Simple Circuit Description Language: An Overview

Michael Clear

1 Introduction

SCDL is a simple functional language to describe an arithmetic circuit over some ring. It has two basic operations, addition (+) and multiplication (*). The language can be interpreted in any ring. For example, when interpreted in the Boolean field, "+" corresponds to XOR and "*" corresponds to AND.

2 Syntax

To illustrate the syntax, below is the definition of the function "not"

```
constant one = 1
func not(x) = x + one
```

The code above first defines the constant "one". The next line defines the function "not" as its argument XORed with "one". Next we define the function "eq" that tests the equality of its two arguments:

```
func eq(x, y) = not(x + y)
```

Similarly we can define the "or" function:

```
func or(x, y) = x + y + (x*y)
```

Now we define a function that determines whether a 2-bit unsigned value is greater than another 2-bit unsigned value. Note that the backslash indicates that the definition continues on to the next line.

```
func gt(X : 2, Y : 2) = \
  or(X[1]*not(Y[1]), eq(X[1], Y[1])*X[0]*not(Y[0]))
```

The examples illustrate the syntax of a function definition. The definition begins with the keyword "func" and is followd by a function name along with a list of formal parameters that are comma-separated and within paranthesis. If there are no formal parametres, the paranthesis may be dropped. An argument to a function may be a vector whose length is specified after a colon. If this is omitted the argument is taken to be a scalar. In the case of a vector argument, say "X" of length k, the i-th element of a vector can be accessed with the syntax $X[i]$ where $0 \leq i < k$.

2.1 Inputs

An input variable with the name x is declared with the syntax

```
input x
```

An input variable may also be a vector which is specified in the same manner as that above. So to declare a vector input variable with the name X whose length is 8, we can write

```
input X : 8
```

An element of a vector is accessed in the same way as that above. Note that all names are case-sensitive. If an undeclared identifier is specified in the body of a function, a new input variable is automatically declared with that name.

2.2 Include Statement

One may include the definitions specified in another file with the "include" statement. Therefore to include a file "library.scdl" we can write

```
include ``library.scdl''
```

2.3 Base Library

Some useful functions such as "not", "or" and "eq" as defined above are incorporated in a base library file with the name "base.scdl" which is packaged with our interpreter.

2.4 Constants

As mentioned earlier a constant can be declared with the keyword "constant" which assigns a name to an integer value. The integer value is interpreted in the associated ring, which in our implementation is the field of 2 elements. This illustrated with the declaration of the constant "one" above.

3 Sample Program

We are now ready to present a straightforward sample program that illustrates the syntax. In this program the output is a single Boolean value that indicates whether one 2-bit unsigned value A is greater than another 2-bit value B. Our convention is that an SCDL program is contained in a ".scdl" file. This sample program is included with our interpreter in the file gt.scdl.

```
include ``base.scdl''

input A : 2
input B : 2
```

```
func gt(X : 2, Y : 2) = \
  or(X[1]*not(Y[1]), eq(X[1], Y[1])*X[0]*not(Y[0]))

func out = gt(A, B)
```

Note that there is nothing special about the function named "out" except that it can be evaluated directly from the inputs alone. One could define several functions without arguments that could also serve as outputs. As we will see later we can tell the evaluator which of these functions constitute our desired outputs.

4 Interpreter

4.1 Vars File

Our implementation of an interpreter for SCDL allows a description of the inputs and outputs to a program with respect to high-level types (such as integers) to be supplied in an accompanying JSON file referred to as a *vars* file, which has the extension ".scdl.vars". The vars file is used to map high-level input/output variables to input/output variables in SCDL. We first give a vars file for our sample program above which is included with our interpreter as gt.scdl.vars.

```
{
  "inputs":[
    {
      "name":"A",
      "type":"uint<2>",
      "components":[
        "A"
      ]
    },
    {
      "name":"B",
      "type":"uint<2>",
      "components":[
        "B"
      ]
    }
  ],
  "outputs":[
    {
      "name":"gt",
      "type":"bool",
      "components":[
        "out"
      ]
```

```
        }
    ]
}
```

This vars file specifies two input variables and one output variable. The first input variable has the name “A” and the type *uint⟨2⟩* i.e. an unsigned integer of length 2 bits. We refer to the SCDL variables it maps to as its components. In this case there is only a single component, namely the SCDL vector variable also named “A”. Similarly the second input variable has the name “B” along with the type *uint⟨2⟩* and maps to the vector variable “B” in our SCDL program above. The output variable has the name “gt” and is of type *bool*. It maps to the SCDL function “out” which evaluates to a Boolean value.

4.1.1 Types

The currently supported types are *int*, *uint* and *bool* which denote a signed integer (in 2’s complement representation), an unsigned integer and a Boolean value respectively. Note that the first two require a length in bits to be specified within angle brackets i.e. we would write *uint⟨8⟩* to specify an 8-bit unsigned integer.

4.1.2 Components

The components of a variable are the SCDL input variables in the case of inputs and SCDL functions with no arguments in the case of outputs. There can be multiple components associated with a variable. The convention is that the components are ordered such that the first bit of the first component (if it is a vector) corresponds to the LSB of the variable’s value. Suppose an input variable (say a 2-bit unsigned integer) has two components “[x0, x1]” where “x0” is an SCDL scalar input variable and “x1” is an SCDL scalar input variable. Then “x0” corresponds to the LSB of the variable’s value and “x1” corresponds to the MSB. Furthermore, suppose a 9-bit unsigned integer has two components “[X, b]” where “X” is an SCDL vector input variable of length 8 and “b” is an SCDL scalar input variable. Then the first 8 bits of the variable’s value from bit 0 to bit 7 map to “X” and bit 8 maps to “b”.