

GateMate™ FPGA

Toolchain Installation User Guide





Cologne Chip AG
Eintrachtstr. 113
50668 Köln

Tel.: +49 (0) 221 / 91 24-0
Fax: +49 (0) 221 / 91 24-100

<https://colognechip.com>
info@colognechip.com

Copyright 2019 - 2024 Cologne Chip AG

All Rights Reserved

The information presented can not be considered as assured characteristics. Data can change without notice. Parts of the information presented may be protected by patent or other rights. Cologne Chip products are not designed, intended, or authorized for use in any application intended to support or sustain life, or for any other application in which the failure of the Cologne Chip product could create a situation where personal injury or death may occur.

Contents

About this Document	4
1 Introduction	5
1.1 Toolchain Overview	5
1.2 Hard- and Software Requirements	6
2 Download pre-build Software	7
3 Build Software from Source	8
3.1 Build Synthesis Software	8
3.1.1 Build Yosys for Linux	8
3.1.2 Build GHDL for Linux	9
3.2 Build Programmer Software	10
4 Install Optional Simulation Tools	11
4.1 Icarus Verilog	11
4.2 GTKWave	11
5 Quick Start	12
5.1 Synthesize Design	12
5.2 Implement Design	14
5.3 Program Bitfile	15

About this Document

This user guide describes the toolchain installation for the Cologne Chip GateMate™ series and is part of the GateMate™ documentation collection.

For more information please refer to the following documents:

- [Technology Brief of GateMate™ FPGA](#)
- [DS1001 – GateMate™ FPGA CCGM1A1 Datasheet](#)
- [DS1002 – GateMate™ FPGA Programmer Board Datasheet](#)
- [DS1003 – GateMate™ FPGA Evaluation Board Datasheet](#)

Cologne Chip provides a comprehensive technical support. Please visit our website for more information or contact our support team.

Revision History

This datasheet is constantly updated. The latest version of the document can be found following the link below:

[UG1002 – GateMate™ FPGA Toolchain Installation User Guide](#)

Date	Remarks
January 2024	Minor changes in Section 5.3 on page 15.
November 2022	Call of Place & Route added for Windows in Section 5.2 on page 14.
September 2022	Comprehensive revision and notes on configuration via SPI in Section 5.3 from page 15.
August 2022	Comprehensive revision.
April 2022	Pre-build binary option of openFPGALoader for Windows added.
March 2022	Initial release.

1 Introduction

1.1 Toolchain Overview

This user guide describes the toolchain installation for the Cologne Chip GateMate™ series. It covers building latest software from source or downloading pre-build binaries for Linux and Windows environments.

An exemplary toolchain flow from design entry to configuration is illustrated in Figure 1.

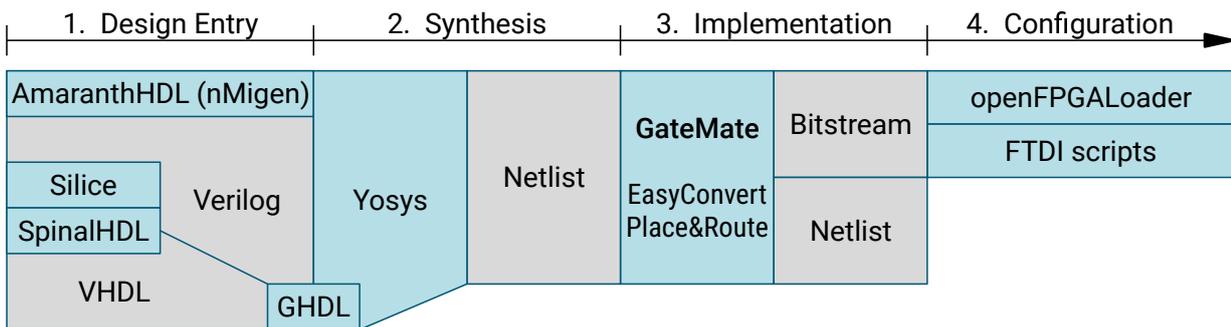


Figure 1: Exemplary Toolchain Flow

The Yosys Open SYnthesis Suite [↗](#) is used to perform RTL synthesis. It has extensive Verilog support. VHDL sources can be synthesized via GHDL [↗](#) through the ghdl-yosys-plugin [↗](#). Other HDLs or tools with Verilog backend can also be used.

