# MaSoCist Evaluation primer

| COLLABORATORS | | | |
|---|---|---|---|
| | *TITLE* : MaSoCist Evaluation primer | | |
| *ACTION* | *NAME* | *DATE* | *SIGNATURE* |
| WRITTEN BY | Martin Strubel | November 29, 2018 | |

| REVISION HISTORY | | | |
|---|---|---|---|
| NUMBER | DATE | DESCRIPTION | NAME |
| | | | |

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Overview

The MaSoCist distribution enables you to quickly design, maintain, document and automatically create a family of Soft core featured System on Chip solutions on various FPGA architectures. It is relying heavily on the Linux kernel config utility and the section5 device description XML language.

**Note**
The full MaSoCist development environment is supported for Unix operating systems only

This evaluation version is featured by the simple System On Chip design 'agathe'. It implements a microcontroller with the following peripherals:

- 2x16 GPIO
- IRQ controller
- SPI interface
- Up to 8 PWMs/Timers
- UART

See separate SoC documentation (`soc-agathe.pdf`) for details.

The MaSoCist environment is available as:

- Docker container. This runs in a virtual machine using boot2docker.

**Note**
Linux container support is dropped for future releases

## 1.1 Evaluation version specifics

The evaluation/opensource version lacks a few options. Therefore, a few configuration options can not be altered or have no effect. This behaviour is currently undocumented. The source is provided 'AS IS'.

Restrictions are present on:

1. SoC design functionality
2. Co-Simulation features, virtual hardware implementations
3. Configureable, pipelined CPU cores
4. Debugging (ICE) functionality

| Module | Description |
|---|---|
| ghdlex, netpp | free Co-Simulation extensions (partially OpenSource) |
| gensoc | commercial SoC generator utility |
| stdtap/uniemu | JTAG debugger (GDB debug agent) |

Table 1.1: Optional (proprietary) modules

### 1.1.1 C preprocessing

VHDL does not have the full conditional compilation option like the C language. However, the cpp (C preprocessor) is a powerful utility for macro expansion. This can be used to generate conditional or configureable VHDL code. The disadvantage is, that the programming environment requires an installed CPP processor. Code maintenance becomes slightly tricky, because an extra step is required, plus one needs to make sure not to edit the generated file instead of the actual source file.

These CPP processed files have the suffix .chdl. The corresponding Makefile rule in `generate.mk` is:

```
# Generate VHDL from CHDL:
%.vhdl : %.chdl $(TOPDIR)/.config
    cpp -w -P -o $@ -D__VHDL__ $(CHDL_FLAGS) -I$(TOPDIR)/hdl $<
```

There are cases where you would not want a macro expanded, like a `CONFIG_FOO` variable holding a time specification for VHDL. In this case you will have to undefine the macro and convert it to a VHDL constant inside the `global_config` package (more about this below). Example:

```
#include "autoconf.h"
#undef CONFIG_FOO
```

## 1.2 Licensing

The licensing of the MaSoCist distribution depends on its distribution's package tag, e.g. for a tar file: `masocist-$(DIST_TAG)-$(VERSION).tgz`. By default, `DIST_TAG="opensource"`.

When it comes to hardware designs, licenses such as the GPL (Gnu Public License) are hardly applicaple, in particular because it would complicate development under a dual licensing scheme. Since, by experience, complicated license schemes are seldom respected by hardware developers, an attempt is made to phrase a few rules as follows.

**NONCOMMERCIAL USAGE**

- You can use it for educational purposes or non-commercial home projects
- You will not get much free support, but you may of course feed back bugs
- You are encouraged to publish your changes, but noone will force you to it. Just respect the principle of fair use.

