A - GITHUB REPOSITORY

The SystemVerilog code for CV32E40X core with 2-bit branch predictor unit and Python scripts used to run the simulation, synthesis, and extract and analyze data can be seen in the following GitHub Repository:

Github repository link

- `https://github.com/Giorgi-Solo/MasterThesis/tree/master`

The repository's content has also been submitted to NTNU Insperra as a .zip file. When the users clone this directory, they **MUST** clone it **RECURSIVELY**.

The rest of Appendix A explains the repository structure and gives the curious readers guidelines about setting up the simulation/synthesis environment, running the branch predictor model developed for the specialization project, simulating/synthesizing core, and extracting and analyzing simulation/synthesis data.

In order to simulate, synthesize, automatic place and routing, and manual place and routing the user needs to have access to the following tools and technology library:

- Simulation - QuestaSim
- Synthesis - Cadence Genus
- Automatic Place-and-route++ - Cadence Innovus
- Manual Place-and-route - Cadence Virtuoso
- STM 28nm technology library

.1 Structure of Repository

MasterThesis repository consists of a README file, and two folders - Master and Workplace.

.1.1 Master

The folder contains another GitHub repository named CV32E40X (link to the repository - https://github.com/Giorgi-Solo/cv32e40x). The CV32E40X repository is forked from openHW group's repository (the baseline core - without branch predictor). I implemented the 2-bit branch predictor unit with 8 cachelines in the core from the forked repository. Figure .1.1 displays the fraction of the RTL module tree which implements a 2-bit branch predictor and misprediction recovery unit.

