Sandbox: A More Visual Hardware Description Language

MEGAN FILLION (mlf2179)
GABRIEL GUZMAN (grg2117)
DIMITRI LEGGAS (ddl2133)

September 26, 2017

Introduction

The motivation behind `Sandbox` was to create a sandbox-like environment where users can build and test digital circuits. We thought of this idea while thinking about our time in Fundamentals of Computer Systems. Most students with a background primarily in software programming struggle when they are confronted with the world of hardware. We were therefore driven to create a language that links the world of programming and electrical engineering—making it easier for us hardware rookies to visualize circuit construction in an setting that we understand. Unlike that of other hardware description languages, the syntax of `Sandbox` will allow the programmer to visually infer the design of the circuit. `Sandbox` will best be used alongside a sketchpad of circuits diagrams.

Language Description

We are aware that we are not the first to develop a hardware description language. Some of the principles of `Sandbox` take cues from `EHDL`—a student-designed language that models digital circuits. Although `Sandbox` and `EHDL` strive to simplify the verbosity of `VHDL` the syntax we implement in our language will look very different [2]. Because `Sandbox` has a didactic purpose, our syntax will emphasize the relationship between the elements of digital circuits. We will implement this using an indentation scheme. We do not plan to give the programmer as much control over the clock and timing as `EHDL` does.

This is how the typical sandbox program would work:

1. **Definition of Circuit Components.** These would look like functions in any language—the function ("circuit") will take arguments, which represent the circuit inputs, and given logical expressions that produce some number of outputs. A great feature of

`Sandbox` is that the most common circuit elements, such as multiplexers, decoders, adders, and registers, will be included in the standard library.

2. **Instantiation of Circuit Components.** Because the designer of a digital circuit will likely need multiple pieces of the same circuitry, we will need to devise a way to create instances of a given type of circuit. Whether the functions described above return instances of a function or a circuit datatype that can update its "output" fields as its "input" fields are changed has yet to be decided.

3. **The Sandbox Environment (Here's the fun part).** In what is essentially a main function we emulate a sandbox environment where programmers can demonstrate the link between each piece of their circuitry by using a sequence of indentations. If the user can visually infer the sequential logic of the circuit through code, they will better understand how each block works individually and as a part of the larger circuit.

The compiler will ensure that there are no issues in building the circuit. It will check for faulty or mismatched links between the user's circuit pieces. If a build-time issue is encountered, the compile will fail and return the location of the faulty link. We intend to implement an "expected output" feature. When running their program, which is essentially a large digital circuit, the user can provide a set of inputs and the expected outputs. `Sandbox` will check if the circuit works as expected, thus allowing the user to build and quickly test circuit-implementations of specific algorithms.

Keywords

`Sandbox` will use the following keywords and symbols (not exhaustive):

- Logical gates: `&&` (and), `||` (or), `!` (not), `^` (xor)
- Types: `int` corresponds to bits, `char` to AASCI
- Built-In Circuits: `mux`(...), `dec`(...), `add`(...) are a few examples.
- I/O: `INPUT` and `OUTPUT` to denote user input and output. We will include a function to prompt the user for input.
- Timing: `CLOCK`, `ENABLE`, `RESET`
- `SANDBOX` will denote the start of the sandbox enviornment

Example Program

As an example of what a `sandbox` program would look like, we implement a circuit for a 4-bit ripple carry adder. The design of this circuit can be found in Figure 1.

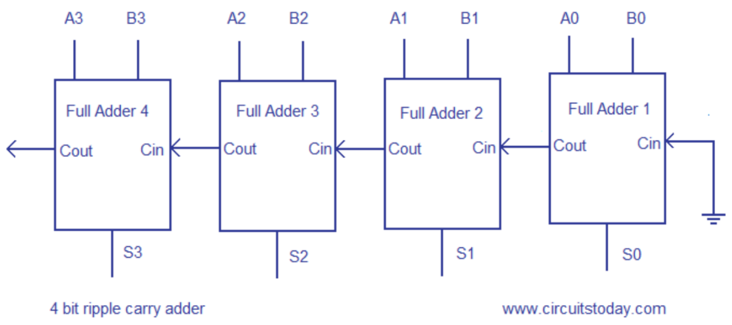

Figure 1: A digital circuit that computes the sum of two unsigned 4-bit integers. Diagram found on circuitstoday.com.

The program follows. Everything up until `SANDBOX` should be fairly self-explanatory. However, as for our indentation scheme... Each level of indentation implies some transfer of signal. For example, the statement `FA2(1->2)` implies that the code block `FA2` will take the **first output** of the block on the preceding level of indentation (`FA1`) as its **second input**. The order of inputs is defined when the circuit pieces are defined.

```
// Taking 4-bit inputs from the user and setting the carry-in at 0.
int[4] A = user_in(4);
int[4] B = user_in(4);
int Cin = 0;
int[4] SUM;

// defining and instantiating full adders
(int sum, int carry) fulladder(int a, int b, int cin){
   sum = a ^ b ^ cin;
   carry = (a && b) ^ (cin && (a ^ b));
}

FA0 = fulladder();
FA1 = fulladder();
FA2 = fulladder();
FA3 = fulladder();

SANDBOX:
Cin
   FA0(INPUT->2)
      FA1(1->2)
         FA2(1->2)
            FA3(1->2)
```

```
A[0]
  FA0(INPUT->0)
    S[0](0->OUTPUT)
B[0]
  FA0(INPUT->1)
A[1]
  FA1(INPUT->0)
    S[1](0->OUTPUT)
B[1]
  FA1(INPUT->1)
A[2]
  FA2(INPUT->0)
    S[2](0->OUTPUT)
B[2]
  FA2(INPUT->1)
A[3]
  FA3(INPUT->0)
    S[3](0->OUTPUT)
B[3]
  FA3(INPUT->1)
```

Given the multiplicity of inputs and outputs for each code block, there are clearly multiple ways to define the same circuit in the `SANDBOX` environment. This ambiguity will allow the programmer to arrange their elements in the most visually coherent and instructive way. We have not decided on a definite syntax and the one demonstrated above clearly needs some slight modifications to remove ambiguities. However, this example highlights the indentation scheme that is the main idea of `Sandbox`.

References

[1] M. Morris Mano, Charles R. Kime, Tom Martin Logic and Computer Design Fundamentals, Pearson Higher Education, Inc. (2015).

[2] Paolo Mantovani, Mashooq Muhaimen, Neil Deshpande, Kaushik Kaul, Easy Hardware Description Language, `http://www.cs.columbia.edu/~sedwards/classes/2011/w4115-fall/proposals/EHDL.pdf` (2011).

[3] Ram Koganti, Khader Mohammad, Greatest Common Divisor Logic Circuit, `http://web.cecs.pdx.edu/~mperkows/CLASS_VHDL/VHDL/gcd/assignme.htm` (1998).