ECE 449, Fall 2017
Computer Simulation for Digital Logic
Project Instruction

1 Summary

We present in this document the project instruction for ECE 449 including the grading policy. Our goal is to build a digital logic simulator using the C++ programming language that is able to simulate nontrivial digital systems including central processing units (CPU). As we will spend almost the whole semester to develop this simulator, we break it into five manageable projects including four standard ones and a bonus. The projects are organized in a way that you don't need to understand the whole simulator before you can demonstrate your progress.

Project specifications will be posted separately, which will indicate what you should accomplish for each individual project. Note that they will only specify the requirements instead of describing how the projects should be implemented. Though we will discuss in our lectures the details of each project implementation before its due date, it is much more rewarding if you could spend time to think on how *you* will solve the problems given the requirements.

The projects should be done individually. Discussions are encouraged. However, all the programs (except those from the lectures) and writings should be by yourself. COPY without proper CITATION will be treated as PLAGIARISM and called for DISCIPLINARY ACTION.

$\boxed{\textbf{NEVER share your programs/writings with others}}$.

2 Logic Simulation with the EasyVL Language

Our simulator simulates digital logics specified as synchronous circuits. A synchronous circuit is made up of inputs/outputs, wires, logic gates like and/or/not, and flip-flops driven by the global clock signal. It represents a finite state machine (FSM) where the current state is the boolean values stored in the flip-flops and and there is exactly one state transition per clock cycle. For each clock cycle, the next state and the current outputs are computed from the current inputs and the current state using the wires and the logic gates. Therefore, one of the tasks of our simulator, named logic simulation, is to determine the current inputs and the current state and to compute the current outputs and the next state repeatedly.

Given that hardware designers may use our simulator to simulate many different synchronous circuits, we should allow them to specify the circuits in a way that is convenient and familiar. While there are many established tools including schematic drawing programs and hardware description languages, it is extremely difficult for us to build one because of our limited knowledge on both digital logic simulation and C++. Therefore, we decide to design our own language called EasyVL for the designers to specify the circuits in a textual form. EasyVL is a simplification of one of the most popular hardware description languages called Verilog so that we can build a simulator in one semester. Nevertheless, it is powerful enough so that we can simulate any synchronous circuit once it is specified in EasyVL.

In the first two projects, we will study the EasyVL language in order to convert the textual form of a circuit stored in a file into a structural description stored in a program. In the third project, the netlist data structure consisting of wires and gates is built from the structural description, which is then used for logic simulation in the fourth project. On the other hand, one of the most important features in any hardware description language is to support hierarchical designs, which enables designers to build very complicated systems by building and reusing components. While our first four projects will not be able to handle hierarchical designs, the bonus project will be required to support this feature.

While we will give many details of our desired simulator in project documents and lectures, nothing could replace a working simulator that you can play with to gain hands-on experiences and to see what exactly each project should accomplish. Such a program is usually called *golden* since its outputs are thought to be superb. As discussed later, a golden EasyVL simulator is provided to you that not only generates desired output for each project but also serves as a tester to validate your program outputs. Please report any bug in the golden simulator to us, e.g. if it crashes or doesn't follow project documents.

3 Working with Your Projects

Please read *Guide to System Setup and Work Flow* before starting working on your projects and follow instructions there to obtain the initial project package.

3.1 Directory and File Conventions

Our continuous integration (CI) system depends on a proper structure of directories and files to understand your project submissions. The desired structure is provided with the initial project package as follows.

- The directory `src`: this is where you should put the C++ source files (`.h` and `.cpp`).
  - You should not put the source files into any sub-directory under `src`.
  - Feel free to name your source files.
  - The file `lex.cpp` is provided to you that serves as the start point of your projects.
- The directory `mytests`: this is where you should put your own test cases (`.evl` and other support files).
  - You should not put the files into any sub-directory under `mytests`.
  - Feel free to name your test cases.
  - The file `test.evl` is provided to you that serves as the start point of your tests.
- The directory `golden`: this is where the golden simulator and tests are located.
  - You should not change any file provided in this directory.
  - The file `EasyVL.exe` is the golden simulator for Windows systems.
  - The file `EasyVL` is the golden simulator for Linux systems, including the ECE UNIX network.
  - All the 10 golden test cases (`.evl` and other support files) are provided here.
- The directory `bonus`: this is where the bonus tests are located.
  - You should not change any file provided in this directory.
  - 4 bonus tests (`.evl` and other support files) are provided here.
- The hidden file `.gitignore`: this file keeps the Git repository clean.
  - You should not change this file unless you know how to update it correctly.
- The shell script file `build.sh`: this file helps to build your program on ECE UNIX servers.

It is very important for you to follow those conventions as otherwise your projects won't be graded.

3.2 Building Your Projects

Our CI system will use GCC 5.4.0 to build your project from the root of the Git repository using the following command.

```
g++ src/*.cpp -Wall -std=c++11 -O2 -lm -o myevl
```

This command first tell `g++` where to find all the source files (`src/*.cpp`), then specifies a few options including turning on all warnings (`-Wall`), using C++11 (`-std=c++11`), optimizing for performance (`-O2`), linking with the math library (`-lm`), and finally indicates the name of the output program (`-o myevl`).

On ECE UNIX servers including `uranus.ece.iit,edu` and `saturn.ece.iit.edu`, since the default GCC compiler is of a different version, you CANNOT use the above command directly to build your program. For your convenience, we provide the shell script file `build.sh` to include the necessary options to use GCC 5.4.0 and to enable debugging on these servers.

```
./build.sh
```

Note that since our CI system will build your project using all the files within the directory `src`, you need to remove C++ source files that are no longer used from that directory. It should not be a concern that you may lose previous project files since the Git system will track all changes as long as they are committed. Please consult the Pro Git book on how to retrieve previous versions of your source files.

