ASL: Auburn Simulation Language

by Charles E. Stroud, Professor
Dept. of Electrical & Computer Engineering
Auburn University
July 7, 2003

ABSTRACT

A simulation language for describing digital logic circuits is presented. The simulation language, called ASL (Auburn Simulation Language), supports the hierarchical description of digital circuits for design verification through logic and fault simulation.

INTRODUCTION

The complexity of current digital logic circuits, the widespread use of Very Large Scale Integration (VLSI), and the desire to shorten the design interval required to develop a concept into a product require extensive design verification prior to the hardware implementation. Without the existence of the hardware, such design verification is possible only through computer simulation. In order to simulate the design, the digital circuit must be described in a manner such that it is compatible with the simulation software. This requires a language and syntax which can be efficiently generated by designers and/or Computer-Aided Design (CAD) tools as well as be efficiently parsed and interpreted by the simulation software. This document describes such a digital logic simulation language developed in the Department of Electrical & Computer Engineering at Auburn University and called Auburn Simulation Language (ASL).

CIRCUIT DESCRIPTION

To begin, consider the logic diagram for the full adder illustrated in Figure 1. The full adder is capable of adding two bits with a carry-in input to produce the sum and a carry-out signal. As a result the circuit has three inputs (in this example, referred to as A, B, and C where C is assumed to represent the carry-in input) and two outputs (the sum, S, and the carry-out, CO). The logic circuit consists of two exclusive-or gates (XOR) to generate the sum along with three 2-input AND gates and a 3-input OR gate to generate the carry-out. The fact that the three AND gates and the two XOR gates are unique in that each gate has a different set of input signals can be easily seen visually from the logic diagram. When discussing the circuit, a designer may refer to "the middle AND gate" or "the first XOR gate" in order to distinguish between the multiple uses of the same type of gate. Similarly, the internal signal nodes or "nets" of the circuit are also unique (for example, "the output of the top AND gate").

Although the uniqueness of the various gates and nets is graphically apparent in the logic diagram, the simulation language represents a textual description of the circuit and requires explicit reference to a given gate or net. As a result, an unambiguous logic diagram for the full adder is given in Figure 2 where each gate and net of the circuit is given a arbitrary but unique name. For example, the three AND gates have been named A1, A2, and A3. In addition, the nets driven by the outputs of the three AND gates have been given the same names (A1, A2, and A3). Naming a net the same as the name of the driving gate is not a rule (as can be seen by the output net of the XOR gate, X2, which produces the sum, S) but rather aids in identification of the source of the signal as well as simplifies the effort of inventing names for all of the gates and nets. From the logic diagram of Figure 2, the ASL description for the full adder can be written. The ASL description consists of a single "circuit statement" and one or more "component statements". The "circuit statement" includes the name of the complete logic circuit, its primary inputs, and its primary outputs. The circuit statement for the full adder would be written as follows:

```
CKT: FADD IN: A B C OUT: S CO ;
```

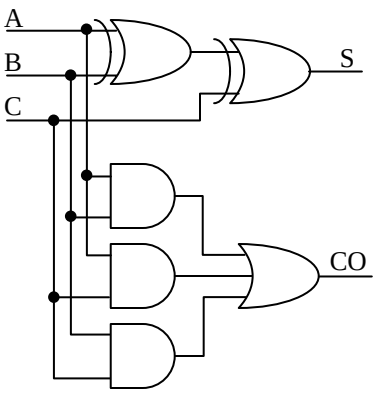

Figure 1. Logic diagram for full adder circuit.

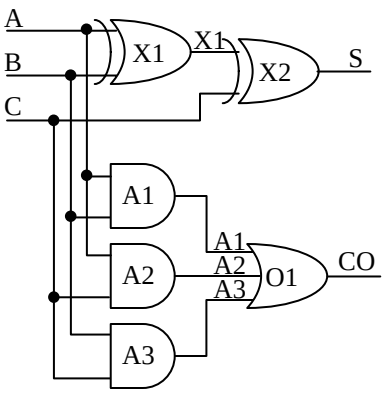

Figure 2. Completely specified logic diagram for full adder circuit.