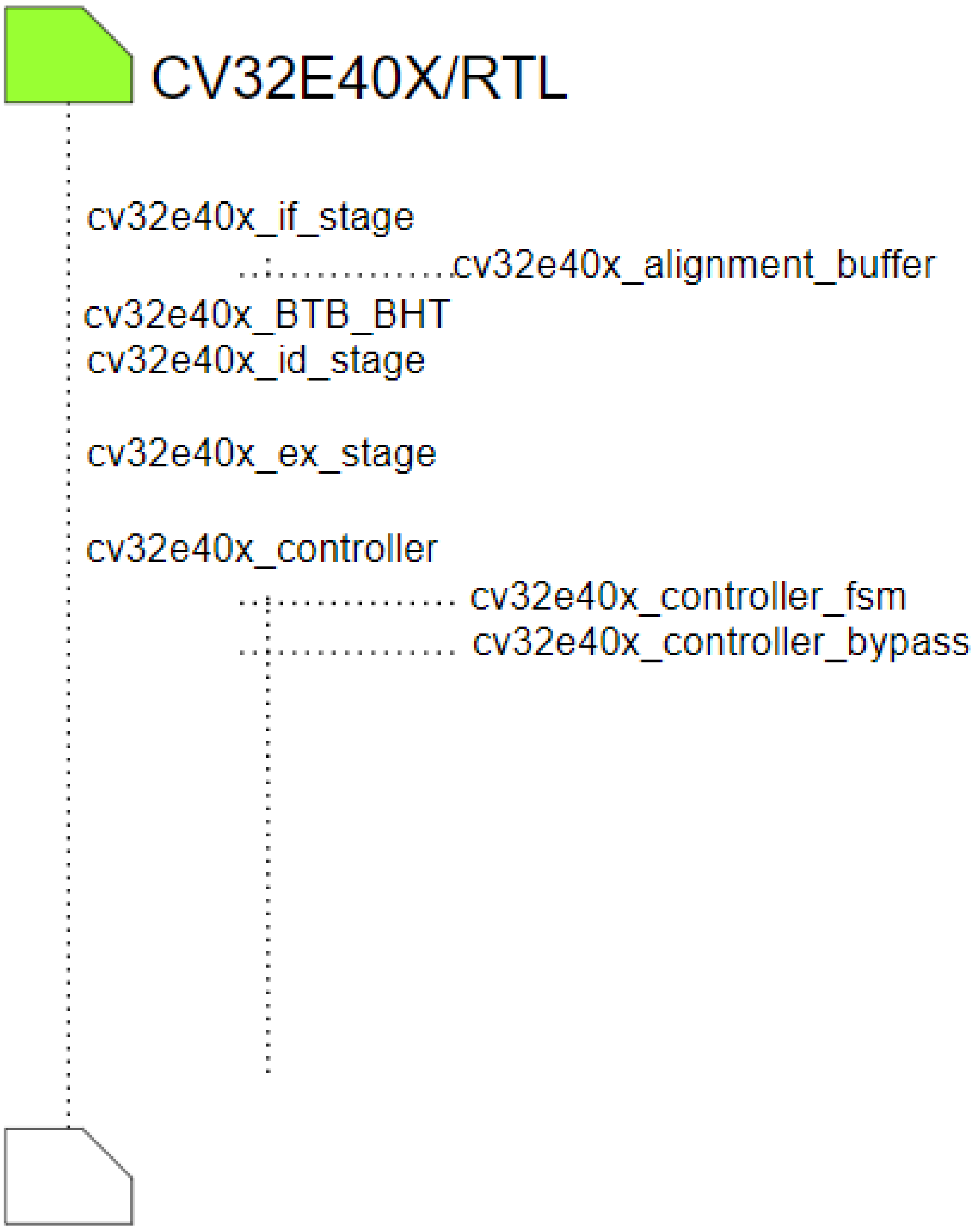


Figure .1.1: Code Tree of CV32E40X Core with 2-bit Branch Predictor

.1.2 Workplace

The folder contains the following Python scripts:

- logParser.py - Implements branch predictor model;
- sim.py - Simulates 128 core versions. The difference between each of the versions is the cache size. The script also extracts information about the maximum number of valid entries in the cache during the simulation, the number of encountered branch instructions, the number of predictions, and the number of correct and incorrect predictions (btb_statistics). Finally, the script runs the test program on each of the 128 versions of the core and extracts the UVM log file;
- sim_analyzer.py - Extracts information about the number of cycles required for each of the cores to finish the test program, clock cycle per instruction (CPI), number of valid entries in the cache, number of branch instructions, number of correct/incorrect predictions, and number of branch predictions. The script also calculates the CPI improvement, what fraction of the cache used by the predictor is, and what fraction of the predictions were correct/incorrect.
- sim_data_analyzer.py - The script analyzes simulation data extracted by sim_analyzer.py script and generates graphs used in chapter 4 (figures 4.1.2 - 4.1.5).
- synth.py - Synthesizes 128 core versions. The difference between each of the versions is the cache size. The script also extracts synthesis reports about area usage and power consumption.
- syn_analyzer.py - The script parses the synthesis reports and extracts information about cell area, net area, total area, total power, and power used by the registers. The script also calculates what fraction of total power is used by the registers
- syn_data_analyzer - The script analyzes synthesis data extracted by syn_analyzer.py and generates graphs used in chapter 4 (figures 4.2.1 - 4.2.2).
- show_log.py - The script displays information about errors and warnings encountered while running the previous six Python scripts

Workplace contains two folders:

- simulations - The folder stores all the simulation data extracted by sim.py and sim_analyzer.py. The folder stored all the synthesis report and synthesis data extracted by synth.py and syn_analyzer.py. The folder also contains a makefile used to simulate or synthesize the core.
- tcl - Contains .tcl codes that need to be added to .tcl scripts in the synthesis environment (explained later).

Finally, the workplace contains a makefile that is able to clone simulation and synthesis environments, run all the mentioned Python scripts, and erase generated logs.

.2 Running Model

In order to run the model, first, the user needs to clone the MasterThesis repository from the GitHub Repository Link, presented at the beginning of the appendix.

Next, the user needs to run a makefile target named "model" from the workplace directory. The target contains the following two commands:

- @echo "Running Model of the core th Predictor" - prints the message into the console.
- @python3 logParser.py - runs logParser.py script.

For more information about the output generated by logParser.py, refer to the specialization project report[1].

.3 Setting up Simulation, Simulating the Core, Analyzing Simulation Data

.3.1 Setting up Simulation

In order to simulate the core, the users need to clone the simulation/verification environment developed by openHW group. The simulation environment can be cloned from the following Github Repository - https://github.com/openhwgroup/core-v-verif. This repository contains a folder Core-V-Verif which represents the verification environment. More information about setting up the environment can be found in the article "CORE-V-VERIF Quick Start Guide" provided by openHW Group[20].

Core-V-Verif can be cloned by running a makefile target named clone_sim_env from the workplace directory. The target contains the following two commands:

- @echo "Simulation/Verification environment is being cloned from openhwgroup github public repository" - prints the message into the console.
- @cd ../ && git clone https://github.com/openhwgroup/core-v-verif.git - clones Core-V-Verif and ensures that Workplace, Master, and Core-V-Verif are at the same level in the directory tree.

.3.2 Simulating the Core and Analyze Simulation Data

In order to simulate the core, the users need to run a makefile target named sim from the workplace directory. The target contains the following five commands:

- @echo "Makefile from workplace is running simulations" - prints message to the console.
- @python3 sim.py - runs sim.py script, which extracts btb statistics and UVM log files.

