This directory includes:

* README.docx
* C source code for simulation
* \amoebasat\_centralized: synthesis source code that uses centralized L update
* \amoebasat\_localized: synthesis source code that uses localized L update
* \amoebasat\_multi\_epsilon
* \benchmarks
* Some preprocessing source codes

To access Vivado 2020.1 on keter: source /opt/Xilinx/Vivado/2020.1/settings64.sh

1. Checking the input cnf file

Before using this solver, be sure to check if the cnf file has the correct format of 3-SAT:

* + Head of the file: p cnf vars# clauses#
  + Each clause is written on a single line, each line contains uniformly 4 numbers (3 literals and ‘0’ ending)



To convert k-SAT into 3-SAT, run: ruby conv3sat.rb “filename.cnf”

* generate a file “filename\_out.cnf”

the output vars# and clauses# are printed on the head of the file as “p cnf vars# clauses#”

To filled ‘0’ to each line, run: ruby pad0.rb “filename\_out.cnf”

* generate the valid cnf file: filename\_out2.cnf

1. Run the AmoebaSAT solver (C simulation)

C++ source code (written with dynamic memory allocation)

Compile C++ code: g++ amoebasat.cpp -o amoebasat

Solve: ./amoebasat “filename.cnf”

When solving SAT instances, the followings can be modified in the source code:

* + MAX\_N\_STEP: the max iterations#
  + MAX\_CONTRA: size of Contra; if input SAT instance is big that gives segmentation fault, try to increase this size
  + EPSILON: by default, it is set at 0.32 (converted into fixed point value as 0.32\*2^31 = 687194767)
  + N\_RUNS: by default, it is set to run 100 times for an instance

C source code (fixed memory allocation): old version, need manual modification & re-compilation

From the source code, edit the followings: N\_VARIABLE (vars#), N\_CLAUSE (clauses#), filename (cnf file path).

Compile: gcc amoebasat0.c -o amoebasat

Solve: ./amoebasat

Operations with Vivado HLS (better open the later versions)