The syntax of the circuit statement begins with the keyword "CKT:" to indicate that the statement is a circuit statement as opposed to a component statement. The "CKT:" keyword is followed by the name of the logic circuit (arbitrarily called FADD in this case for full adder) and separated by a delimiter. In ASL, delimiters can be any combination of SPACE, TAB, or NEW LINE characters. The circuit name is followed (after another delimiter) by the keyword "IN:" to indicate that the primary inputs to the circuit will subsequently be defined. The names of all of the primary inputs to the circuit are given immediately following the "IN:" keyword and are separated by delimiters. After all primary inputs have be listed, the keyword "OUT:" is used to indicate the beginning of the list of primary outputs. The list of primary outputs is specified in the same manner as the primary inputs with delimiters separating each entry. All ASL statements (regardless of type) are concluded with a delimiter followed by a semicolon (";").

The circuit statement indicates the beginning of the circuit description and is followed by component statements, each of which describes one component (a gate in this case) used to construct the complete circuit. The syntax of the component statement is similar to that of circuit statement with the exception of the "CKT:" keyword. In the component statement, the type of component (for example, "XOR:", "AND:", or "OR:" for the full adder circuit) is used as the keyword instead of the "CKT:" keyword. In the same manner as the circuit statement the component is given a unique name followed by the "IN:" keyword, the list of inputs to the component, the "OUT:" keyword, the list of outputs of the component, and the semicolon indicating the end of the statement. The component statement for the first XOR gate in the full adder would be as follows:

```
XOR: X1 IN: A B OUT: X1 ;
```

Notice that once the logic diagram has been completely specified with unique gate and net names, the circuit and component statements can be generated directly from the information in the logic diagram. As a result, the complete circuit description for the full adder of Figure 2 would be given by:

```
CKT: FADD IN: A B C OUT: S CO ;
XOR: X1 IN: A B OUT: X1 ;
XOR: X2 IN: X1 C OUT: S ;
AND: A1 IN: A B OUT: A1 ;
AND: A2 IN: B C OUT: A2 ;
AND: A3 IN: A C OUT: A3 ;
OR: O1 IN: A1 A2 A3 OUT: CO ;
```

One final note at this point is that the use of upper case letters is not mandatory in generating the ASL such that lower case letters may be used. However, the upper and lower case characters may not alias such that "a" and "A" are not the same and if used as net names may represent two separate nets or signals. Such aliasing properties are a function of the

simulation software which parses the circuit description. For example, if the output of the first XOR gate is labeled "X1" and the input to the second XOR gate is labeled "x1", no connection between the gates exists in CAD tools which distinguish between upper and lower case characters. (This is the case in the parser for AUSIM [1].)

ELEMENTARY LOGIC GATES

The logic gate simulation models supported are a function of the simulator being used (and not necessarily the hardware description language itself). For example, the most basic version of AUSIM supports the five elementary logic gates (AND, OR, NAND, NOR, and NOT). More advanced version of AUSIM support exclusive-OR (XOR) and exclusive-NOR (NXOR) gates as well as D-type flip-flops for both rising (DFF) and falling (NDFF) edge-triggered operation.

COMMENT STATEMENTS

Comments in circuit descriptions are useful in improving the readability of the circuit description or to include notes about the functional operation. A comment statement in ASL is indicated by a "#" at the beginning followed by a delimiter (remember that a delimiter in ASL is any combination of SPACE, TAB, or NEW LINE characters). The text of the comment follows and can extend over any number of lines. The comment is completed in the same manner as all other statements with a delimiter followed by a semicolon. Comment statements can be included at any point in the ASL description as long as they are inserted between statements and not within statements. For example, a valid commented version of the full adder circuit description could be as follows:

```
# ASL description for a full adder ;
CKT: FADD IN: A B C OUT: S CO ;
# summing circuit ;
XOR: X1 IN: A B OUT: X1 ;
XOR: X2 IN: X1 C OUT: S ;
# carry-out circuit ;
AND: A1 IN: A B OUT: A1 ;
AND: A2 IN: B C OUT: A2 ;
AND: A3 IN: A C OUT: A3 ;
OR: O1 IN: A1 A2 A3 OUT: CO ;
# end of ASL description for full adder ;
```

HIERARCHICAL CIRCUIT DESCRIPTION

For a large circuit, the numbers of components may require many component statements and, as a result, a good deal of effort in generating the circuit description. When multiple uses of one or more subcircuits in the complete circuit is prevalent in the design, a hierarchical description of the complete circuit is desirable in order to reduce the effort required to generate the description as well as to reduce the chance of syntax errors. A hierarchical description of a circuit also improves the readability of the circuit description by allowing the designer to functionally group components (this is true even when multiple uses of subcircuits is not a characteristic of the given design).

