Comparison of Implementations of Real Arithmetic in ELLA and VHDL

C O Newton  M G Hill

Royal Signals and Radar Establishment
Malvern  UK.

Abstract

A comparison is made between the newly available 'numerics' features of ELLA Version 4 and equivalent facilities in VHDL. The comparison is made for two particular functions, an infinite impulse response filter and an implementation of an IEEE 454 standard 32 bit arithmetic multiplier. While these designs can be readily expressed in both ELLA and VHDL, only ELLA provides a direct migration path from behaviour to structure. This is claimed to be a considerable research achievement, with attractive development prospects for synthesis and formal verification.

1 Introduction

ELLA™ [1],[2] is a hardware description language which originated at the Royal Signals and Radar Establishment (RSRE), part of the UK Ministry of Defence. VHDL [3] is the US Department of Defence standard behavioural language, supplied to RSRE on a government to government basis for evaluation. In a previous document [4], a comparison was made between VHDL and ELLA for operations on bits and integers. The latest release of ELLA (Version 4V0), which will be available in Spring 1990, offers bit vectors, operations on bit vectors and real arithmetic [5], giving a significant improvement on the capabilities of the previous version, ELLA 3V2.

The emphasis of this paper is to compare the newly available 'numerics' features of ELLA with equivalent facilities in VHDL. Few other hardware description languages with structural facilities offer floating point numbers. Thus the objective of a consistent approach to numerics for behavioural and structural expressions is a live research issue. For ELLA it is a particularly important issue because ELLA aims at the full interoperability of behavioural, functional and structural notations. In fact the goal for ELLA is still more ambitious. It is to provide a homogeneous transformation route from behaviour (to include real arithmetic) to structure (involving bits), integrating computer aided synthesis and mixed mode simulation.

2 Differences in Aim between ELLA and VHDL

ELLA originated in 1978 as a language for circuit design. A fundamental concept behind the language design is that it should provide constructs which are orthogonal. A single syntactical system covers structure, functionality and behaviour so that designs can freely mix the different formats. A simple core language was tried out and has been progressively expanded as a result of feedback from language implementation, simulator development and extended use by chip designers. Further enhancements are under consideration to provide additional facilities for synthesis, testability and analogue applications.

VHDL originated as part of the VHSIC programme in 1982. ELLA was considered as a starting point for the DoD language but was not accepted, partly because DoD wanted as much commonality as possible with ADA and partly because DoD was looking for a language to specify and archive designs in 'black box' fashion. The emphasis in VHDL has increasingly shifted away from specification in the direction of efficient black box modelling. The design of VHDL was originally allocated to a team of companies (IBM, TI, Intermetrics) and has been carried out more recently by an IEEE committee. The methodology has been to agree a complete standard without feedback from implementers or users.

3 Differences in Style between ELLA and VHDL

The aim in ELLA has been to go for maximum generality because only in this way can a single notation cover structure, functionality and behaviour. Thus in ELLA there are no predefined types and there are no predefined modules. It is up to the user to define types and modules which are consistent with each other. Sequential statements are allowed in ELLA subject to limitations which ensure that structural, functional and behavioural modules may be mixed in any combination, and that synthesis of circuits using such statements is feasible. It is consistent with this approach that the ability to use real numbers has been introduced into ELLA so as to be totally general.
It is accepted that the degree of freedom offered for real arithmetic is more than most users will want. It is further accepted that part of the purpose of behavioural simulation is to achieve both faster and easier simulation. For this reason a library of implicitly parameterisable functions is provided. These functions are not part of the language but are linked in at assembly time and provide direct access to host machine functions. It should be noted that the functions on bit strings deliver results without truncation: the user has to trim the result to the required length, and thus take a conscious decision about rounding and overflow.

In VHDL more emphasis has been put on simulation speed than on generality. User defined types coexist with predefined types which are part of the language. Several predefined datapath functions are available for use with the predefined types as part of the language. Simulation speed is given as a reason for the user employing the predefined items. Rigid demarcation lines are drawn between behavioural, datapath and structural descriptions; this is a valid method of overcoming the problem of designing a simulator to cope simultaneously with behaviour and structure. It does however place severe constraints on the user especially when the design strategy is meet-in-the-middle rather than strictly top-down or bottom-up.

The implementation of real arithmetic in VHDL is consistent with ease of simulation. A fixed set of characters has been chosen for denoting a real number and a set of host arithmetic operators is provided. The implementation is similar to that used in a programming language. There is no facility to control overflow, underflow or rounding error during a computation.

4 Comparison of Implementations of Real Arithmetic

4.1 Difference in Types

ELLA types have names and tags so that the same numerical value may be differentiated between types. Thus:

TYPE numbera = NEW na/(-100..100),
numberb = NEW nb/(-200..200),
fchara = NEW fa('0.'9'|e|l'1
| ','ji),
fcharb = NEW fb('0.'9'|e|l |b'|c'|d'|e|l|f),
floata = STRING [6] fchara,

Here na/11 and nb/11 are both of value 11 but are not interchangeable. fa"1.36Oe9" and fb"1.36Oe9" are floats, floatb character strings respectively; they look similar but the first represents a floating point decimal number, the second a fixed point hex number.

