FRoC Testing Documentation

This document gives a general overview of FRoC's functionality tests. The tests can be broken into two distinct parts: compilation analysis and simulation analysis. Below, the different programs created to run each test will be described, along with their inputs and outputs and how they should be used.

1. Compilation Analysis
    1.1 Compilation Parser
        The compilation parser examines Quartus' compilation output for an FRoC project. It looks for errors, which suggest a failed compilation, and pieces of information having to do with routing.

        The Parser requires one argument to run:
        1. A path to a text file containing Quartus' compilation output

        With this input the parser will run. It will output any errors or other pertinent routing/placement information it finds. There are no output files.

    1.2 Placement Parser
        The Placement Parser Compares the meta parser placement output when run on the Quartus timing analysis of the circuits FRoC is testing to the meta parser placement output when run on the Quartus timing analysis of FRoC itself. It searches for paths with identical placement. It then compares the number of found matches to the number of paths that are supposed to be replicated by FRoC. If the former is less than the latter, an error is generated as one or more paths have not been identically placed.

        The Parser requires three arguments to run:
        1. The number of expected matches
        2. A path to the meta parser placement output from the original circuit
        3. A path to the meta parser placement output from the FRoC circuit

        With these three inputs the process with run and report any errors. If an error is found, it will exit with value 1. There are no output files.

    1.3 Routing Parser
        The Routing Parser Compares the meta parser routing output when run on the Quartus timing analysis of the circuits FRoC is testing to the meta parser routing output when run on the Quartus timing analysis of FRoC itself. It searches for paths with identical routing. It then compares the number of found matches to the number of paths that are supposed to be

replicated by FRoC. If the former is less than the latter, an error is generated as one or more paths have not been identically placed.

The Parser requires three arguments to run:

1. The number of expected matches
2. A path to the meta parser routing output from the original circuit
3. A path to the meta parser routing output from the FRoC circuit

With these three inputs the process with run and report any errors. If an error is found, it will exit with value 1. There are no output files.

2. Simulation Analysis

2.1 DUT Maker

The DUT Maker takes FRoC's top.v output file as input, and from it creates a wrapper around FRoC that acts as a device under test (DUT) used for simulation. This DUT connects to a bus that provides access to the internal signals that must be analyzed for design verification. The DUT.sv file can then be combined with other SystemVerilog files to provide a complete test bench for the FRoC circuit.

The DUT Maker requires one argument to run:

1. A path to FRoC's top file, containing module top (usually called top.v or top_i.v)

The DUT Maker creates four output files:

1. A DUT.sv file: this is the file that wraps around FRoC's top module to provide access to internal signals
2. A constants.sv file: this file provides constant values that are used by other pieces of the test bench
3. A constraints.txt file: this file is read by the simulation parser in order to determine which signals can't change simultaneously. In the file, each LUT is started by LUT, then on the following lines, identifiers for each LUT input are printed on new lines. Different LUTs are separated by empty lines.
4. A labels.txt file: this file records which identifiers (as used in the constraints.txt file) correspond to which signals (as found in FRoC's top.v file). This conversions can also be seen, though in a less neat way, in the DUT.sv file. This file can be used to interpret errors outputted by the simulation parser

The DUT Maker can be run successfully if the FRoC file is in the expected form. If there is an error, due to an issue with the form of top.v or another issue, the DUT Maker will fail silently. Following the running of the DUT Maker, a simulation project can be created by combining the FRoC project files with DUT.sv and constants.sv from the DUT Maker

and the ready made files, Controller.sv, Interface.sv, and test_top.sv. This project can then be run through the simulator with the script test_top.do.

2.2 Simulation Parser

The Simulation Parser analyzes the output file of the simulation and uses it to determine three things. First, that every intermediate value and every sink register switched from 0-1 and from 1-0 (in any order). Second, that in any case where a LUT has multiple inputs from non-control signals, only one of those signals changes value at a time. And third, that in simulation, the fail flag is never raised.

The Simulation Parser requires two input arguments to run:

1. A path to the simulation output file
2. A path to the constraints.txt file created by the DUT maker

With these inputs, the Simulation Parser can be run. It produces no outputs. If it finds any problems with the design, error messages will be generated. These messages may reference signal IDs, which can be connected to signal labels in the labels.txt file produced by the DUT Maker.

3. Testing Scripts

3.1 run_FRoC_tester.py

This tester is capable of running all the tests mentioned above in one test flow, it takes the following option arguments:

-h show the help menu
-o select a file to write the script's output to
-a run all the available tests
-s only run simulation tests
-p only run routing and placement tests
-b build FRoC before testing
-f select a timing file to test (giving a path)
-n select a timing file to test (by giving a file name in the timing files folder)

3.2 testing_functions.py

This contains several functions that are called by run_FRoC_tester.py. The functions perform the actual tests, usually using the shell to call the test executables and to move produced files around.

3.3 runFRoC

This script takes the output files of FRoC, moves them to a selected directory and renames them to suit the tester's purposes. It then builds a tcl script that can be run by Quartus to create and build a FRoC project with the proper details and constraints. It builds the tcl file around the content found in createFRoCProject.tcl

3.4 newFRoCProject

This script is called by the testing functions. It builds a prepared FRoC project using a tcl script produced by runFRoC

3.5 FRoCTiming

This script gets the timing information from Quartus for a FRoC project. It calls the quarts_sta executable, giving it the getTiming.tcl script found in the same folder, to get the timing information.