ASL supports hierarchical circuit descriptions via the use of a subcircuit statement which is indicated by a "SUBCKT:" keyword. As an example, suppose that an elementary gate level description of the XOR gates used in the full adder is required by the simulation software. The ASL description of the full adder can be made to accommodate the simulator requirements without modification of the full adder ASL, but rather by the addition of a subcircuit description for the XOR gate. An elementary gate level design for an XOR gate is shown (completely specified) in Figure 3 and the subcircuit description is given as follows:

```
SUBCKT: XOR IN: A B OUT: Z ;
NOR: G1 IN: A B OUT: G1 ;
```

```
AND: G2 IN: A B OUT: G2 ;
NOR: G3 IN: G1 G2 OUT: Z ;
```

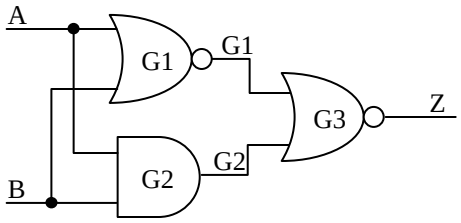

Figure 3. Logic diagram for elementary logic gate representation of an exclusive-OR.

The subcircuit description for the XOR gate can be appended or prepended to the circuit description for the full adder to complete the hierarchical ASL description. The only requirement to keep in mind is that ALL components for a circuit or subcircuit must follow the circuit or subcircuit statement, respectively, before another circuit or subcircuit description can begin with its circuit or subcircuit statement. The hierarchical ASL description for the full adder would now be given as follows:

```
# functional description of XOR gate at the elementary gate level ;
SUBCKT: XOR IN: A B OUT: Z ;
NOR: G1 IN: A B OUT: G1 ;
AND: G2 IN: A B OUT: G2 ;
NOR: G3 IN: G1 G2 OUT: Z ;
# ASL description for a full adder ;
CKT: FADD IN: A B C OUT: S CO ;
XOR: X1 IN: A B OUT: X1 ;
XOR: X2 IN: X1 C OUT: S ;
AND: A1 IN: A B OUT: A1 ;
AND: A2 IN: B C OUT: A2 ;
AND: A3 IN: A C OUT: A3 ;
OR: O1 IN: A1 A2 A3 OUT: CO ;
```

As an additional example of the use of hierarchy in a circuit description, one can build a four bit (or any number of bits) adder from the circuit description of the full adder. This can easily be done by changing the "CKT:" statement for the FADD to a "SUBCKT:" statement and adding a "CKT:" statement to interconnect four full adder circuits to create the four bit adder as shown in

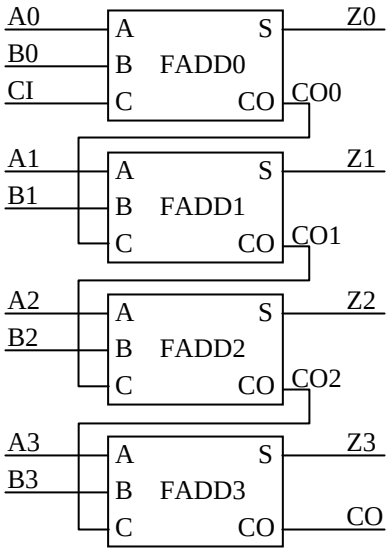

Figure 4. 4-bit adder constructed from 4 full adders.

Figure 4. The interconnections are made by connecting the carry-out signal of a given stage of the adder to the carry-in input of the subsequent stage. The resulting ASL circuit description would be given by:

```
SUBCKT: XOR IN: A B OUT: Z ;
NOR: G1 IN: A B OUT: G1 ;
AND: G2 IN: A B OUT: G2 ;
NOR: G3 IN: G1 G2 OUT: Z ;
# ASL description for a full adder ;
SUBCKT: FADD IN: A B C OUT: S CO ;
XOR: X1 IN: A B OUT: X1 ;
XOR: X2 IN: X1 C OUT: S ;
AND: A1 IN: A B OUT: A1 ;
AND: A2 IN: B C OUT: A2 ;
AND: A3 IN: A C OUT: A3 ;
OR: O1 IN: A1 A2 A3 OUT: CO ;
# ASL description for 4-bit adder ;
CKT: ADD4
IN: A3 A2 A1 A0 B3 B2 B1 B0 CI
OUT: CO Z3 Z2 Z1 Z0 ;
FADD: Z3 IN: A3 B3 CO2 OUT: Z3 CO ;
FADD: Z2 IN: A2 B2 CO1 OUT: Z2 CO2 ;
FADD: Z1 IN: A1 B1 CO0 OUT: Z1 CO1 ;
FADD: Z0 IN: A0 B0 CI OUT: Z0 CO0 ;
```

