DESCRIBING AND TESTING ARITHMETICAL CIRCUITS IN A FUNCTIONAL LANGUAGE

Frederico A. Mameri *fred@comp.ufu.br*
Hélio D. Batista Júnior *darlan@comp.ufu.br*
Nélio M. M. Alves *nelio@comp.ufu.br*
Sérgio M. Schneider *sergio.schneider@facom.ufu.br*

Faculdade de Computação - Universidade Federal de Uberlândia

ABSTRACT

We are working on a research project, which includes the implementation of a hardware description language (HDL) embedded in the lazy functional language Haskell. This paper discusses how "functional" HDLs can ease the task of describing and testing arithmetical circuits structurally. Some small, didactical examples are shown using the implemented language.

1. INTRODUCTION AND MOTIVATION

This work follows a particular style of hardware description, which models the full history of the sequence of values on a wire by a stream, where the *i*th element of the stream represents the value carried by the wire at the *i*th clock tick in a simulation. In this context, circuit components are modeled by functions that, essentially, receive input streams as arguments and return output streams as results. Simpler components can be grouped together to generate more complex ones, as in functional composition.

This description style is not new. Since the early eighties [4] many efforts have been carried out in that direction. However, such style is probably new to many hardware designers, usually involved with widely used languages such as VHDL and Verilog.

During the design process of arithmetical circuits in widely used HDLs, such as VHDL, we observed a few aspects not interesting to the designer, such as the difficulty of proving the correctness of the circuit (out of the scope of this paper), the length of the code and the constant concernment about the language syntax. On the other hand, "functional" HDLs have shown to be both powerful and easy, and also very productive.

This paper presents the advantages of describing arithmetical circuits in a functional language over widely used HDLs, which is part of a larger project held by the computer science graduate program of our university.

1.1. OVERVIEW OF THE PAPER

Section 2 and Section 3 are concerned with the advantages of describing arithmetical circuits in a functional HDL and testing the circuits respectively. Then we cite related work and, finally, we discuss conclusions and future work.

2. DESCRIBING CIRCUITS

The description of arithmetical circuits in a language implemented as a Haskell [3] embedded library inherits all the advantages of it, such as lazy evaluation (allowing mutually recursive definitions appearing in circuits with feedback), productivity due to short code, ease of proving the correctness of the circuit (because of the very simple semantics of this description style), strong typing, and legibility.

2.1. A SIMPLE EXAMPLE

A half adder is a circuit that takes two bits as input and outputs their sum and the carry out bit. Let us consider a VHDL code for a half adder:

```
library ieee;
use ieee.std_logic_1164.all;

entity halfAdd is
   port (a, b : in std_logic;
         sum, carry : out std_logic);
end halfAdd;

architecture comp of halfAdd is
begin
  carry <= a and b;
  sum <= a xor b;
end comp;
```

It is very clear that most of the code is spent with the language syntax rather than with the circuit description itself. Below we describe the same circuit using Haskell:

```
halfAdd (a, b) = (sum, carry)
  where
    sum   = xor2 (a, b)
    carry = and2 (a, b)
```

The above function gives an idea of how functional languages can be powerful, productive, and simple. The simplified syntax increases the designer productivity and also the understand of someone who is learning digital circuits, once one will be able to focus on the circuit itself, rather than on the language used for describing it. We are supposing that the logical gates `xor2` and `and2` belong to the basic library of the language.

2.2. A BIGGER EXAMPLE

Next we show the full gate-level description of a sequential multiplier and its subsidiary circuits: a full adder and a shift register. The former is a circuit that takes two operand bits and a carry-in bit as input and outputs their sum and the carry out bit. The ladder is a

circuit used mainly for storage of digital data. Finally, the sequential multiplier is an arithmetical circuit that takes two numbers of four bits and outputs their product.

```
mult (reset,[a0,a1,a2,a3],[b0,b1,b2,b3]) = s
 where
  s1 = shiftreg (reset, [a0,a1,a2,a3])
  s20 = and2 (b0,s1)
  s21 = and2 (b1,s1)
  s22 = and2 (b2,s1)
  s23 = and2 (b3,s1)
  (s,z1)  = fullAdd (dff z1,(s20,dff y1))
  (y1,z2) = fullAdd (dff z2,(s21,dff y2))
  (y2,z3) = fullAdd (dff z3,(s22,dff s23))
```

In the above code, `dff` is a D flip-flop, which also belongs to the basic library.