ELLA comes with a built-in library defining floating point types with 32 bits for short precision and 64 bits for long precision. An error flag detects results which cannot be represented in the defined format (i.e. overflow and underflow). Ranges and automatic range checking are not provided for these floating point numbers.

VHDL provides a typing mechanism with definable ranges for use in port definitions but there are no tags to differentiate values. Literals may be decimal literals, based literals, enumeration literals, string literals or bit string literals. The allowed characters for decimal, based and bit string literals are rigidly fixed. Thus:

type numbera is range -100 to 100;
type numberb is range -200 to 200;
type floata is range -1.0E6 to 1.0E6;
type floatb is range 16#-1.0E6E6 to 16#1.0E6E6;
type floatc is range X"0" to X"EEE";

Numbers, numberb and floata are examples of decimal literals. The value 11 is an allowed value of both numbera and numberb. floatb is an example of a based literal, using base 16; bases 2, 4, 8 are also allowed. floata is an example of a bit string literal, X referring to base 16; B for binary, O for octal are also allowed. Numerics operators such as addition and multiplication are defined for decimal and based literals but not for bit string literals.

VHDL leaves decimal precision vague, with a requirement for at least six decimal digits of precision. Calculations on types integer and real are implementation dependent. Overflow and underflow are not detected because this is stated to be difficult on some hosts. Values returned are checked against ranges where they have been declared.

4.2 Difference in Operators

The standard ELLA Version 4 library offers currently 70 built-in operators for logical and arithmetic operations on bit strings and reals. Each has to be characterised before it may be used. The characterisation may be for a particular type or for a polymorphic (generic) type. One of the extensions to ELLA to accommodate Version 4 numerics is the introduction of implicitly parameterisable functions (polymorphic macros). This enables one macro to deal with many input string lengths. For example:

TYPE bit = NEW b('0'|'1'),

# particular type #
FN AND_R = (real, real) -> real: BIOP AND.

# polymorphic type #
MAC AND_P = (STRING [INT n] bit, STRING [n] bit) -> STRING [n] bit:
BIOP AND.
Polymorphic macros for unsigned add, multiply and divide may be customised thus:

```vhdl
TYPE flag = NEW (error | ok).
```

Note how addition automatically creates an extra bit over the longer of the addends which will require deliberate truncation in a calculation. Multiplication similarly creates a longer result. Division provides a flag, quotient and remainder, with the flag set to error on divide by zero.

VHDL provides a set of logical, relational and arithmetic operators to work on bits, bit vectors, integer numbers and floating point numbers. There is no specific control of floating point rounding but input and output formatting is available.

Comparison of operators between VHDL (predefined) and ELLA (BIOP library):

- **Logical operators:**
  - common: AND, OR, NOT, XOR
  - VHDL also offers NAND, NOR.

- **Relational operators:**
  - common: =, >, >=, <, <=
  - VHDL also offers /= (not equal).

- **Arithmetic operators:**
  - common: +, -, *
  - VHDL also offers ** (exponentiation)
  - ELLA also offers SL/SR (shift left, right) and SQRT.

ELLA BIOPs work on bits, unsigned bit strings, signed bit strings, binary floating point numbers and decimal floating point numbers. ARITH is used to handle ELLA decimal and radix integers. Thus the text phrase "a*b" may in successive transformations represent i) host machine arithmetic ii) binary arithmetic with various experimental bit lengths iii) a multiplier structure with selected bit length, using optimised simulation in each case. VHDL operators work on bits, bit strings (logical operators only), and binary, octal, hex and decimal integer and real numbers. This leaves a big hole in the VHDL armoury in terms of arithmetic operations using bit strings and binary floating point numbers. The hole may be filled by laboriously writing such operators in VHDL code, which will simulate at VHDL speed. Neither ELLA nor VHDL currently offer logarithmic, trigonometric, hyperbolic or other standard functions, but these are expected to become available in ELLA and VHDL in due course.

### 4.3 Use of Unknown Value in ELLA

A universal unknown or query value was introduced into ELLA with Version 3. The advantages of query in debugging designs [6] and in using non-exhaustive case clauses to improve synthesis results [7] are described elsewhere. It may be observed that the notation for floating point numbers introduced for Version 4 is fully consistent with universal query. A bit string is not an array of bits, requiring a query value for each bit, which would rule out translation to machine bit vector arithmetic. A bit string is a single entity which has a single query flag that is flagged should any bit become query.

In VHDL there is no universal query value. Before the availability of universal query in ELLA the normal method of providing a query value for integers was the use of an associated type (variant record). VHDL does not offer variant records either. The only way to achieve an unknown value for an integer or real type in VHDL is to employ record types thereby creating a significant overhead.