**COMMERCIAL USAGE**

- If you use the code in a commercial environment, you are free to do so, but you are *required* to publish changes made to the MaSoCist code base, including your own code that depends on MaSoCist functionality.
- If you are making a product that you are re-licensing or re-selling to others, you will need to acquire a license for the IP you use

The opensource variant also has the `CONFIG_OPENSOURCE` variable defined. Code under an exclusive OpenSource license will then throw a warning on compilation into a simulation. This is used to mark source released under the implicated OpenSource agreement. Removal of the `CONFIG_OPENSOURCE` checks are considered a violation of this OpenSource license.

If you wish to have a maintained custom package, you will have to sign up for a custom license agreement. This entitles you for a distribution tag. In this case, you are completely free to keep further development proprietary, EXCEPT changes made to the ghdlex package.

In any case, contributions made by third parties under an open source license agreement will always remain in the open source.

Note that you also have to follow the GHDL license agreements for distribution of simulation executables.

## 1.3 Target audience

The MaSoCist is made for hardware/software developers who mainly wish to play with configureable IP cores and OpenSource HDL CPU cores. The focus lies mostly in rapid prototyping and automatization. It is assumed, that:

- The developer is experienced with linux command line tools
- Independence of vendor specific tools is a strong desire
- No support is needed, and the "read the source, luke" principle is acceptable.

## 1.4 Board supply packages

The following boards are supported by this MaSoCist edition 'AS IS':

### 1.4.1 netpp node

The *netpp node* is a highly flexible SoC module based on a Xilinx Spartan 6-LX9 for process control and measurement systems. Highlights:

- Analog backend options:,
  - Up to six channel 10 bit ADC
  - Three channel differential 16 bit Sigma-Delta ADC
- Add-on options (not part of evaluation kit):
  - SDK with GCC and simple downloader
  - High speed UDP streaming support (RTP)
  - Hardware debugging (integrated JTAG ICE)
  - Safety watchdog functionality for critical applications

ht

| Designator | Description |
|---|---|
| J1, J2 | Two expansion headers. For the pinout, please consult netpp node HW reference |
| SW1, SW2 | Push button switches (Reboot/Reset, User button) |
| D1..D4 | LEDs (green, red, blue, yellow) |
| X1 | USB B mini connector (to PC) |
| X2 | 100MBit RJ45 connector |

Table 1.2: netpp node user I/O

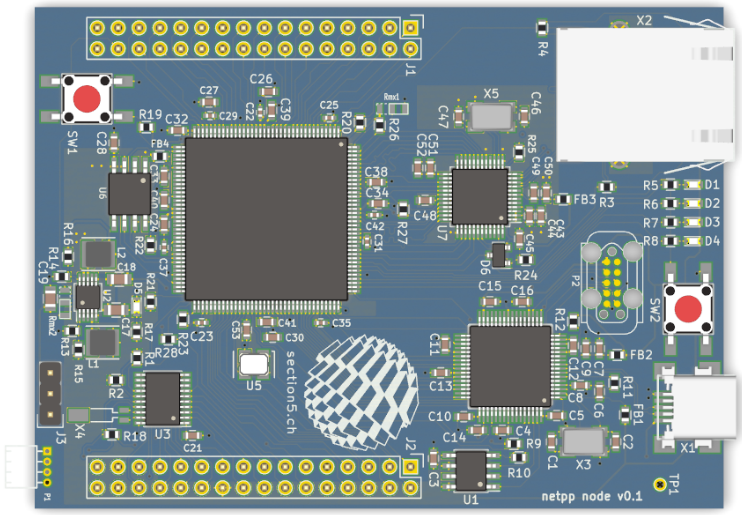

Figure 1.1: netpp node

### 1.4.2 Papilio One

The Papilio platform is a Xilinx Spartan3 based development platform with a USB JTAG solution on board. This board has gained some popularity similar to the Arduino boards. It can be extended using pluggable *Wings*, developed and supported mostly by a large independent developer community. Figure 1.2 shows a Papilio with a TFT wing.