Synthesis generates a gate-level representation of the design entry in form of a Verilog netlist of architecture-specific primitives. It can be simulated or passed to the Place and Route tool for implementation and bitstream generation. Furthermore, simulation of the resulting netlist with back-annotated timing delays can be done using third-party simulators such as Icarus Verilog [↗](#) together with the GTKWave [↗](#) waveform viewer.

Configuration bitstreams are loaded into the FPGA or an external flash memory via openFPGALoader [↗](#).

1.2 Hard- and Software Requirements

- Computer with Linux or Windows operating system
- Supported Linux environments:
 - Debian-based Linux (Debian, Ubuntu, ...) with apt package manager
 - Arch-based Linux (Arch, Manjaro, ...) with pacman package manager
 - Red Hat-based Linux (Fedora, ...) with dnf or yum package manager
- Windows environments:
 - Windows 7 or later, 64 bit
 - Zadig USB driver installer (<https://zadig.akeo.ie/>)
- (optional) GateMate™ FPGA Evaluation Board (Link [↗](#))
- Third-party simulator, e.g. Icarus Verilog (<http://iverilog.icarus.com/>)
- Third-party waveform viewer, e.g. GTKWave (<http://gtkwave.sourceforge.net/>)

2 Download pre-build Software

Cologne Chip provides pre-build packages containing all toolchain binaries and sample projects. Please check availability on the official website. Login to your account at <https://colognechip.com/mygatamate/> and follow the menu to the software download section.

There are packages for Windows and Linux. WSL users are advised to use the Windows packages. Chapter 5 describes the contents of the packages and how to operate the software using the included scripts.

3 Build Software from Source

This chapter is optional and is intended for advanced users who want to compile the toolchain from source.

3.1 Build Synthesis Software

The Yosys Open SYnthesis Suite [↗](#) is used to perform RTL synthesis. VHDL support is only available by installing the GHDL [↗](#) and ghdl-yosys-plugin [↗](#) extensions.

3.1.1 Build Yosys for Linux

Install prerequisites for Debian/Ubuntu Linux with apt:

```
$ sudo apt install build-essential clang bison flex libreadline-dev  
  ↪ gawk tcl-dev libffi-dev git graphviz xdot pkg-config python3  
  ↪ libboost-system-dev libboost-python-dev libboost-filesystem-dev  
  ↪ zlib1g-dev
```

Install prerequisites for Arch-based Linux with pacman:

```
$ sudo pacman -S base-devel git tcl zlib
```

Install prerequisites for Red Hat-based Linux, e.g. Fedora, with dnf:

```
$ sudo dnf install make clang tcl-devel zlib-devel readline-devel  
  ↪ libffi-devel bison flex
```

The following commands download, compile and install the source files regardless of the Linux distribution used. First, clone the repository with git and change into the directory.

```
$ git clone https://github.com/YosysHQ/yosys.git  
$ cd yosys
```

Inside the Yosys directory, configure the build system to use a specific compiler such as gcc or clang.

```
$ make config-clang
```

The typical build and installation process for Linux is invoked with the following two commands.

```
$ make -j$(nproc)  
$ sudo make install
```

3.1.2 Build GHDL for Linux

Both GHDL and `ghdl-yosys-plugin` are only required for VHDL support in Yosys. Please proceed with the steps from Section 3.1.1 if VHDL support is not required.

Install prerequisites for Debian/Ubuntu Linux with `apt`:

```
$ sudo apt install gnat zlib1g-dev
```

Install prerequisites for Arch-based Linux with `pacman`:

```
$ sudo pacman -S gcc-ada zlib
```

Install prerequisites for Red Hat-based Linux, e.g. Fedora, with `dnf`:

```
$ sudo dnf install gcc-gnat zlib
```

The following commands download, compile and install the source files regardless of the Linux distribution used. First, clone the repository with `git` and change into the directory.

```
$ git clone https://github.com/ghdl/ghdl.git  
$ cd ghdl
```

The configuration, build and installation process is invoked with the following three commands.

```
$ ./configure --prefix=/usr/local  
$ make  
$ sudo make install
```

Now clone the `ghdl-yosys-plugin` with `git` and change into the directory.

```
$ cd ..  
$ git clone https://github.com/ghdl/ghdl-yosys-plugin.git  
$ cd ghdl-yosys-plugin
```

