CPR E 381 Toolflow Manual

February 2022

1 Overview

The enclosed toolflow implements a testing and synthesis workflow using Modelsim (Questasim) and Quartus Prime. Both the testing and synthesis portions are necessary to ensure proper completion of all phases of the CPR E 381 Course project - a MIPS processor.

This tool is divided into 2 tools: 'test' and 'synth'. The test tool optionally compiles your processor, simulates 1 or more assembly programs on MARS, initializes a Modelsim simulation with your processor, loading in a test program, and then compares the simulation results of your processor with MARS.

The synth tool synthesizes your processor for an Altera FPGA, and then outputs a timing summary that should indicate your critical path.

2 Usage

381_tf.sh: This runner provides some environment setup and actually calls the test scripts. It may be run like

```
./381_tf.sh <new|test|synth|submit> [options]
```

2.1 Running 381_tf.sh new

The first mode for running the tool flow, new, initializes a new project in the directory "proj". This directory contains several sub-directories.

- mips
  - This directory will contain all of your requires mips source code for the project. Several sample programs are provided.
  - Add additional MIPS test files here as needed.
- src
  - This directory contains your processor source code. You may structure the directory however you wish, however, the base structure we provide is as follows.
    - MIPS_types.vhd
      - This file is guaranteed to be compiled first by the toolflow
      - Place all type declarations and global constants in here
      - You may include this file in any other source
    - TopLevel/MIPS_Processor.vhd

        - This file contains the skeleton processor. Be mindful of all of the required signal names.
    - TopLevel/mem.vhd
        - This contains a complete memory entity. This is the same module tested in Lab 2

- test
    - Place your VHDL test benches in here.
    - Required for submission

It is recommended that your team use some sort of version control. If you elect to use git, initialize a git repository inside of the proj directory.

```
cd proj
git init

cat << EOF | tee README
CPR E 381 Project 1
Teamate 1 Name <email@iastate.edu>
Teamate 2 Name <email@iastate.edu>
Teamate 3 Name <email@iastate.edu>
EOF
```

2.2 Running 381_tf.sh test [options]

This command is what you will use the majority of the time to test your processor's integration functionality. The command will compile your code, assemble one or more MIPS programs, simulate the MIPS programs on both MARS and your processor, and then compare the output. This framework is very similar to the one that will be used to grade the correctness of your processor.

- `--help` Prints a help message displaying these options
- `--asm-file <file>` Run an assembly file (.s)
- `--asm-dir <file>` Use instead of above; this runs a whole directory of .s files
- `--run-file <file>` Run all files assembly files listed in file
- `--max-mismatches #` Allow # mismatches (default 30)
- `--nocompile` Don't recompile VHDL (default compiles)
- `--sim-timeout #` Number of seconds to simulate for (Default 30)
- `--output-all` When running multiple assembly files via asm-dir or run-file, save the output from all runs, not just failures
- `--config CONFIG` Selects an alternate system configuration

```
./381_tf.sh test --asm-file proj/mips/addiseq.s
```

Example 1: Running 1 file

```
./381_tf.sh test --asm-file proj/mips/addiseq.s --nocompile
```

Example 2: Running 1 file, no compile

Examples 1 and 2 run a single file, proj/mips/addiseq.s. The results of this run will be stored in the temp directory, as well as the output directory if you failed the test.

In general, the temp directory contains all of the output, intermediate files, and logs of a single run. The toolflow throws these values away each run, so if you do many runs you will need to save the contents of the temp directory. This is handled automatically if you call the toolflow using either `--asm-dir <file>` or `--run-file <file>`, as these commands implicitly run the framework on many input assembly files.

```
./381_tf.sh test --asm-dir proj/mips/
```

Example 3: Running a directory

```
./381_tf.sh test --asm-file proj/mips/addiseq.s --output-all
```

Example 4: Running a directory, outputting all

Output from these "batch" runs are saved, by default, only if there is an error. A zip file is saved for each failure in the 'output' directory, and the filename of the assembly file that caused the failure is written to output/test_failures.txt. All of the output from a batch run may be saved with the `--output-all` option.

