Tai Lung RV32I Project

Tools Flows & methodologies (TFM)

Authors:

Christian Shakkour, chrisshakkour@gmail.com

Shahar Dror, shdrth1@gmail.com

Git repository:

https://github.com/ChrisShakkour/RV32I-MAF-project

Release Information

DATE	AUTHOR	DESCRIPTION
26-NOV-2021	Chris	created
02-JAN-2022	Shahar	Added table of contents and relevant section
31-JAN-2022	Chris	Content creation

Acknowledgements

Special thanks to our supervisor Amihai Ben David for contributing knowledge and experience throughout the development of this project. Many thanks to the VLSI Lab at the Technion - Israel Institute of technology for supporting and providing tools and resources that were an integral part of the development process.

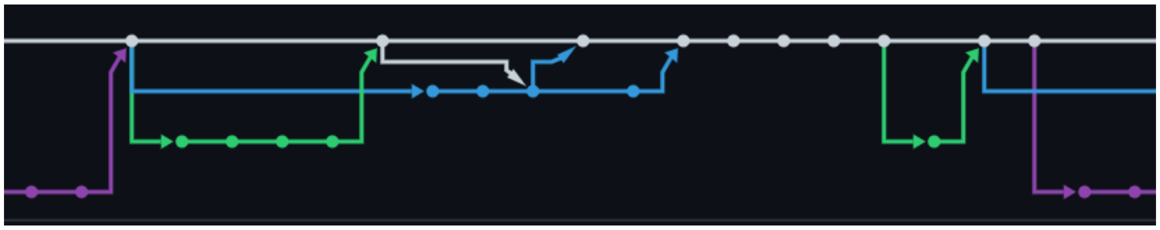
About this document

In this document you will find information regarding the development environment, the tools, and other useful information to get you started. the development of this project has been done on a Linux based OS, all the tools, scripts, files are meant to run on linux and not windows so before you get started make sure you have a working Linux based OS with basic prerequisites of a Linux terminal with Bash and Python3 installed. following along this document you will have to download a few more tools if you intend to use our workflow or migrate this project into your own Environment.

Table of Contents

Workflow & Version Control

Git is used for version control, and it is done using the GitHub application.



GitHub

GitHub provides many great features that has been used seamlessly in the development process, parallel development using branches, task management using issues, Discussions, and much more. following workflow on this repository.

Workflow

1. Clone repository.

   git clone <repo url>
2. Create a new branch.

   git branch <branch name>
3. Checkout into created branch.

   git checkout <branch name>
4. Develop.
5. Add desired changes.

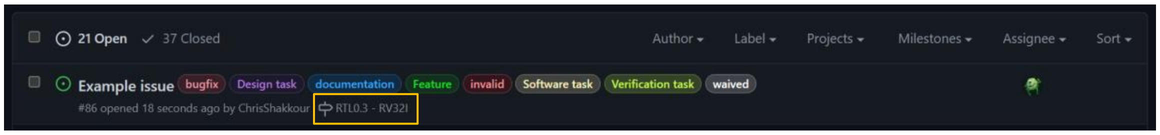
   git add <desired files>
6. git commit desired changes.

   git commit -m "<commit message>"
7. Create a pull request.

   git pull
8. Design review.
9. Merge branch

Task management

Task management can be done using many different applications. In this project GitHub issues tab was used to manage and monitor the tasks.



Notice the colorful labels, they help classify the task's objective, on the right the assignee can be seen. A nice feature is milestone creation that can be linked to tasks see in small yellow box.

Repository structure

The best practice to get to know the structure is by exploring the git repo, but here are some general information about the different directories, following first level tree.

```
                                Project repo

                                     |

    ---------------------------------------------------------------------------------------------------

    |           |        |        |          |          |          |          |

Source_me     FPGA      HDL     vfiles     verif     scripts      doc      OUTPUT
```

Source script

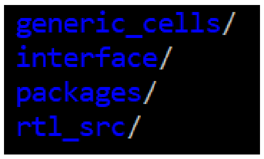
The source script is a bash script that needs to be sourced before running anything on the repo this script sets environment variables, aliases and creates the OUTPUT directory for later use.

FPGA

This directory occupies all FPGA related files as well as the compiled bitstream, currently empty, to be added in the next project.

HDL

The HDL directory has all the synthesizable rtl source Code of the RISC-V Core this directory will be released for future tape out activities (Backend, production), inside the HDL directory.



```
generic_cells/
interface/
packages/
rtl_src/
```

The names of the directories tell their purpose however the rtl_src has the Core and all the sub hierarchies in the design.

vfiles

when compiling the design, we will use a dedicated vfile to direct the compilation tool to all desired units needed in this unit, this methodology is used widely when working on multi hierarchical designs with lots of design modules, almost every design unit has a vfile.

verif

the verification directory holds verification-based RTL code like a testbench, tracker modules, checkers, and software C code for running simulations.

Scripts

here you will find various scripts, some like Linker and CRT0 scripts for simulations, and other scripts like python scripts that are used for automating different flows.

doc

documentation repository occupies RISC-V spec documents, project documents, How-to documents, HAS/MAS and other useful documentation on this project.

OUTPUT

Tis directory will be generated when sourcing the source_me script and is not tracked by git, the purpose of this directory is to hold all the logs and output files generated from compilation, simulation and other processes like debugging that doesn't need to be saved in the directory.