- @echo "Running simulations analysis" - prints message to the core.
- @python3 sim_analyzer.py - runs sim_analyzer.py script, which analyzes simulation data.
- @cd ../master/cv32e40x/rtl/ && git stash && git stash drop - Running sim.py modifies cache size by changing the control variable - size, in cv32e40x_if_stage module. This command restores the cache size to 8 cachelines.

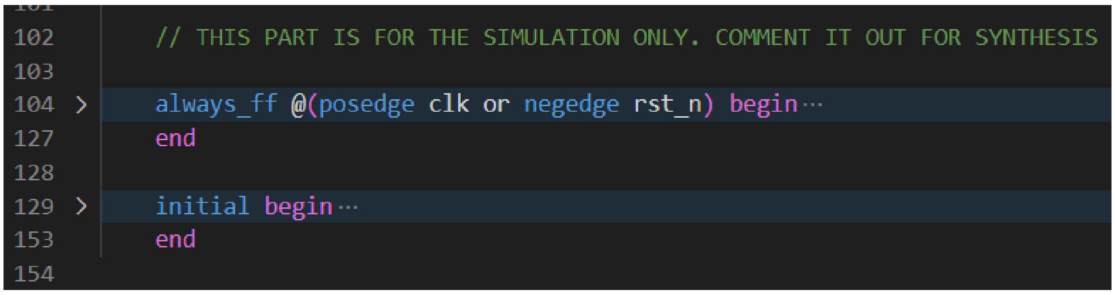
.3.3 Plot the Analyzed Simulation Data

In order to plot the analyzed simulation data (figures 4.1.2 - 4.1.5), the users need to run a makefile target named sim_data_analyzer from the workplace directory. The target contains the following two commands:

- @echo "Running simulation data analyzer" - prints message to the console.
- @python3 sim_data_analyzer.py - runs sim_data_analyzer.py script.

.4 Setting up Synthesis, Synthesizing Core, Analyzing Synthesis Data

Before we discuss synthesizing the design, the users must comment lines from cv32e40x_BTB_BHT module that store btb statistics in btb_statistics.txt file. Figure .4.1 displays SystemVreilog ALWAYS_FF (Line 104-127) AND INITIAL (Lines 129-153) blocks. These blocks need to be removed from the module before the user tries to synthesize the design.



```
102     // THIS PART IS FOR THE SIMULATION ONLY. COMMENT IT OUT FOR SYNTHESIS
103
104 >   always_ff @(posedge clk or negedge rst_n) begin···
127     end
128
129 >   initial begin···
153     end
154
```

Figure .4.1: cv32e40x_BTB_BHT module: Lines 104-153 store btb statistics into btb_statistics.txt file.

.4.1 Setting up Synthesis

In order to synthesize the core, the users need to clone the synthesis environment developed by EECS-NTNU. The synthesis environment can be cloned from the following Github Repository - https://github.com/EECS-NTNU/asic-flow. Note that this is a private repository, and the users will need permission to access it. This repository contains a folder asic-flow which contains tcl scripts for setting up the synthesis environment. More information about setting up the environment can be found in WiKi section of the asic-flow GitHub repository[21].

asic-flow can be cloned by running a makefile target named clone_synth_env from the workplace directory. The target contains the following two commands:

- @echo "Synthesis environment is being cloned from NTNU asic-flow github private Repository" - prints message to the console.
- @cd ../ && git clone https://github.com/EECS-NTNU/asic-flow.git - clones the repository and ensures that Workplace, Master, Core-V-Verif, and asic-flow are at the same level in the directory tree.