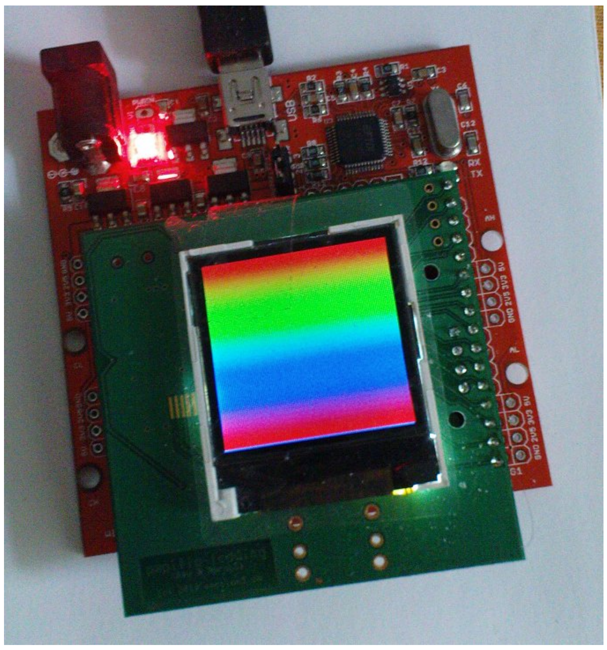

Figure 1.2: Papilio with TFT wing

The Papilio can be purchased from various online shops, see http://papilio.cc for more information.

### 1.4.3 Virtual silicon

This is simply a virtual board and nonexistent in hardware. It is meant to be a playground to experiment with.

# Chapter 2

# Quickstart

## 2.1 Docker container

You need a development environment which is typically supplied as a docker container. In this case you do not need to worry about the prerequisites, because everything has been installed "ready to go".

The contrib/docker directory contains a Dockerfile that installs a work environment from scratch. Tar-File-Distributions are no longer supported.

### 2.1.1 Building the 'masocist' container

First, build the `masocist_sfx.sh` file by running **make dist** inside `contrib/docker`. Then, build the docker container using:

**docker build -t masocist .**

Run it with mounted local directory:

**docker run -it masocist -v$LOCAL_MASOCIST:/usr/local/src/masocist**

Then, run the self extracting archive from inside the docker container -- make sure you are in the home directory:

**sh /usr/local/src/masocist/contrib/docker/masocist_sfx.sh**

This will unpack a Makefile plus the recipe folder in the current directory (which should be the home directory).

Then build everything by using:

**make all**

This will pull all necessary sources and build them, therefore it can take a while. Upon success, it may directly jump into a test bench, depending on the distribution you have.

### 2.1.2 Advanced Docker usage

If you wish to use some extended functionality, such as the gtkwave display, you'll have to run the docker container with some extended privileges. For this case, it might help to create a script as follows:

```
#!/bin/bash
docker run -ti -e DISPLAY=:0 --mac-address=00:de:ad:be:ef:aa \
        -p 7208:7208/udp -p 2008:2008/tcp -p 2008:2008/udp \
        -u masocist -w /home/masocist \
        -v/dev/bus/usb/:/dev/bus/usb/ -v/tmp/.X11-unix/:/tmp/.X11-unix \
        -v/home/me/src/masocist:/usr/local/src/masocist --privileged --ipc host masocist
```

It may also be necessary to install additional packages. This works using the console command:

**sudo apt install <package>**

A few extra packages that might be required:

**netpp-dev** Netpp development package for remote controlled simulation

**zpu-toolchain** Compiler support for ZPUng and Zealot cores

**gtkwave** GTKwave wave display (see also Section 2.3.1)

**minicom** Terminal program to connect to virtual UART

## 2.2 Quick board selection

The supported board supply packages for your distribution are shown by the top level (i.e. in the masocist source directory) command:

**make which**

To configure for one of these boards, run

**make <boardname>**

Then you are ready to (re)build the source code and HDL files.:

**make all**

For available custom or third party configurations, see also `vendor/$(VENDOR)/defconfig_*` or check your custom SoC documentation.

