LCSL

Logic Circuit Simulation Language

Sachin Nene, Chaue Shen, Bogdan Caprita, Julian Maller

Introduction

1.1 Background

The logic circuit is an integral part of virtually all aspects of computational electronics. They form the backbone to the transistors that reside in the intricate microchips that run almost every piece of electronics today—from the airbags in a car to the alarm in a watch. The fundamental aspect of a logic circuit is that it operates on a digital signal that always carries one of only two values. The representation of these values varies through different technologies, whether it be a high or low voltage at a very low hardware level or Boolean representations of true and false in a high-level programming language.

Logic circuits act as a physical representation of a mathematical or logical function between a set of inputs and a set of outputs. Inputs are usual carried in by bit wires that carry a high or low voltage (which, again, can be translated to true/false values on a very high level). The physical medium that is a circuit transforms these signals in such a way as to produce a specific set of outputs which are sent out by bit wires as well. Combining several of these "function" circuits results in a useful tool for data, whether it be simple mathematical functions such as adding and subtracting, or at a much more complex level, displaying graphics on a CRT monitor. The average microchip in a computer contains millions upon millions of these circuits forming the processing power of today's computing products.

A structural representation is always a key component of the design process of a digital logical circuit. More and more, software and simulation is becoming an integral part in most fields of experimentation. Circuit research is not an exception, and simulation through software before even touching a piece of silicon is vital in making the exercise more efficient and worthwhile. Designing a digital logic circuit on a computer and simulating it with various inputs allows electrical engineers to tweak and prod at their discretion without the expense of materials or loss of valuable time.

1.2 Goals

1.2.1 Simplicity

The main purpose of designing this language is to simplify the process of creating digital circuits that electrical engineers must endure on two levels. On the first level, the fact that they can simulate the execution of a designed circuit before experimenting with actual hardware saves time, money, and patience. On the second level, the language allows an engineer to be able to program a simulation using familiar terms such as gates and switches rather than dealing with the esoteric intricacies of broader programming languages such as Java or C.

1.2.2 Extensibility

Simulating a circuit does not mean starting from scratch every single time one sit downs and programs. Rather, simulating a circuit in LCSL allows an engineer to build several components of the circuit(s) at different times and bringing them together at the end. It also allows the engineer to develop circuits that can be used

as components of other circuits or as a prototype for an improved version of the same circuit. The basic premise of the language is to bring small components together to form the building blocks of a larger goal.

1.2.3 Portability

LCSL is developed through the Java programming language, and implicitly, via the Java Virtual Machine. Because hardware-specific JVMs have been developed for almost all platforms, Java has become famous for being a highly portable language. Because LCSL is based on Java, it inherits the quality of portability.

1.2.4 Versatility

There are multiple ways to create circuits. One way is to define all the gates that are used and immediately configure them correctly. Another way is merely to define the inputs and outputs and to create a "black box" circuit which we have defined as *pseudo-circuits*.

1.2.5 Pseudo-circuits

Adding to the versatility is something that would be very difficult to do with a real circuit. A certain component of a circuit may have to act in a certain way, or in other words, produce a certain set of outputs for a given set of inputs. Sometimes the logic of this component isn't entirely clear and takes quite a bit of time just to figure out, thus inhibiting the entire process of forming the larger circuit. Instead, the engineer can, for simulation purposes, replace this component with a pseudo-circuit that acts as a temporary substitute in order to test other

components of the circuit. The pseudo-circuit can be broadly understood as a sort of lookup table where an efficient logical representation isn't necessary.

1.3 Sample Code

Here are some examples of programs in our language in order to show what the syntax looks like:

```
Circuit circuit1 {
      AndGate a;
      OrGate b;
      Switch s1, s2;

      HI -> a;  #bit value of 1 in first input of a
      b _> a;     #output of "b" in second input of a
      LO, s1 => b;      #bit value 0, flip switch in
                        #inputs of b

}

include::circuit1;
Simulation sim1 {
      int k;
      for (k = 0..10) {
            run(circuit1);
            if (k = 5) {
                  flip(circuit1.s1);
            }
            #print the output of gate b in circuit 1
            print(circuit1.b->);
      }
}
```

1.4 Summary

LCSL provides a convenient and intuitive way to design and test digital logic circuits without the hassle of time-consuming and expensive hardware experimentation.