMD` (Single Instruction Multiple Data) accelerations on the ones that withstand it. OpenCV provides a relatively practical method to port the optimized kernels to a whole new architecture, as long as it withstands `SIMD`/vector instructions.

In this case, we want to use it normally inside the Cartesi Machine as in any Linux operating system. For that, do the following.

#### Get the prerequisites for OpenCV

```
# You can make a .sh script for all those commands below.

$ sudo apt install build-essential cmake git unzip pkg-config
$ sudo apt install libjpeg-dev libpng-dev libtiff-dev
$ sudo apt install libavcodec-dev libavformat-dev libswscale-dev
$ sudo apt install libgtk2.0-dev libcanberra-gtk*
$ sudo apt install python3-dev python3-numpy python3-pip
$ sudo apt install libxvidcore-dev libx264-dev libgtk-3-dev
$ sudo apt install libtbb2 libtbb-dev libdc1394–22-dev
$ sudo apt install libv4l-dev v4l-utils
$ sudo apt install libgstreamer1.0-dev libgstreamer-plugins-
base1.0-dev
$ sudo apt install libavresample-dev libvorbis-dev libxine2-dev
$ sudo apt install libfaac-dev libmp3lame-dev libtheora-dev
$ sudo apt install libopencore-amrnb-dev libopencore-amrwb-dev
$ sudo apt install libopenblas-dev libatlas-base-dev libblas-dev
$ sudo apt install liblapack-dev libeigen3-dev gfortran
$ sudo apt install libhdf5-dev protobuf-compiler
$ sudo apt install libprotobuf-dev libgoogle-glog-dev libgflags-
dev
# a symlink to videodev.h
$ cd /usr/include/linux
$ sudo ln -s -f ../libv4l1-videodev.h videodev.h
$ cd ~
```

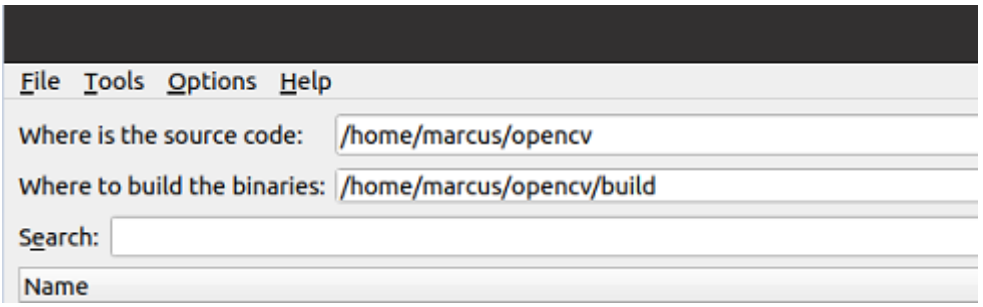
#### Cloning OpenCV

Now that all the pre-requisites are set, get to the OpenCV repository.

### Building OpenCV for RISC-V

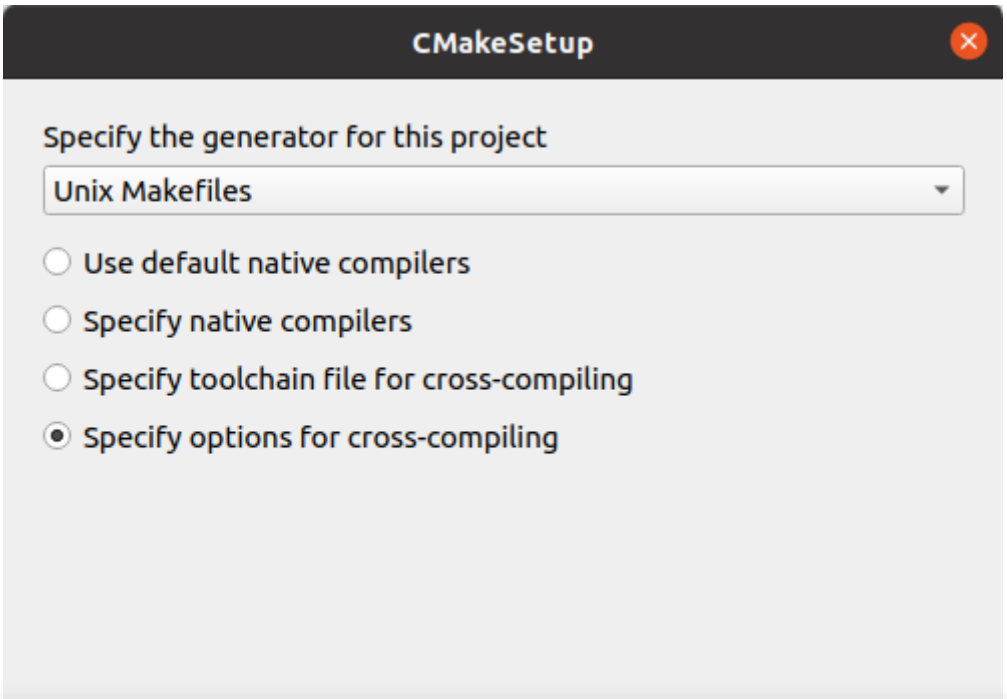
The CMake can be configured in the command line or via GUI. For the sake of simplicity, the steps below show the process with cmake-gui.