1. Synthesize with Vivado HLS (amoebasat\_localized)

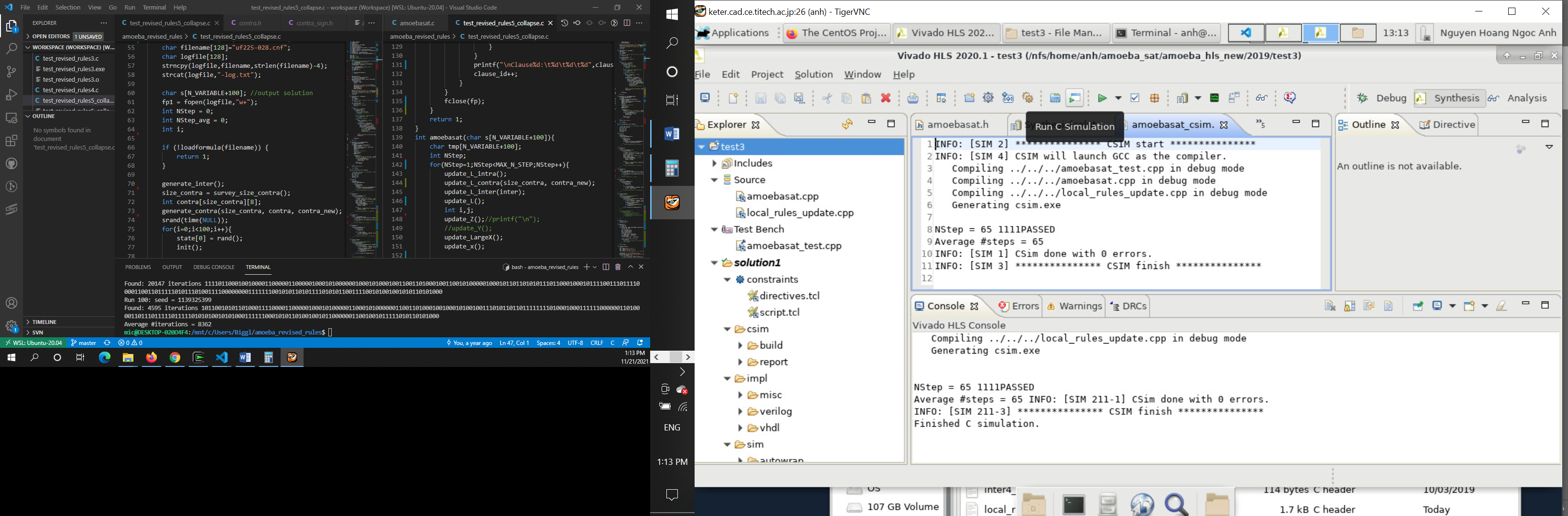
Go to directory: cd localized\_hls

* 1. Using GUI (recommend to check on small instances; if vars#>20, GUI synthesis may take very long time)
