# How to Contribute

Patches are welcome. This is a broad project with many components, both

software and hardware. Here are some examples of areas to help out, which are

by no means exhaustive:

1. Hardware - Improve performance, increase clock speed, reduce area, add new

instructions or fixed function blocks. Synthesize for other FPGAs or ASICs

(fix errors with other tools, add build scripts and config files)

2. Verification - Create new tests and test frameworks, improve existing ones.

3. Compiler - Improve code generation, port other language frontends

(especially parallel languages).

4. Tools - Improve and implement new profiling, visualization, and performance

measurement tools.

5. Benchmarks - A variety of benchmarks help evaluate instruction set or

microarchitectural tradeoffs. There are many libraries of parallel benchmarks

that could be ported.

6. Software - Optimize or add capabilities to librender, implement a raytracer,

port games or demo effects (which do double duty as a tests and benchmarks)

The [issues](https://github.com/jbush001/NyuziProcessor/issues) section has list of

tasks and known bugs.

## Submitting Changes

Larger architectural changes or features should be proposed on the

[Mailing List](https://groups.google.com/forum/#!forum/nyuzi-processor-dev)

There are a number of [good pages](https://help.github.com/) on how to use

Github's standard pull request workflow.

The compiler is a submodule under the tools directory, but nothing in the project

directly references anything in that directory, only stuff that has been installed

in /usr/local/... (using `make install`) To make changes to the compiler, the

easiest thing to do is probably to fork <https://github.com/jbush001/NyuziToolchain>

and clone it into another directory.

## Testing Changes

When adding new features, add tests as necessary to the tests/ directory. The

'make test' target will run most tests and automatically report the results,

but here are a few other tests to run manually:

1. Create random cosimulation tests - Randomized tests aren't checked into the

tree, but it's easy to create a bunch and run them. From tests/cosimulation:

```bash

$ ./generate\_random.py -m 25

generating random0000.s

generating random0001.s

...

$ ./runtest.sh random\*.s

Building random0000.s

Random seed is 1411615265

496347 total instructions executed

PASS

```

2. Synthesize for FPGA - The Quartus synthesis tools catch different types of

errors than Verilator. It will also print some basic information about the

synthesized design after synthesis:

Fmax 54.3 MHz

73,034 Logic elmements

Ensure the frequency hasn't decreased too much (the design will not work on FPGA

if it is below 50 MHz), and that the number of logic elements hasn't increased

disproportionately.

3. For compiler and emulator changes, compile and execute run apps in software/apps.

## Coding Style

When in doubt, be consistent with existing code. Coding style adheres to that

used by auto-formatting utilities, which I would recommend running on code before

submitting. To install:

sudo apt-get install astyle

pip install --upgrade autopep8

To reformat C/C++ code:

astyle --style=allman --recursive \*.cpp \*.c \*.h

To reformat Python code:

autopep8 --in-place -r

Python scripts use Python 3.

Additional coding conventions are found

[here](https://github.com/jbush001/NyuziProcessor/wiki/HDL-Conventions).