The Makefile will find the `ghdl` installation in `/usr/local`. To start building the plugin, run:

```
$ make
```

The output is a shared library (`ghdl.so` on GNU/Linux). To install the module, the library must be copied to `YOSYS_PREFIX/share/yosys/plugins/ghdl.so`, where `YOSYS_PREFIX` is the installation path of Yosys. This can be achieved through a make target:

```
$ sudo make install
```

The library can then be used by Yosys directly, e.g. with:

```
$ yosys -m ghdl
```

The Quick Start Guide in Section 5 describes how to synthesize VHDL sources using the GHDL plugin.

3.2 Build Programmer Software

openFPGALoader [↗](#) integrates well into the toolchain. Version v0.7.0 or later is required.

Build OpenFPGALoader for Linux

Install prerequisites for Debian/Ubuntu Linux with apt:

```
$ sudo apt install libftdi1-2 libftdi1-dev libhidapi-hidraw0 libhidapi-  
↪ dev libudev-dev zlib1g-dev cmake pkg-config make g++
```

Install prerequisites for Arch-based Linux with pacman:

```
$ sudo pacman -S git cmake make gcc pkgconf libftdi libusb zlib hidapi
```

Install prerequisites for Red Hat-based Linux, e.g. Fedora, with dnf:

```
$ sudo dnf install cmake libftdi-devel zlib hidapi-devel systemd-devel
```

The following commands download, compile and install the source files regardless of the Linux distribution used. First, clone the repository with git and change into the directory.

```
$ git clone https://github.com/trabucayre/openFPGALoader.git  
$ cd openFPGALoader
```

Next, use the following three commands to prepare cmake:

```
$ mkdir build  
$ cd build  
$ cmake ..
```

The build and installation process is invoked with the following two commands:

```
$ make -j$(nproc)  
$ sudo make install
```

4 Install Optional Simulation Tools

This section describes some useful third-party simulation and viewing tools.

4.1 Icarus Verilog

Icarus Verilog [↗](#) is a widely used Verilog compiler, e.g. for simulation. It is available for a wide range of platforms.

Install from package manager for Debian/Ubuntu Linux with apt:

```
$ sudo apt install iverilog
```

Install from package manager for Arch-based Linux with pacman:

```
$ sudo pacman -S iverilog
```

Install from package manager for Red Hat-based Linux, e.g. Fedora, with dnf:

```
$ sudo dnf install iverilog
```

Pre-build binaries for Windows are available for download here:

<https://bleyer.org/icarus/>

4.2 GTKWave

GTKWave [↗](#) is a waveform viewer which reads most available value change dump files and is available for a wide range of platforms.

Install from package manager for Debian/Ubuntu Linux with apt:

```
$ sudo apt install gtkwave
```

Install from package manager for Arch-based Linux with pacman:

```
$ sudo pacman -S gtkwave
```

Install from package manager for Red Hat-based Linux, e.g. Fedora, with dnf:

```
$ sudo dnf install gtkwave
```

Pre-build binaries for Windows are available for download here:

<http://gtkwave.sourceforge.net/>

5 Quick Start

This tutorial guides you through all toolchain components with an example project. Please make sure to setup or install all required software components from Sections 2 to 4.

The easiest way is to download the corresponding archive from the website:

`https://colognechip.com/downloads/cc-toolchain-win.zip`
or `https://colognechip.com/downloads/cc-toolchain-linux.zip`

In this tutorial, we will use the sample projects that come with the pre-built binaries. The `cc-toolchain-{win,linux}` package contains the two sub-directories `bin` and `workspace`.

- The `bin` directory contains all binaries for `yosys`, `p_r` and `openFPGALoader`.
- The `workspace` directory contains sample projects in Verilog and VHDL.

Linux or WSL users should use the provided Makefiles. Windows users can use the batch script in the respective project directory. Project directories have the following structure:

- The `log` directory will contain log files after synthesis and implementation.
- The `net` directory will contain a Verilog netlist after synthesis.
- The `sim` directory contains a simple testbench for post-synthesis or post-implementation simulation.
- The `src` directory contains all Verilog, VHDL and CCF constraint files.
- Both `run.bat` or `Makefile` scripts help calling the tools.