Tools

In this section you will find the tools used with a sufficient explanation to get up and running. **Modelsim** is used for compiling and simulating the RTL. **RISC-V gcc toolchain** is used for compiling and linking c programs into executable format to run on the Tai Lung Core, in this project we implemented a python wrapper scripts that run the native tool commands for automation purposes these scripts are python based and can be found under $GIT_ROOT/scripts/tools/ , from this stage on we assume you have installed modelsim and risc-v gcc toolchain if not then do so by checking out the references for quick online links.

Before you start using the following scripts make sure you sourced the source_me script.

```
>> source source_me.sh
```

compile_hdl

compile_hdl is the script used for compiling RTL modules, design and testbench modules, following available arguments and example usecase.

argument	description	Example
h	Display help	-h
top	Design module name	-top CoreTop
cmd	Generate cmd only, don’t execute	-cmd

following command line example for compiling the main testbench.

>> compile_hdl **-top** CoreTopTB

What can be put in the **top** argument?

>> compile_hdl **-top** <name>

Go to $GIT_ROOT/vfiles/*/* the script will add <.vfile> extension to the name provided in the top argument like so name.vfile and search the vfile directory for any match, execute the following to see the available modules that can be compiled using this script.

```
>> basename -a $GIT_ROOT/vfiles/*/*.vfile
```

*basename is a unix command!

To add your own modules create a vfile for them first and run the script, see other vfiles for reference.

compile_gcc

compile_gcc is wrapper script for the native gcc compiler for risc-v, it use used to compile C and assemble programs, link executables, objdump, and objcopy following arguments and example use case.

argument	description	default	Example
h	Display help		-h
src	Source file to compile .c or .s extension		-src TowerOfHanoi
arch	Architecture flavor {rv32[I, m, a, f, ….]}	rv32i	-arch rv32imf
abi	Aplication binary interface {ilp32f, ilp32}	ilp32	-abi ilp32f
elf	Generate executable and linkable format		-elf
txt	Generate human readable text flag		-txt
mem	Generate mem image flag		-mem
od	Output directory	Same as src path	-od <somepath>
name	Output directory name		-name prog_files
linker	Linker script	$LINKER	-cmd <linker path>

following command line example for compiling TowerOfHanoi C program.

>> compile_gcc **-src** $GIT_ROOT/verif/tests/TowerOfHanoi/TowerOfHanoi.c **-elf -txt -mem**

This command will generate a few output files under the same path of the source path because -od argument has not been asserted, the -elf argument will generate an ELF file, the -txt argument will dump the contents of the elf file in a human readable text file, the -mem argument will enable the memory image generation, one file for the instruction memory the other is for the data memory, notice the compiler used the default values for the architecture and the ABI type because we didn't specify the arch and abi arguments, same goes for the linker.

What can be in the src argument?

>> compile_hdl -src <path>

the -src argument takes in a full path to a C or assemble source codes.

simulate

simulate is a wrapper script for modelsim vsim.exe command, it is used to simulate RTL and produce waveforms, this command(simulate) simulates the main CoreTop testbench. following arguments and example use case. *notice to be able to simulate you need to compile your c code first using compile_gcc command above and then run the simulate.

argument	description	default	Example
h	Display help		-h
top	Top module name	CoreTopTB	-top CoreTopTB
cmd	Only generate run command no execution	False	-cmd
gui	Enable gui	False	-gui
waves	Create a waveform (only used in gui mode)	False	-waves
test	C test to execute on the Core simulation	None	-test TowerOfHanoi

following command line example for simulating TowerOfHanoi c test on the cpu. Given TowerOfHanoi c test was compiled using the compile_gcc command, run the following command.

```
>> simulate -test TowerOfHanoi -gui -waves
```

This command will search for the executable memory extension files (mem.I, mem.D) under verif/tests/TowerOfHanoi/TowerOfHanoi.mem.(I/D), if found the test will start running. in this comman -gui and -waves are asserted hence a gui and waveform will pop up and the test should run successfully. At the end you shall see new files created in the OUTPUT directory where simulation data has been dumped, like the simulation command, tracker files data, and memory image data.

RTL Design Methodologies

This section explains the methodologies and coding style used in the design process:

- clk : common clock signal for all modules.
- rstn : common **async reset** signal for all modules **active low**.
- `timescale 1ns/1ns in all sv blocks, Design and tb.
- Module names are CamelCase Ex. InstructionFetch.sv
- Signal names are all lowercase! With underscore Ex. data_in
- Latch free design!!!
- Prevent hardcoded numbers.
- No defines or macros, only parameters and packages.
- Parametric design.
- Parameters are In UPPERCASE only!
- Instantiation in design: core #() Core_inst ();
- Instantiation in testbench: core #() DUT_Core ();
- Testbench always with $finish; at the end.

References

Modelsim

Modelsim is a multi-language environment designed by Mentor Graphics, for simulation of Hardware description language(HDL) such as VHDL, Verilog, and SystemVerilog.

Link: https://eda.sw.siemens.com/en-US/ic/modelsim/

RISC-V gcc toolchain

RISC-V gcc toolchain is a C and C++ command based cross-compiler for this risv-v eco system, it supports two build modes, a generic ELF/Newlib toolchain and a more sophisticated Linux-ELF/glibc toolchain.

Repo link: https://github.com/riscv-collab/riscv-gnu-toolchain

Intro link: https://www.youtube.com/watch?v=XSyH9T-Cj4w&ab_channel=RISC-VInternational