  + Preprocess:
    - Generate necessary files (local\_rules.h, local\_rules\_sign.h, local\_rules\_update.cpp, f.h, f\_sign.h):

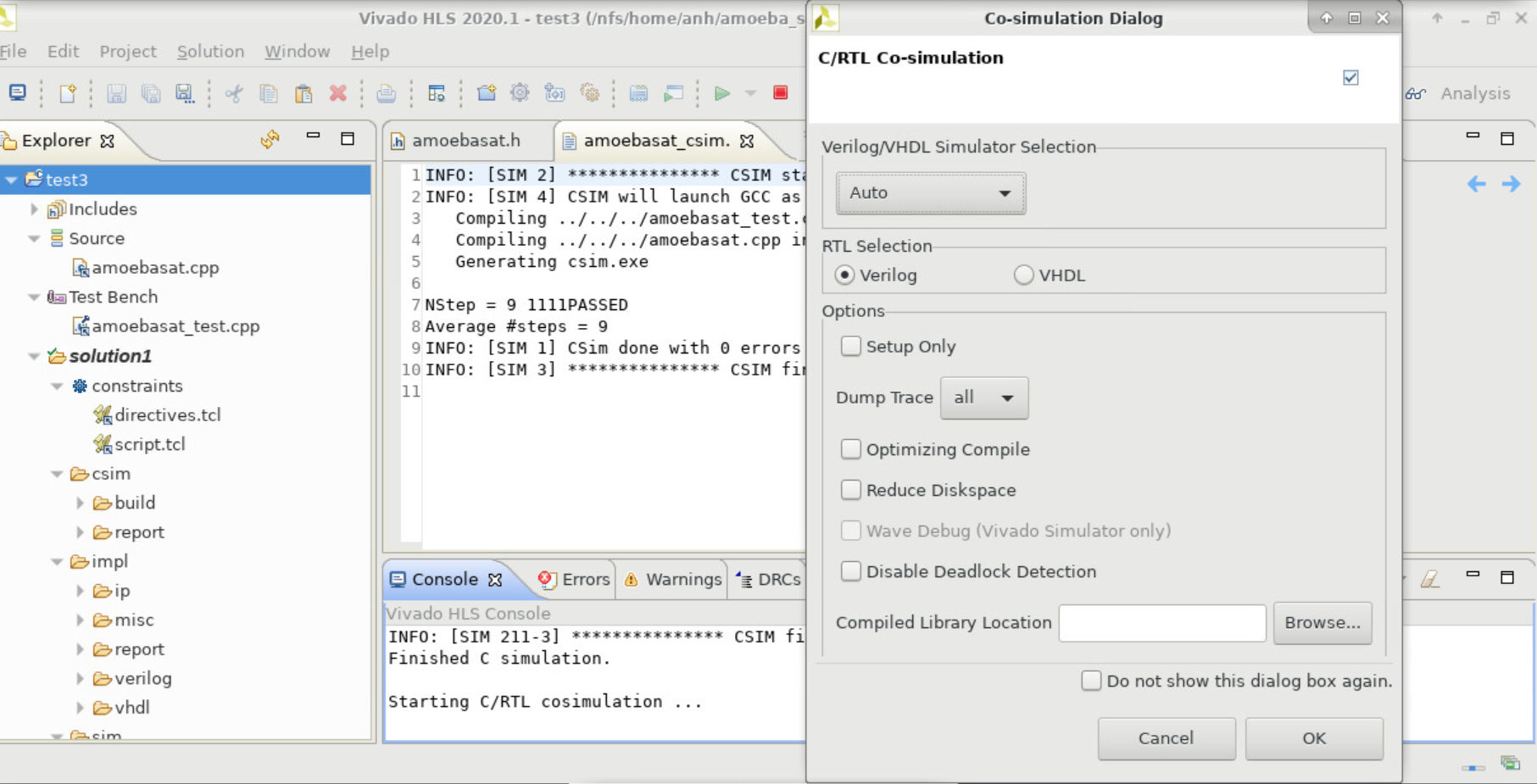
Run: g++ preprocess\_localized.cpp -o preprocess