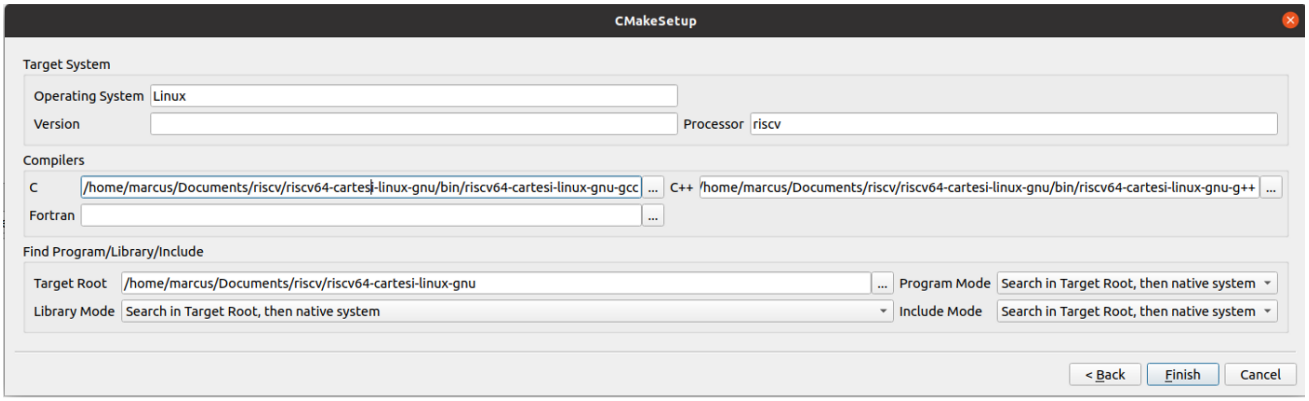
1. Create a cmake-gui instance in the **build** directory itself by typing `$cmake-gui` .