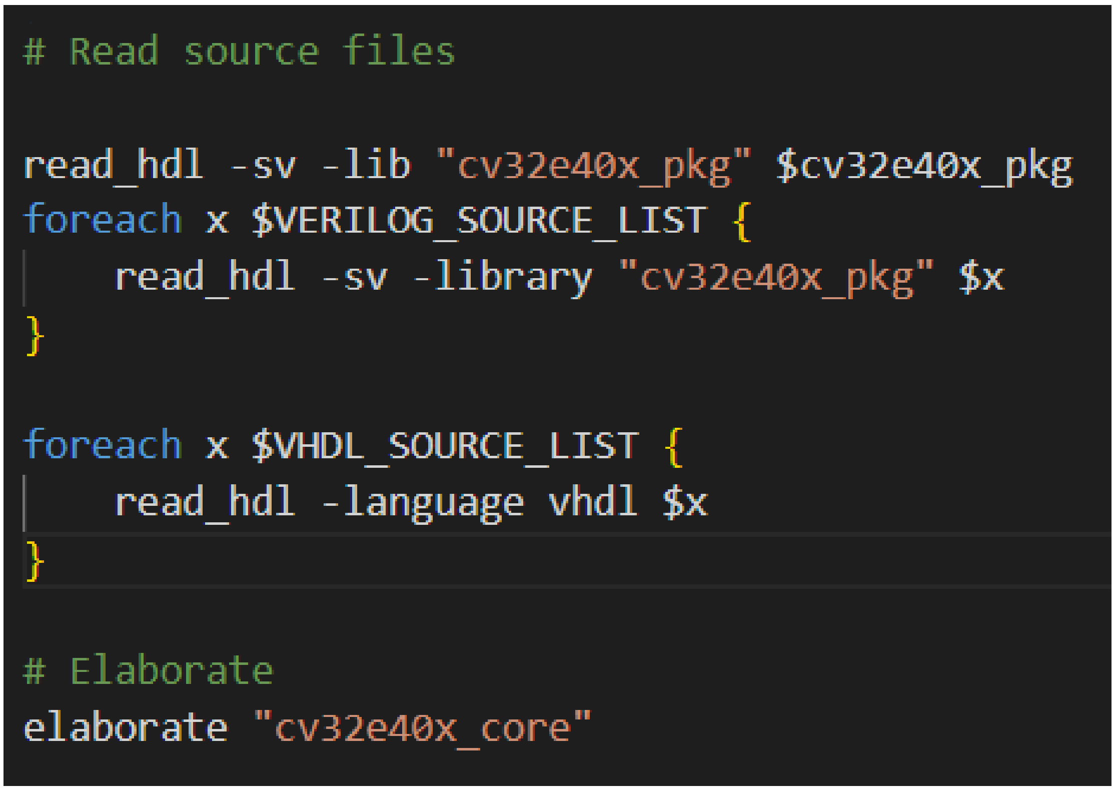
Before using the repository, we need to modify two files: asic-flow/stm28/counter/config_syn.tcl and asic-flow/stm28/counter/tcl /synth.tcl.

.4.1.1 Modificatoins to asic-flow/stm28/counter/tcl /synth.tcl

The user should delete line number 6 in asic-flow/stm28/counter/tcl/synth.tcl file and replace it with the content of MasterThesis/workplace/tcl /config_syn.tcl

.4.1.2 Modifications to asic-flow/stm28/counter/tcl /synth.tcl.

The user should replace lines 70-80 from asic-flow/stm28/counter/tcl /synth.tcl with lines from figure .4.2.



```
# Read source files

read_hdl -sv -lib "cv32e40x_pkg" $cv32e40x_pkg
foreach x $VERILOG_SOURCE_LIST {
    read_hdl -sv -library "cv32e40x_pkg" $x
}

foreach x $VHDL_SOURCE_LIST {
    read_hdl -language vhdl $x
}

# Elaborate
elaborate "cv32e40x_core"
```

Figure .4.2: TCL script reading core modules. These lines can be found in MasterThesis/workplace/tcl /synth.tcl file

Furthermore, the user needs to replace lines 95-97 from asic-flow/stm28/counter/tcl/synth.tcl with lines 21-84 from MasterThesis/workplace/tcl/synth.tcl

Finally, the user needs to add "quit" command at the end of asic-flow/stm28/counter/tcl/synth.tcl

.4.2 Synthesizing Core and Analyzing Synthesis Data

In order to synthesize the core, the users need to run a makefile target named syn from the workplace directory. The target contains the following five commands:

- @echo "Makefile from workplace is running synthesis" - prints message to the console.
- @python3 synth.py - runst synth.py script.
- @echo "Running synthesis analysis" - prints message to the console.
- @python3 syn_analyzer.py - runs syn_analyzer.py script.
- @cd ../master/cv32e40x/rtl/ && git stash && git stash drop - Running syn.py modifies cache size by changing the control variable - size, in cv32e40x_if_stage module. This command restores the cache size to 8 cachelines. Furthermore, cv32e40x_BTB_BHT module is also restored, and ALWAYS_FF and INITIAL blocs from figure .4.1 are uncommented.

.4.3 Plot the Analyzed Synthesis Data

In order to plot the analyzed synthesis data (figures 4.2.1 - 4.2.2), the users need to run a makefile target named syn_data_analyzer from the workplace directory. The target contains the following two commands:

- @echo "Running synthesis data analyzer" - prints the message to the console.
- @python3 syn_data_analyzer.py - runs syn_data_analyzer.py script