./preprocess ../test.cnf

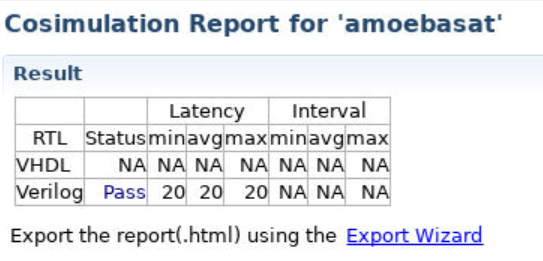
* + - Edit parameters in file “amoeabasat.h”: N\_VARIABLE, N\_CLAUSE, typedef ap\_ufixed<n,n> f\_t (n is the bitwidth needed to represent the literals; if N\_VARIABLE=100, n=7)
    - Create the HLS project, choose top function as “amoebasat”
    - Include these files: source files (amoebasat.cpp, local\_rules\_update.cpp) & test bench (amoebasat\_test.cpp) - see the left pane of below figure



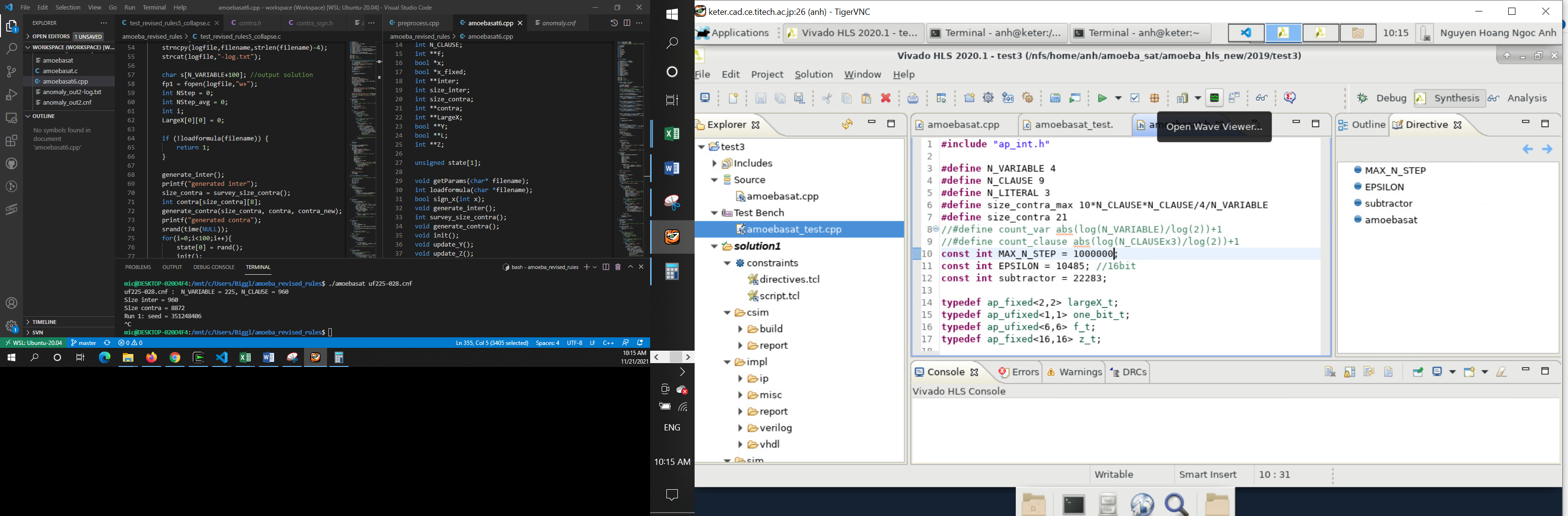
* Run C simulation (optional): If “PASSED” is printed then the C source code works correctly. (Note: this simulation runs gcc/g++ within Vivado HLS, so almost the same with C simulation in section 2).
* Run C/RTL Co-simulation: hit the button (next 2 buttons on the side of ‘Run C Simulation’ button) -> OK at the dialog