## 2.3 Simulation

The entire SoC is simulated by GHDL with a few extensions to support virtual interfaces. Not all boards can be simulated by default, see below.

To build the simulation, GHDL and the GHDLex library must be present. Then you execute

**make sim**

in the MaSoCist directory. Possibly you will have to call **make -C sim clean** after a reconfiguration.

### 2.3.1 Running the simulation

Once the simulation was built correctly, a `tb_<platform>` or `net_<platform>` executable will be created in the `sim/` directory, depending on the configuration. Some configurations are based on a virtual UART instance, see `sim/virtualuart.vhdl`. This is started using the `init-pty.sh` script. Then you can connect to the running simulation using a terminal program, such as minicom:

```
minicom -o -D /tmp/virtualcom
```

**Note**
When running inside a container (Docker), no manual setup using init-pty.sh is required. The virtual COM port is set up by the Docker container.

Note that this UART is a full UART emulation, so the baud rate defined by the divider value passed to the VirtualUART must match the UART peripherals baud rate, unlike the virtual console (`CONFIG_VIRTUAL_CONSOLE`), which does not depend on a baud rate.

Finally, the simulation is run by starting the tb_<platform> executable from the command line. For interactive waveform tracing using GTKwave, there is a script `run.sh`, taking the platform name as argument, e.g.

```
./run.sh papilio
```

### 2.3.2 Advanced interactive simulation

If an executable has the 'net_' prefix, it integrates a netpp daemon to listen to external accesses. This allows interactive stimulation of the test bench or data transfer with user applications.

Enter the sim/ directory and type "make run". You should see the GTKwave window popping up, showing a slowly progressing wave display due to the enabled throttle. To disable the throttle, run the following command inside the virtual machine (or on your host, when running the default Docker container with networking as in Section 2.1.2):

**netpp localhost SimThrottle 0**

If you run netpp from your host to access the Virtual Machine or LXC (Linux Container), you need to determine the IP address of its virtual bridge interface (or what you have configured your VM with). For example, **ifconfig** on Linux may display a VirtualBox specific network interface:

```
vboxnet0  Link encap:Ethernet  HWaddr 0a:00:27:00:00:00
          inet addr:192.168.56.1  Bcast:192.168.56.255  Mask:255.255.255.0
```

Then you simply access the simulation via

**netpp 192.168.56.1 SimThrottle 0**

**Note**
This option only applies when the simulation is configured with a virtual TAP, like `CONFIG_VTAP`

### 2.3.3 Vendor specific simulation issues

Simulation of some boards may require libraries that are not included in the MaSoCist, because they are vendor specific. There may be several solutions:

- Obtain necessary files from your local FPGA tool installation and create a GHDL library. Use the -P option to GHDL to specify the search path to the GHDL config file.
- Try the `CONFIG_EMULATE_PLATFORM_IP` option
- Use a virtual board config that is fully vendor IP independent

For example, when you receive an error message like this:

```
../hdl/plat/breakout_top.vhdl:16:9:cannot find resource library "machxo2"
```

You have to create a file lattice/machxo2-obj93.cf somewhere using the rule:

```
MACHXO2_VHDL = $(wildcard $(LATTICE_SIM)/machxo2/src/*.vhd)
lattice/machxo2-obj93.cf: $(MACHXO2_VHDL)
    [ -e lattice ] || mkdir lattice
    ghdl -i --workdir=lattice --work=machxo2 $(MACHXO2_VHDL)
```

where LATTICE_SIM is the directory of your simulation VHDL files, like `/usr/local/diamond/3.1_x64/cae_library/simulation/vhdl/`

Then set the LIBGHDL variable in `vendor/default/local_config.mk` to the directory where you created `lattice/machxo2-obj93.cf`, like:

```
LIBGHDL = $(HOME)/src/vhdl/lib/ghdl
```

**Note**
For full 'model in the loop' simulation, it will be necessary to acquire an additional, non-free Simulation package by section5

## 2.4 Prepare synthesis

If you have configured something using the menuconfig utility, run either "make syn" from the top level or "make" inside the `syn/` directory to update your files. Then run your synthesis tool and open up the corresponding project file in `syn/<FPGA_VENDOR>/<PLATFORM>`, for example:

**netpp_node** syn/xilinx/netpp_node/v0.1/netpp_node.xise

If the project files are not present for your platform, see below on how to quickly import the files required for synthesis.