The test_failures.txt file generated for batch runs may also be fed back into the toolflow to rerun failures.

```
./381_tf.sh test --run-file output/test_failures.txt --
    output-all
```

Example 5: Running failures via a file

The above command can be run again and again, with each run only running the failures, then overwriting the test_failures.txt file with new failures.

2.3 Running 381_tf.sh synth

Synthesis can be run via -

```
./381_tf.sh synth
```

Example 6: Running Synthesis

Synthesis output, such as timings files, can be found in the temp directory. Be aware – synthesis may take several hours, particularly for Project 1.

2.4 Running 381_tf.sh submit

Lastly, configure your proj/ file structure to match the submission guidelines. This should look like:

```
proj
|---mips
|      |---file.s
|      \---file2.s
|---src
```

```
|      |---proc...
|---test
|      |---tb_1.vhd
|      \---tb_2.vhd
\---report.pdf
```

```
./381_tf.sh submit
```

Example 7 : Generating Project 1 submission

Your submission will be sanity checked, and a final zip and report pdf will be placed in the submissions directory. Please make sure to upload both the zip and the report pdf to Canvas.

For project 2, add the option 'hw' or 'sw' to create a submission for either the hardware scheduled or software scheduled pipeline. Ultimately, you should create 2 zip folders, and submit both.

```
./381_tf.sh submit sw
```

Example 8 : Generating Project 2 Software Pipeline submission

```
./381_tf.sh submit hw
```

Example 9 : Generating Project 2 Hardware Pipeline submission

A Git

A.1 Motivation

While students are free to share code amongst their team however they please, one common solution used in many development projects is Git. If you are not aware of the fundamentals of Git, there are plenty of online resources that teach the fundamentals.

A.2 Initializing your repository

First, navigate to git.ece.iastate.edu. Click "New Project", followed by "Create Blank Project". Name your project, set visibility to "private", and **Deselect "Initialize with Readme"**. After your project has been created, record the repository URL by clicking "clone" and copying the HTTPS URL.

Next, on the lab machine or VDI, open terminal and first configure Git locally if you have yet to do so using –

```
git config --global user.name "Your Name"
git config --global user.email "netID@iastate.edu"
```

Next, use the "CD" command to change directories into the the cpre381-toolflow directory. Follow the steps in Section **??** to create your project directory. Next, add your repository to Git by initializing a new repo in the proj directory. For example:

```
./381_tf.sh new
cd proj
git init
git add .
git commit -m "Initialize repo"
```

```
git remote add origin <Repository URL>
git push -u origin master
```

Lastly, your teammates may clone the repo into the toolflow directory as follows. Make sure to first navigate inside of the cpre381-toolflow directory using cd.

```
git clone <Repository URL> proj
```

The newly cloned processor can then be run like normal with the toolflow.

B Config Files

B.1 Motivation

To run the toolflow on a non-lab machine, you may need to use an alternate configuration. Configuration is handled via a file config.ini, placed in the root of the toolflow directory. A sample of this file may be found in the internal directory.

B.2 Format

The file consists of one or more configurations organized into sections. Sections are denoted using brackets, "[MySectionTitle]". Key value pairs are then inserted underneath the section block using the form "key = value". There are 3 user configurable parameters - "modelsim_paths", "quartus_paths", and "needs_license". In general, "needs_license" should always be set to false unless you are using a univerisity licensed version of quartus. The two path parameters each accept a list of quoted *nix style paths, separated by commas and enclosed in brackets. If multiple paths are provided, then each path will be checked in order until a valid option is found.

```
[Config]
modelsim_paths=["/path/to/modelsim/bin", "/path2/to/
    modelsim/bin"]
quartus_paths=["/path/to/quartus/bin"]
needs_license = false

[AnotherConfig]
modelsim_paths=["/anoother/path/to/modelsim/bin"]
quartus_paths=["/another/path/to/quartus/bin"]
needs_license = false
```

Example 10 : Example Configuration

B.3 Using the configuration

To use a non-default configuration, you must specify to both the test and synthesis frameworks which configuration you would like to use. The name of the configuration that is passed to the toolflow must match a section in the config.ini file.

```
./381_tf test --asm-file file.s --config Config
./381_tf test --asm-file file.s --config AnotherConfig

./381_tf synth --config Config
```

```
./381_tf synth --config AnotherConfig
```

Example 11 : Running with New Configurations