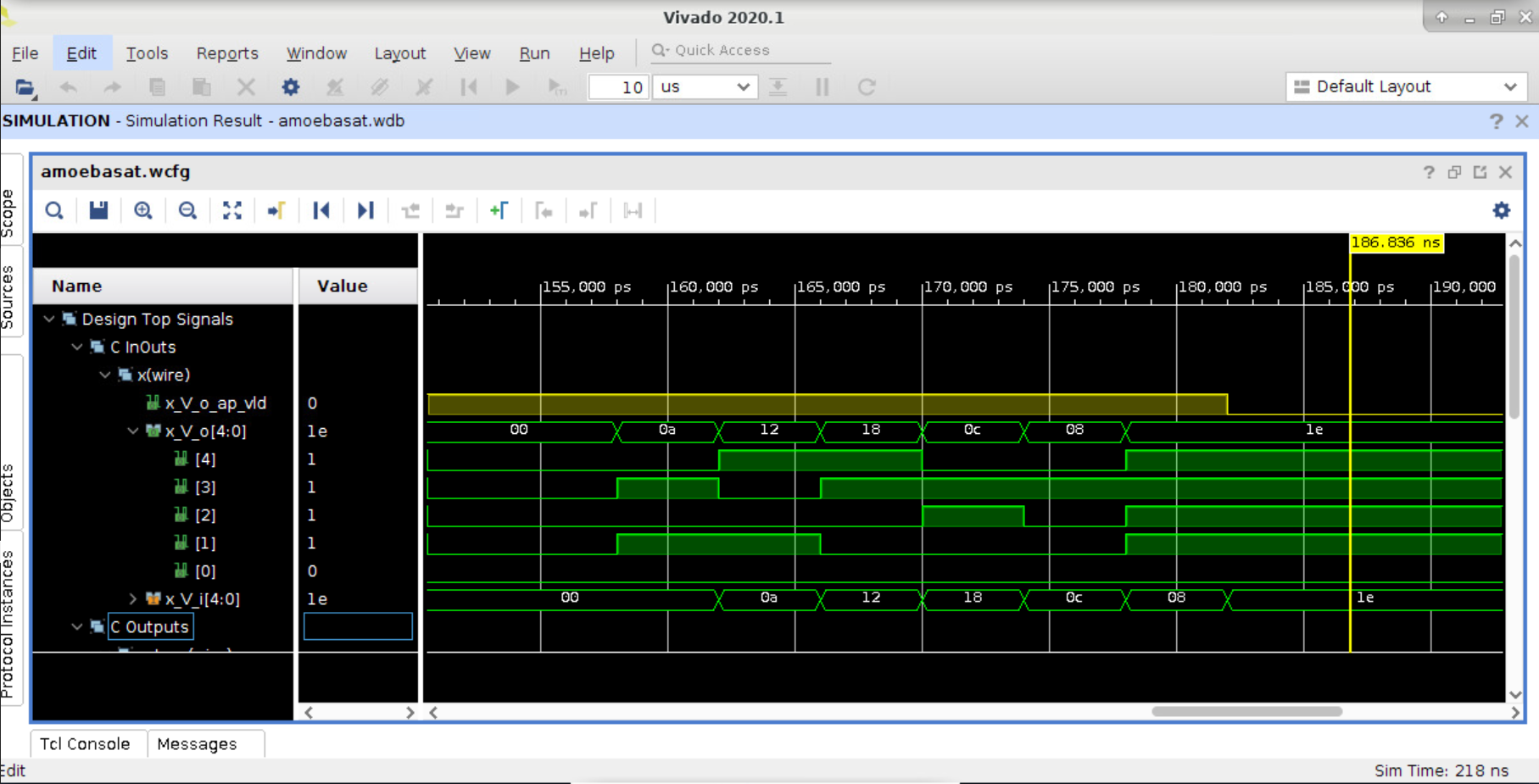


Cosimulation results: Latency at Verilog is the clock cycles#



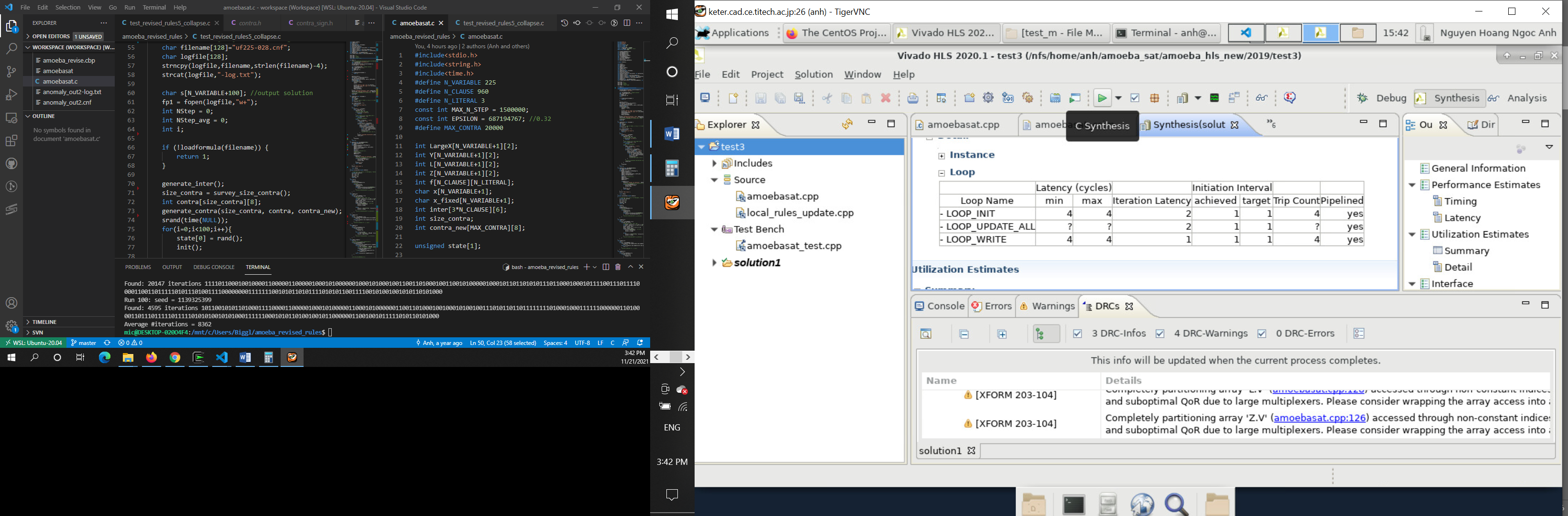
To open waveform view:



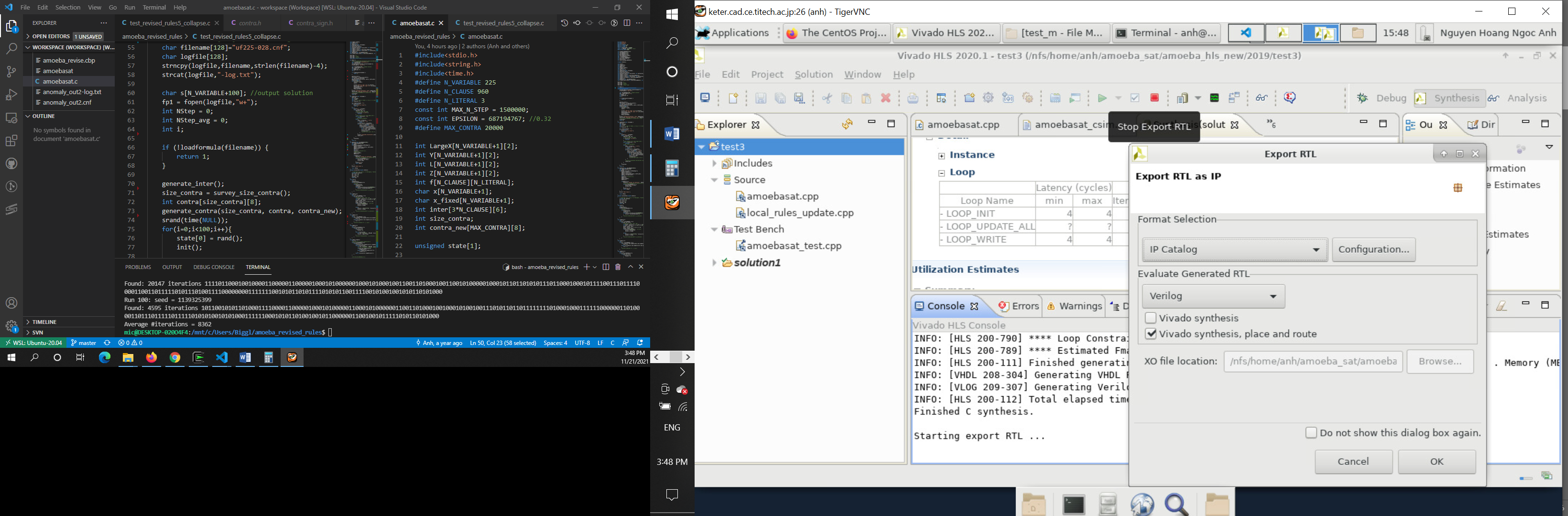


* Synthesize: Hit the C synthesis button

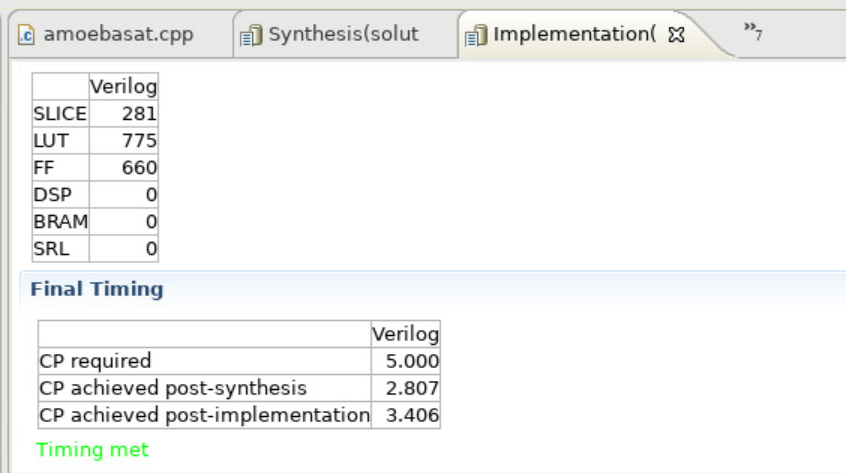
Ideally, we should get Innitiation Interval (achieved) =1



* + Export RTL: Hit the Export RTL button -> Check “Vivado Synthesis, place and route” if you want to get implementation results without manually opening Vivado



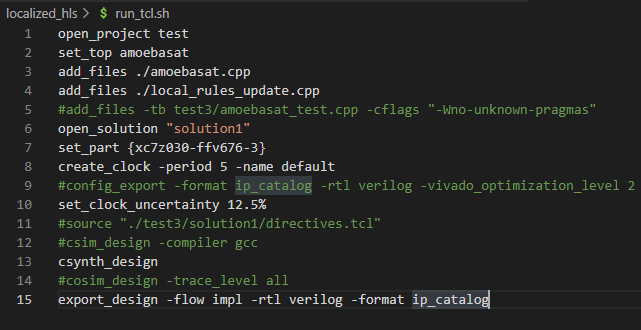
Implementation results



Final timing achieved post-implementation: implementation clock speed in ns

* 1. Using tcl script (recommended, much faster than GUI): run\_tcl.sh

To save more time, the followings (in green) are commented. Who wants to run all simulations can uncomment green parts



Desired clock period can be edit at line 8, unit is in nanosecond. For example, in the above figure, desired clock is set at 5ns.

Run: vivado\_hls run\_hls.tcl