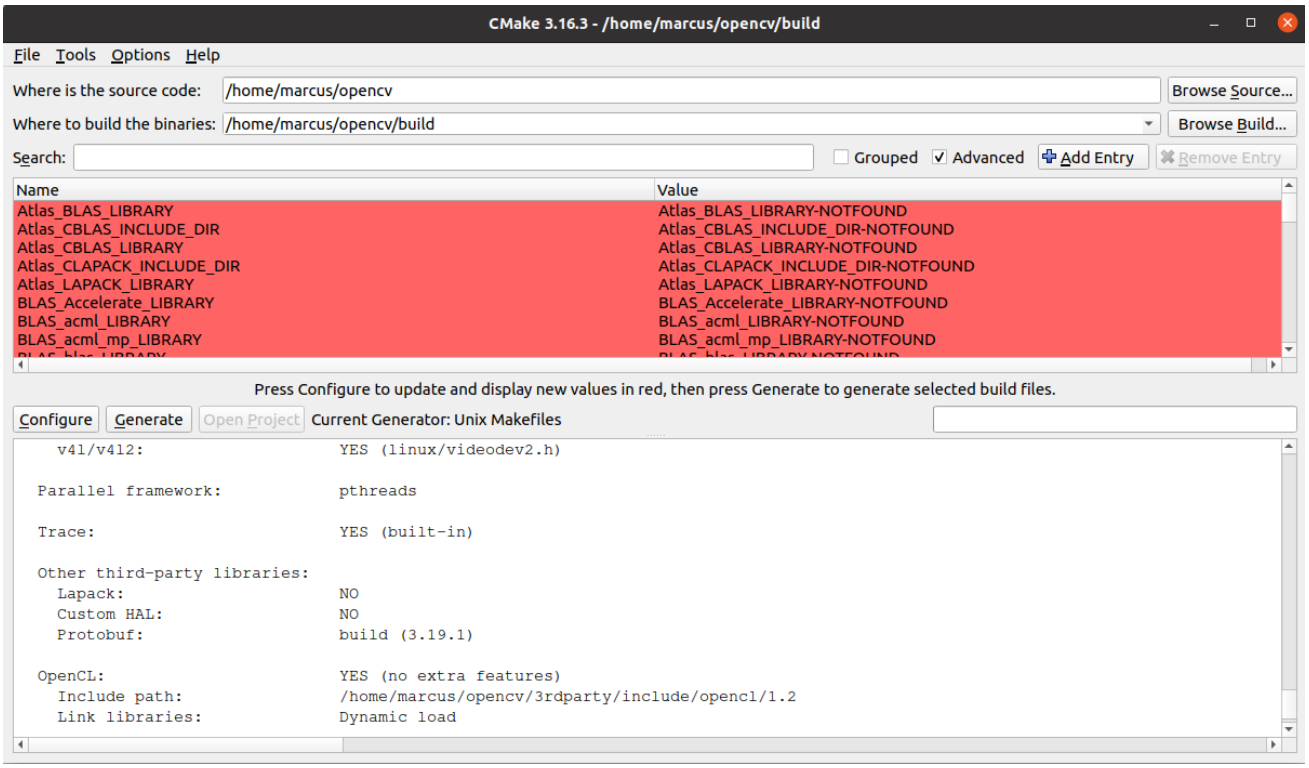

2. Take a closer look at *Where is the source code* and *Where to build the binaries* and set the path accordingly.

3. After you've set the appropriate paths, click configure, and a window will appear. Set the project's generator to Unix Makefiles and tick the box next to Specify cross-compilation options.



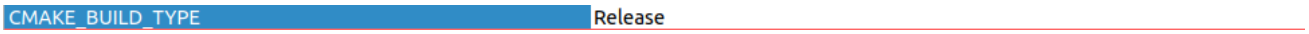



5. The GUI will appear like this once you click Finish:

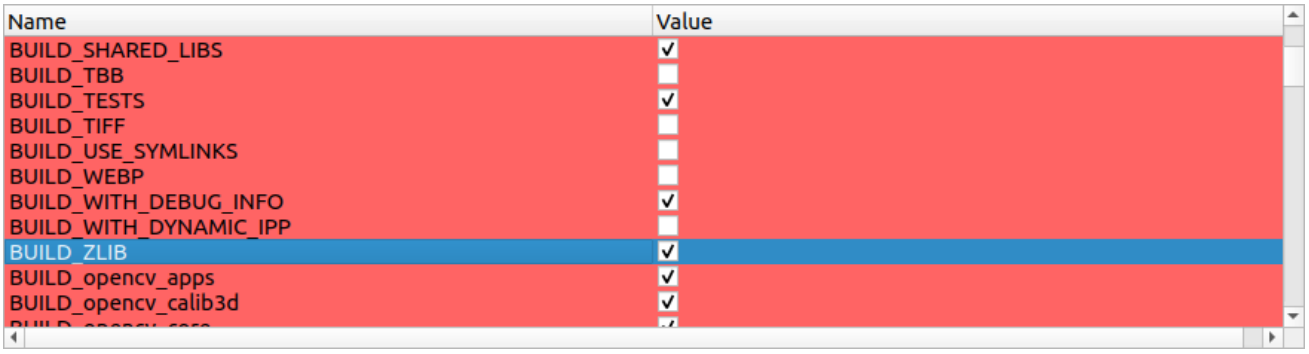



6. The modules to be installed may be checked here. As an example:

- CMAKE_BUILD_TYPE to Release.

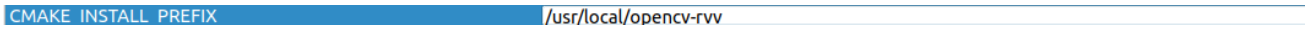



- Check BUILD_ZLIB.