3.3 Testing Your Projects

While our CI system can report whether your project fails on the test cases, no additional detail is provided currently and you should obtain why your project fails certain test case by repeating the test in your local Git repository. In addition, it is also a good idea to test your project before pushing it to our central repository.

To test your project for a specific test case, you should first remove any existing output file, and then run your program as usual with the `.evl` test file as the only argument,

```
rm golden/simple_comb.evl.tokens
./myevl golden/simple_comb.evl
```

Then, you need to run the golden simulator in the testing mode with the same test file as the first argument and the project as the second argument,

```
golden/EasyVL golden/simple_comb.evl proj=LEX testing=true
```

Repeat with other test files as necessary. As you may have noticed, the argument `proj=LEX` is for Project 1. The argument should be `proj=SYN` for Project 2, `proj=NET` for Project 3, and `proj=SIM` for Project 4 and 5. All these instructions can also be found by running the golden simulator without providing any argument.

Your program will pass a test case if it generates exactly the same output as the golden simulator. Otherwise, the golden simulator in the testing mode will report the first different line and you should work from there to debug your program.

4 Deliverables

We obtain a copy of your source files and test files as you push the changes to the central Git repository so there is no need for you to submit them to us using any other mechanisms. Moreover, please be advised that since to learn the use of Git and a CI system is among the objectives of this course, we will NOT accept project submissions outside the central Git repository, e.g. via emails. If you have difficulty accessing the central Git repository, it is your responsibility to act promptly to seek help from us well before the project deadlines; otherwise, not able to access the central Git repository is NOT an excuse for late submissions.

We separate each of the first four projects into two releases – the initial release (with the postfix '.0') and the final release (with the postfix '.1'). By doing so, we expect to help you to decompose each project into smaller pieces that you would feel comfortable to work with.

For the initial release, we would expect you to choose a subset of the required features from the project specification (will be posted separately), to design at least five (5) test cases for them as your own tests, and to implement your project accordingly. These test cases could and should be as simple as containing no more than 10 lines of EasyVL code. It is not necessary for your program to pass all the golden tests for the initial release but it should pass all of your own. Please make sure the golden simulator will process your own test cases without any error – you are not required to handle problematic test cases for all the projects, which otherwise would require a substantial amount of work.

On the other hand, it is expected that you implement all the required features for the final release. For each of the first four projects, you should also submit a project report of 2 to 4 pages to Blackboard. The project reports should discuss the following three items.

1. What features have you chosen for the initial release and why do you make such decisions?
2. How do you design the test cases for those features chosen by you?
3. How do you implement the initial release and the final release? More specifically, how does your class design evolve from the initial release to the final release?

For the bonus project, it is suffice to only push your project to the central Git repository. More details can be found in its specification.

Your program should perform reasonably well for all test cases. We set a hard limit on both the running time and the memory usage as listed in the table below:

Test Cases	Running Time	Memory Usage
Your own test cases.	15 seconds	750MB
bus.evl, io.evl, simple_comb.evl, simple_seq.evl.	30 seconds	750MB
counter_flat.evl, lfsr10.evl, tris_lut.evl.	60 seconds	750MB
s15850.evl, cpu8_flat.evl, cpu32_flat.evl.	600 seconds	750MB
Bonus test cases.	600 seconds	750MB

As a comparison, the golden simulator could complete all of them in less than 2 seconds.

5 Deadlines

The deadlines for each project are listed below.

	Due by the End of the Day		
Project Title	Initial Release	Final Release	Report
Project 1: Lexical Analysis	09/08 (Fri.)	09/15 (Fri.)	09/18 (Mon.)
Project 2: Syntactic Analysis	09/29 (Fri.)	10/06 (Fri.)	10/09 (Mon.)
Project 3: Netlist Construction	10/20 (Fri.)	11/03 (Fri.)	11/06 (Mon.)
Project 4: Logic Simulation	11/17 (Fri.)	12/01 (Fri.)	12/04 (Mon.)
Bonus: The EasyVL Simulator	12/01 (Fri.)		

6 Grading

Each of the first four projects will contribute 20 points to your final grade. The bonus project will contribute 15 points to your final grade. The grade for each project will be given based on the most recent push you have made to the central Git repository before the deadlines. Deadlines will NOT be extended except otherwise noticed. $\boxed{\textbf{Regrading will NOT be allowed.}}$.

The first four projects will be evaluated using the following four criteria:

- Properly formatted source code (1 points): with indentation and reasonable line width.
- Reasonable implementation (1 points): meaningful names and purposes for variables, functions, and classes.
- Project report (3 points): concise explanation of the three items mentioned in Section 4.
- You own test cases (5 points): 1 point for each of your own test cases that your program passes, up to 5 points total, as indicated by the testing report generated by our CI system. We will evaluate this criterion for the initial release only.

- The golden test cases (10 points): 1 point for each golden test that your program passes as indicated by the testing report generated by our CI system. We will evaluate this criterion for the final release only and please be aware of the performance requirement as discussed in Section 4.

Please be advised that for the last two criteria, no partial credits will be given. In addition, they only depend on the testing report generated by our CI system – we CANNOT test your program for grading outside of our CI system.

For the bonus project, each of the 5 test cases (one is hidden from you) contributes 3 points if your program could handle it correctly as indicated by the testing report. No partial credits will be given.