5.1 Synthesize Design

First change to the desired project directory with a console. Synthesis can be started via the supplied scripts or manually via command.

Run synthesis with all Verilog or VHDL files in the `src` directory (Linux or WSL):

```
$ make synth_vlog
```

```
$ make synth_vhdl
```

Run synthesis with all Verilog or VHDL files in the `src` directory (Windows):

```
$ run.bat synth_vlog
```

```
$ run.bat synth_vhdl
```

Basically, the script executes the following commands for Verilog or VHDL:

```
$ yosys -l yosys.log -p 'read_verilog <FILES>; synth_gatemate -top <TOP>
↪ > -vlog net/<TOP>_synth.v'
```

```
$ yosys -l yosys.log -p 'ghdl --warn-no-binding -C --ieee=synopsys <
↪ FILES> -e <TOP>; synth_gatemate -top <TOP> -vlog net/<TOP>_synth
↪ .v'
```

After successful synthesis, yosys generates a Verilog netlist and a log file in the net and log directories. The final parts of the log file displays information on primitive utilization after synthesis. Note that several elements can be combined during implementation, so final utilization reports may differ after implementation.

```
[...]
2.50. Printing statistics.

=== example ===

Number of wires:                8
Number of wire bits:           22
Number of public wires:        5
Number of public wire bits:    19
Number of memories:            0
Number of memory bits:        0
Number of processes:          0
Number of cells:               20
  CC_BUFG                       1
  CC_DFF                         8
  CC_IBUF                        2
  CC_L2T4                       1
  CC_OBUF                        8
  [...]
```

Post-Synthesis Simulation

Simulation of the synthesis netlist requires a third-party simulator and a waveform viewer such as Icarus Verilog [↗](#) and GTKWave [↗](#). The sample projects contain a testbench in the sim directory.

Run post-synthesis simulation (Linux or WSL):

```
$ make synth_sim
```

Basically, the script executes the following commands:

```
$ iverilog -o sim/<TOP>.vvp net/<TOP>_synth.v sim/<TOP>_tb.v ../../bin/
↪ yosys/share/gatemate/cells_sim.v
$ vvp -N sim/<TOP>.vvp
```

The resulting VCD can be opened with any waveform viewer.

```
$ make wave
```

5.2 Implement Design

The Cologne Chip Place & Route loads the Verilog netlist in the `net` directory. Constraints for pin assignments or CPE pre-placement are set with the CCF file in the `src` directory.

Run implementation (Linux or WSL):

```
$ make impl
```

Run implementation (Windows):

```
$ run.bat impl
```

Basically, the script executes the following command:

```
$ p_r -i net/<TOP>_synth.v -o <TOP> -ccf src/<TOP>.ccf > log/impl.log
```

Please check the [DS1001 – GateMate™ FPGA CCGM1A1 Datasheet](#) for a more detailed description of the available implementation parameters. Depending on the parameters, the tool generates at least the following output files:

- Log output in `log/impl.log`
- Configuration bitstream: `<TOP>_00.cfg.bit`
- Verilog netlist for post-implementation simulation: `<TOP>_00.v`
- SDF delay file for post-implementation simulation: `<TOP>_00.sdf`
- Pin file: `<TOP>_00.pin`
- Place file: `<TOP>_00.place`

Post-Implementation Simulation

Run post-synthesis simulation (Linux or WSL):

```
$ make impl_sim
```

Basically, the script executes the following commands:

```
$ iverilog -o sim/<TOP>.vvp <TOP>_00.v sim/<TOP>_tb.v ../../bin/p_r/  
→ cpelib.v  
$ vvp -N sim/<TOP>.vvp
```

The resulting VCD can be opened with any waveform viewer.

```
$ make wave
```

5.3 Program Bitfile

This tutorial assumes that e.g. an GateMate™ Evaluation Board is available. In Linux environments, openFPGALoader can be used to send the configuration via JTAG or SPI. Please make sure to set the corresponding CFG_MD on your evaluation board.

In Linux environments, it may be necessary to install some dependencies via the build-in package manager. openFPGALoader will report the missing packages.