Designing the same circuit in VHDL would be very cumbersome, for its code is very long and not straightforward.

The Haskell code of the full adder and the shift register is shown below:

```
fullAdd (carryIn, (a, b)) = (sum, carryOut)
  where
    (s1,  c1) = halfAdd (a, b)
    (sum, c2) = halfAdd (carryIn, s1)
    carryOut  = or2 (c1, c2)

shiftreg (reset, [x0,x1,x2,x3]) = out
  where
    out = mux1 (reset, (dff sx1, x0))
    sx1 = mux1 (reset, (dff sx2, x1))
    sx2 = mux1 (reset, (dff sx3, x2))
    sx3 = mux1 (reset, (low, x3))
```

3. TESTING CIRCUITS

Testing the circuit is very simple. All one has to do is using the `sim` function, provided by the system we implemented, which receives the circuit and an input list. Below there is an example of a halfAdd simulation, from a Haskell interpreter prompt:

```
Circ> sim halfAdd [(high,low),(high,high)]
[(1,0),(0,1)]
```

Testing the multiplier is also very easy. Below we define an input list (in the source code), representing the multiplication of 9 and 7:

```
x = [high,low,low,high]  -- (decimal 9)
y = [high,high,high,low] -- (decimal 7)
input = [(high,x,y),(low,x,y),(low,x,y),
         (low,x,y), (low,x,y),(low,x,y),
         (low,x,y), (low,x,y)]
```

Now, the simulation in the Haskell interpreter prompt can be performed:

```
Circ> sim mult input
[1,1,1,1,1,1,0,0]
```

The resulting value is the decimal 63, as expected.

The lazy evaluation allows us to provide a list of as many signals as wanted. That same feature provides infinite lists evaluation, which can be very interesting to simulate real systems, for the signals in them are potentially infinite.

4. RELATED WORK

Our work was strongly inspired in Lava [2], a system embedded in Haskell that can perform interesting circuit interpretations including simulation, verification and translating to VHDL. Another interesting HDL embedded in Haskell is Hawk [5], created for designing superscalar microprocessors, but implemented in a different way if compared with Lava.

5. CONCLUSIONS AND FUTURE WORK

Functional languages provide clear, concise and polymorphic specifications that can be easily tested through a simple simulation function. Combined with lazy evaluation, high-order functions and type classes, which form the basis of the modern functional languages, can significantly increase the productivity in the logic design process.

Our next step is describing and simulating microprocessors, that is, a set of digital and arithmetical circuits that takes instructions and executes them. Also, we plan to build functions for outputting circuits in Verilog and EDIF, in order to extend the link with other designing tools.

The system described is based on free software. All that is needed is a simple ASCII text editor and a Haskell interpreter like *Hugs* [1].

6. REFERENCES

[1] http://www.haskell.org/hugs/
The Haskell Interpreter Hugs

[2] K. Claessen. "*Embedded Languages for Describing and Verifying Hardware*". PhD thesis, Chalmers University of Technology and Göteborg University, Göteborg, Sweden, April 2001.

[3] Hudak et al., P., Report on the Programming Language Haskell, A Non-Strict, Purely Functional Language, Version 1.2, ACM SIGPLAN Notices, 1992. See *http://www.haskell.org/definition* for latest version

[4] S. Johnson. *Synthesis of Digital Designs from Recursion Equations*. The ACM Distinguished Dissertations Series, The MIT Press, 1984.

[5] Matthews, J. and Cook, B. and Launchbury, J., Microprocessor specication in Hawk, IEEE International Conference on Computer Languages, 1998.