### 2.4.1 Porting to new platform

When synthesizing for a new FPGA platform, you will have to create a new project first. The MaSoCist environment helps you with importing all the necessary files by creating TCL scripts for the supported synthesis tools (Xilinx ISE 14.7, Lattice Diamond 3.9 at this time).

1. Create the new project in `syn/<fpga_vendor>/<platform_name>/<project_name>`
2. To import the project files, open a TCL shell inside your IDE and run
   ```
   source ../../proj_<platform>.tcl
   ```
3. Run Synthesis to check if all files referenced are imported

**Note**
FPGA platform specific IP cores may have to be manually imported into the project

## 2.5 Configure and build

For usage of the kconfig environment, see SoC specific documentation.

Basically, you execute the kconfig menu by running

**make menuconfig**

from the top level MaSoCist directory.

Before synthesis or simulation, it is required to rebuild the corresponding system files:

**make syn**

**Note**
In the evaluation version (no gensoc included), the hardware peripherals can not be configured. You can only choose among supplied board configurations in `CONFIG_SOCDESC`.

## 2.6 Target download

### 2.6.1 netpp node

First, you have to set the `XILINX_ISE_DIR` variable in order to assemble the full firmware image for the SPI flash, you have either options:

- Define `XILINX_ISE_DIR` in vendor/opensource/local_config.mk
- Export `XILINX_ISE_DIR` in .bashrc or alternative shell startup file

Example:

```
XILINX_ISE_DIR = /media/sandbox/Xilinx/14.7/ISE_DS
```

The Xilinx tools allow to recompile the software without the full ROM synthetisation. See `syn/xilinx/papilio/Makefile` rules:

**$(PLATFORM)_fw.bit:** Build rule to merge program data with existing firmware bit file

**download:** Downloads bare SRAM image into target

**flash:** Merges extended memory (cacheable) space into overlay SPI flash image and programs it into the target

For download to the target, the papilio-prog application is required. See Papilio homepage for details.

The full firmware download is run by **make flash**. If no SPI code cache and program overlay is used, you can run **make download** instead. Otherwise, you have to make sure that all addresses are in sync.

### 2.6.2 Other boards

All other boards such as proprietary / custom development can be programmed using an ICEbearPlus adapter, if the JTAG pins are accessible. Some boards may have embedded USB JTAG controllers. See Table 2.1 for details.

| Board name | Programming | JTAG adapter |
|---|---|---|
| netpp node, papilio | xc3sprog | Integrated USB JTAG |
| HDR60 | Diamond Programmer | Integrated USB JTAG |
| versa ECP5 eval kit | Diamond Programmer | Integrated USB JTAG |
| Digilent Spartan-3 Board | xc3sprog | ICEbearPlus |

Table 2.1: Programming method

The Impact Cableserver for ICEbear is no longer supported.

# Chapter 3

# Vendor customisations

## 3.1 Use cases

When you wish to use the MaSoCist build system for your own, possibly non-OpenSource designs, you may want to not clutter the publicly visible Makefiles with your vendor specific or private details. In this case, you would typically:

- Create a branch of the git repository
- Add your vendor specific files in the usual places
- Add your build rules in the *vendor specific* Makefiles

## 3.2 Files

These files are typically found in the `vendor/<YOUR_ORGANISATION>/` directory:

**Makefile** Makefile for HDL project only

**Kconfig** Your vendor specific configuration options

**config.mk** Configuration file for all extra Makefile rules, including Software and Hardware options. This file also holds information on the software modules you have acquired and installed.