- Set CMAKE_LINKER_FLAGS to **-lrt -lpthread**.


- Change the CMAKE INSTALL PREFIX to a separate directory to avoid interfering with the local architecture libraries.




- Disable WITH_1394


8. Once all these values have been set, click the generate button to generate the Makefile and exit.

After exiting the CMake-GUI, build and install the OpenCV at the prefix that was given earlier.

## Part 3: Put the OpenCV in the Cartesi Machine

To put the OpenCV inside the Cartesi Machine, this can be done either mounting

```
marcus@cartesiVM:~/Documents$ sudo mount -o loop rootfs.ext2
/mnt/cartesi
```

This will mount the rootfs system in the `/mnt/cartesi path` . Simply copy the OpenCV cross-compiled folder inside the filesystem (either with the command line or with nautilus explorer). Before copying, be sure that the filesystem has enough free space. If not, its size can be increased with the normal commands for it. For this tutorial, the copying folder is `/mnt/cartesi/usr/local/opencv-rvv` . With that, OpenCV is ready to be used inside the Cartesi Machine!

## Part 4: Testing the built library inside the Cartesi Machine

In this part, the operation of converting an image from RGB to grayscale inside a Cartesi Machine will be used as an example.

Code to convert an input image to grayscale and save it on disk:

```
#include <opencv2/opencv.hpp>
#include “opencv2/imgproc/imgproc.hpp”
#include <string>
#include <iostream>

int main(int argc, char* argv[]){
 cv::Mat imagem;
 cv::Mat imgGrayscale;

 imagem = cv::imread(argv[1], cv::IMREAD_COLOR);

 cv::cvtColor(imagem,imgGrayscale,cv::COLOR_BGR2GRAY);
 cv::imwrite(“imggray.jpg”, imgGrayscale);
 printf(“DONE!”);
 return 0;
}
```

Cross-compile the above code using the following command:

With this, an executable named 'lerImagem' will be compiled and ready to run inside the Cartesi Machine. It can either be used building a DApp or using it directly, copying in the same way OpenCV was copied. The same is illustrated below:

```
cartesi-machine:/mnt/echo-dapp # ./lerImagem cartesi.jpg
DONE!cartesi-machine:/mnt/echo-dapp # ls
cartesi.jpg  echo-server  imggray.jpg  lerImagem    lost+found   run.sh
cartesi-machine:/mnt/echo-dapp #
```

### Common Errors:

1. The header cannot be located.

```
fatal error: opencv2/opencv.hpp: No such file or directory
```

Solution: Include this `-I /usr/local/opencv-rvv/include/opencv4` while compilation.

2. Cannot locate the library file.

```
cannot find -lopencv_core ld: cannot find -lopencv_imgcodecs
```

Solution: Include this `-L /usr/local/opencv-rvv/lib` while compilation.

3. After a successful compilation, the dynamic library cannot be detected during runtime loading.

```
error while loading shared libraries: libopencv_core.so.4.5:
cannot open shared object file: No such file or directory
```

Solution: Export the library path (before compilation).

world. Locus provides services such as CTO as a service, Staff Augmentation, Squad Outsource, QA Test Factor, Cloud Security, Digital Product Design, and Design Inception. Locus turns technological dilemmas that seem unsolvable into opportunities for growth, bringing fast solutions with guaranteed quality. The world has changed, and the demands of companies and individuals alike. The need marks the present for innovation and agility.

## About Cartesi

The Blockchain OS is a decentralized layer-2 infrastructure that supports Linux and mainstream software components. For the first time, developers can code scalable smart contracts with rich software tools, libraries, and the services they're used to, bridging the gap between mainstream software and blockchain.

Cartesi is enabling millions of new startups and their developers to use The Blockchain OS and bring Linux applications on board. With a groundbreaking virtual machine, optimistic rollups, and side-chains, Cartesi paves the way for developers of all kinds, to build the next generation of blockchain apps.

Welcome to The Blockchain OS, home to what's next.

## Follow Cartesi across official channels:

Telegram Announcements | Telegram | Discord (Development Community) | Reddit | Twitter | Facebook | Instagram | Youtube | Github | Cartesi Improvement Proposal (CIP) | Website

Thanks to Marcus Vinicius