In Windows environments, it is necessary to install USB drivers using Zadig [↗](#). Download the software and connect the GateMate™ FPGA Evaluation Board to your USB port. In the Zadig Window, select **Options > List All Devices** to refresh the device list. Then, unmark **Options > Ignore Hubs or Composite Parents**. From the drop-down list, select **GateMate™ FPGA Evaluation Board (Composite Parent)**. Now select **libusb-win32 (any version)** from the driver list and replace the drivers (see Figure 2).

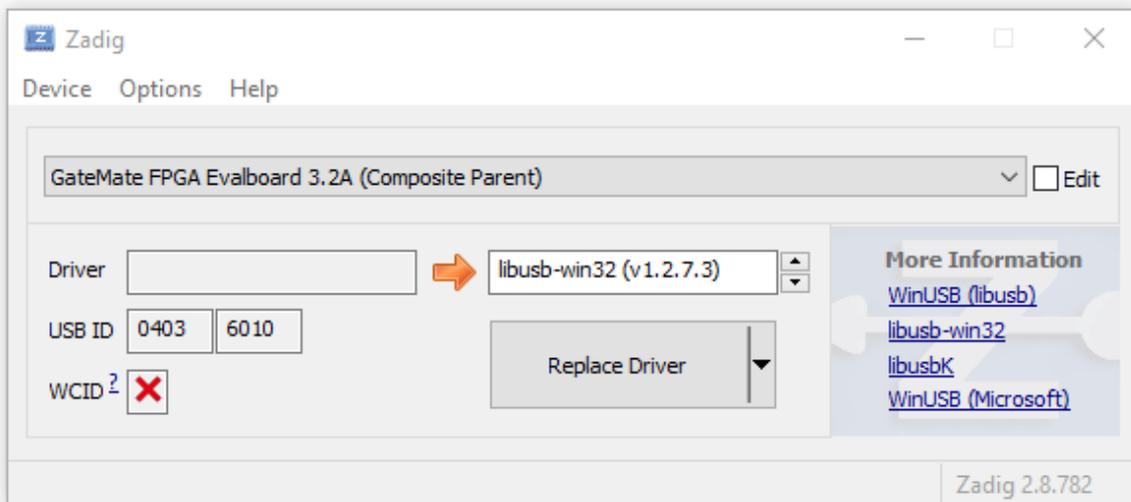


Figure 2: Zadig Window with selected GateMate™ FPGA Evaluation Board

Replacing drivers might take a moment. Your GateMate™ FPGA Evaluation Board should then be listed as **libusb-win32** devices in the Device Manager as shown in Figure 3.

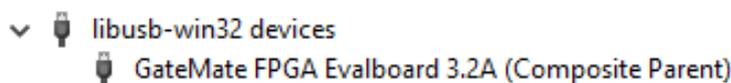


Figure 3: GateMate™ FPGA Evaluation Board in Device Manager

Configure the FPGA via JTAG (Linux or WSL):

```
$ make jtag
```

Configure the FPGA via JTAG (Windows):

```
$ run.bat jtag
```

The script executes the following command:

```
$ openFPGAloader -b gatemate_evb_jtag <TOP>_00.cfg.bit
```

After successful configuration via JTAG, the FPGA starts automatically.

Alternatively, the device can also be configured directly via SPI. The Makefile command is (Linux or WSL):

```
$ make spi
```

The batch command is (Windows):

```
$ run.bat spi
```

The script runs the following command:

```
$ openFPGAloader -b gatemate_evb_spi -m <TOP>_00.cfg.bit
```

The FPGA starts automatically after successful configuration via SPI.

Store the configuration bitstream to an external flash via JTAG-SPI-bypass (Linux or WSL):

```
$ make jtag-flash
```

Store the configuration bitstream to an external flash via JTAG-SPI-bypass (Windows):

```
$ run.bat jtag-flash
```

The script executes the following command:

```
$ openFPGAloader -b gatemate_evb_jtag -f --verify <TOP>_00.cfg.bit
```

Alternatively, the flash can also be programmed directly via SPI. The Makefile command is (Linux or WSL)

```
$ make spi-flash
```

The batch command is (Windows)

```
$ run.bat spi-flash
```

The script runs the following command:

```
$ openFPGALoader -b gatemate_evb_spi <TOP>_00.cfg.bit
```

After successful programming the external flash, the CFG_MD must be set to active SPI in order to start the FPGA from flash after reset.

GateMate™ FPGA
Toolchain Installation User Guide
UG1002
January 2024