**local_config.mk** Configuration of software or tool locations that are local to the machine. Optional.

**vhdlconfig.mk** Included by `$(TOPDIR)/hdl/vhdlconfig.mk`, holds extra config variable conversion options

The file usage is documented in detail below.

### 3.2.1 Makefile

Variables to know:

**GENSOC_MODULES-y** A list of module names defined in the SoC XML file

**CUSTOM_XMLCONF** A rule to convert a configuration variable to a VHDL configuration variable

**GENERATED_FILES-y** List of VHDL files that are generated, like by a python or .chdl to .vhdl rule

**PERIO-y** Specific peripheral modules that reside in hdl/perio

**SRCFILES-y** General source files that reside in `hdl/`

**PLAT_SRCFILES-y** Platform specific source VHDL files

#### 3.2.1.1 Examples

This will for example add the `lcd_core` design module in hdl/perio/lcd.vhdl to the project:

```
PERIO-$(CONFIG_LCDIO) += lcd.vhdl
```

This adds the lcd module to the generated SoC peripheral decoders:

```
GENSOC_MODULES-$(CONFIG_LCDIO)        += lcd
```

This adds a new CONFIG_TEST_ABC configuration to the `global_config` VHDL package:

```
CUSTOM_XMLCONF += $(call xml_convert_boolean,CONFIG_TEST_ABC)
```

### 3.2.2 config.mk

Variables to know:

**DIST_PLATFORMS** These platforms are to be distributed and tested. If you have a working platform, add it to this list.

**DISTFILES** The files to distribute for a ’make dist’ rule

**MODULE_<MODULE>** Module configuration, set to ’y’ if the corresponding module is installed

**GENSOC_VERSION** If `MODULE_GENSOC=y`, this describes the used GENSOC_VERSION. Do not alter, unless you are upgrading to a compatible version.

All other rules add to the project building procedures. For example:

#### 3.2.2.1 Adding a SoC configuration

When you wish to add a SoC specific configuration (bound to the `hdl/plat/<soc_edition>.xml`), use a statement like below:

```
ifeq ($(CONFIG_SOCDESC),"extra")
CHDL_FLAGS = -DHAVE_EXTRA
endif
```

In the corresponding CHDL module (see also Section 1.1.1) you can then switch using:

```
#ifdef HAVE_EXTRA
      extra_vhdl_assignment <= extra_value;
#endif
```

#### 3.2.2.2 Adding a platform configuration

Example:

```
ifdef CONFIG_my_platform
HAVE_PLATFORM_DESCRIPTION = yes # Use if a hdl/plat/plat_my_platform.xml exists
REQUIRE_PREREAD = y             # If we require a common preread wrapper
CHDL_FLAGS += -DHAVE_MAC
FPGA_VENDOR = lattice
FPGA_ARCH = ECP5
endif
```

#### 3.2.2.3 Adding custom C source objects

```
OBJS-$(CONFIG_MY_VENDOR_STUFF) += my_vendor/stuff.o
```

Note that when you are building against any of the public opensource source, you have to publish your code as well. If it is of general use, you might rather add it to the `sw/Makefile`.

# Chapter 4

# Troubleshooting

Typical errors that may appear on your console:

**/usr/bin/ld: cannot find -lslave** You need to install the `netpp-dev` package or the full netpp source and set the `NETPP` environment variable to it. Possibly you might have to set `LD_LIBRARY_PATH`, if you have a custom installation.

**virtualuart.vhdl:42:24:@0ms:(assertion failure): Failed to open PTY pipe** The virtual UART is not running. Start `sim/init-pty.sh` first, see Section 2.3.1.

**../hdl/plat/breakout_top.vhdl:16:9: cannot find resource library "machxo2"** Vendor specific library not installed, see Section 2.3.3.