Notice that there is only one "CKT:" statement in the ASL description and that the full adder has been changed to a subcircuit description by modification of the circuit statement. It is also important to notice the ordering of the inputs and outputs in the "FADD:" component statements that must be observed to obtain the correct functionality. This requirement is most easily seen by sum and carry-out outputs of the full adder such that, in the "FADD:" subcircuit statement, the sum output precedes the carry-out. In the four "FADD:" component statements in the ADD4 circuit description, the four sum outputs precede the carry-outs as well. This example points out the fact that ASL uses *positional notation*, meaning that connectivity between the net names specified in a component statement and the corresponding "SUBCKT:" statement is maintained on the basis of the position of the net name in the lists of inputs and outputs.

There is no specific limit on the number of levels of hierarchy that can be used in developing a circuit description. However, since each gate and net must be unique in order to avoid multiple gates driving the same net, hierarchical descriptions must also imply a unique identity for each gate. For example, in the ADD4 circuit' each "FADD:" component statement implies a completely separate and unique FADD circuit rather than all four FADD sharing the same logic. As a result, there are actually eight XOR gates described in the complete circuit description for the ADD4 circuit as well as twelve AND gates and four OR gates. The uniqueness of the identity of each gate and net name can be obtained through concatenation of the name given to the component in the circuit description and the name given to the gate in the subcircuit description. In fact, the parser for the simulation software must perform this concatenation to remove the hierarchy prior to simulation. The process of removing the hierarchy is often called *flattening* the hierarchical description. When the hierarchical description of the ADD4 circuit is *flattened*, the circuit description consists of 40 AND, OR, and NOR gates, each with a unique name and each driving a unique net. When compared to the 15 statements used to describe the ADD4 hierarchically, the value of such a capability in reducing the design and development effort begins to become evident. The following ASL description represents an example of a flattened circuit description for the hierarchical ASL description of the full adder given previously:

```
# flattened ASL description for a full adder ;
CKT: FADD IN: A B C OUT: S CO ;
NOR: X1_G1 IN: A B OUT: X1_G1 ;
AND: X1_G2 IN: A B OUT: X1_G2 ;
```

```
NOR: X1_G3 IN: X1_G1 X1_G2 OUT: X1 ;
NOR: X2_G1 IN: X1 C OUT: X2_G1 ;
AND: X2_G2 IN: X1 C OUT: X2_G2 ;
NOR: X2_G3 IN: X2_G1 X2_G2 OUT: S ;
AND: A1 IN: A B OUT: A1 ;
AND: A2 IN: B C OUT: A2 ;
AND: A3 IN: A C OUT: A3 ;
OR: O1 IN: A1 A2 A3 OUT: CO ;
```

To improve the readability of the flattened description, the underscore character ("_") was used in the concatenation of the circuit and subcircuit names. Designers may be denied the used of such special characters when generating a circuit description if the character is reserved for use by the parsing software. (The underscore character is a reserved character in the parser for AUSIM and may not be used by the designer when generating ASL for AUSIM simulation [1].) In addition, there may exist limits in terms of the number of characters that may be used in a name, including flattened names, and, as a result, the designer should be aware of these limitations and consider them in determining the length of names and the number of levels of hierarchy in a given circuit description (particularly when additional characters are inserted during the concatenation process of flattening).

SUMMARY

A simulation language, called ASL, for describing digital logic circuits has been presented. The language is quit flexible in that there are few rules to observe, yet, circuits can be described hierarchically to improve designer productivity and the ability to describe circuits for different simulators. However, designers should be aware that some rules or limitations may be implied as result of the application specific parsing software. Finally, the delimiter and syntax used in ASL improves parsing when compared to those used in other simulation languages.

REFERENCES

[1] Charles E. Stroud, "AUSIM: Auburn University SIMulator - version 2.0", Dept. of Electrical & Computer Engineering, Auburn University, July 7, 2003.