### 5 Example Circuits

#### 5.1 Infinite Impulse Response Filter

It is possible to describe an infinite impulse response filter in terms of structure in VHDL or ELLA but it makes more sense to work in terms of behaviour when the multiplications involve real arithmetic. The filter descriptions are shown in Appendix 1. The filter is fixed for simplicity to use a through path and three delayed paths with arbitrarily selected coefficients. In either ELLA or VHDL a generic path structure may be used with coefficients and initial values also supplied as parameters. For reasons of brevity the VHDL description is given without the test harness required to drive the simulation. The ELLA description does not include the simulation file or the renaming of BIOPs to overload +, -,*.

The basic similarity in the ELLA and VHDL representations of the behaviour is remarkable. The vital difference is that in the ELLA +, -, * may be directly replaced by functional or structural versions of the operators. By reinterpretation of the parameters and manual rewriting, the ELLA text transforms to structure. This gives an indication of the potential of ELLA for transformational reasoning. The VHDL text is not suitable for handling in this way.

Looking at the text in more detail ELLA makes a distinction between variables which hold their value from one time instant to the next, persistent variables (PVAR), and variables which are reinitialised at each successive time instant (VAR). Persistent variables imply a data storage element. In VHDL all variables are effectively persistent. This explicit distinction again makes ELLA more amenable to synthesis.
5.2 Arithmetic for a Microprocessor

A microprocessor may be incorporated into a behavioural simulation model for one of two reasons. Either it may be a black box model to interact with the rest of the system or it may be described for implementation. In either case it is important to model the exact arithmetic, including overflow, underflow and rounding because it is at the extreme limits that anomalous operation is most likely and needs careful simulation. IEEE 754 standard arithmetic has been defined to accommodate this requirement [8]. A model based on an IEEE 754 multiplier is shown in ELLA and VHDL in Appendix 2. It is shown for clarity in 32 bit form, but generic sizing is available in both ELLA and VHDL.

The numerical errors introduced by multiplication routines are determined by how the trimming is done and in this example we have specified the accuracy of the computation of the result to within half a unit in the least significant bit. For this case two bits are needed for perfect rounding (the guard and sticky bit). The presence of a guard and sticky bit is necessary to ensure accurate unbiased rounding of the result, where the guard bit is the first bit beyond rounding precision and the sticky bit is the logical OR of all the remaining bits. This example uses normalised numbers and delivers 'Not-a-number' on underflow. For further information the reader is referred to [8].

The immediate advantage in ELLA is that the bit vector operators used are provided as BIOPs; in VHDL these have to be written by the user as VHDL code (functions bplus, bsub, bmult are not shown). Both ELLA and VHDL use declare before use, but ELLA is more flexible in that VAR and LET declarations take their type contextually from obligatory initialisation. The ELLA case clause seems to be more flexible than equivalent VHDL expressions.

6 Conclusions

The basic deduction is that real numbers and bit vectors can be handled in both ELLA and VHDL with reasonable ease. More effort has been expended in ELLA to provide a migration path from real floating point numbers through binary floating point numbers to bit vectors. This is consistent with the presupposition in VHDL that behaviour and structure are separate and the presupposition in ELLA that a direct transformation should be available between behaviour and structure. It is the provision of this transformation which enables ELLA to support synthesis and formal verification. The circuit implementations in this paper demonstrate a considerable research achievement, namely that ELLA can offer comparable real arithmetic to VHDL without sacrificing its direct link between behaviour and structure.

7 Acknowledgements

The authors are pleased to acknowledge the Alvey Advanced Information Technology Initiative, which has supported the implementation of numerics in ELLA. The authors would like to thank all members of the ELLA teams at RSRE under Mr J D Morison and at Praxis Electronic Design for their contributions.

8 References


ELLATM is a registered Trade Mark of the Secretary of State for Defence.

Appendix 2

VHDL Implementation of IEEE454 Multiplication

function times (x, y: FLOAT) return FLOAT is
    variable xfrac, yfrac, zfrac: BIT_VECTOR (1 to 24);
    variable xexp, nexp, nfrab, inx: BIT_VECTOR (1 to 10);
    variable nfrac, nfrac, fbc, ffb, ffc, i:
    constant n: INTEGER := 4;
    type BV is array (1 to n) of INTEGER;
    type STRING is array (1 to 23) of STRING;
begin
    zfrac := xfrac * yfrac;
    if nexp >= ("00" & "11111111") and
       (exp2 (2 to 9) > "01111110") then
        xfrac := ("11111111", "00000000000000000000000000000000");
    elsif nexp < ("00" & "11111111") and
       (exp2 (2 to 9) <= "01111110") then
        xfrac := ("00000000000000000000000000000000", "11111111111111111111111111111111");
    else
        xfrac := (sign, nexp, nfrab);
    end if;
end times;

end if;
end loop;

Process (input)

constant n: INTEGER := 4;
    type BV is array (1 to n) of INTEGER;
    type STRING is array (1 to 23) of STRING;
    begin
        fbc := "00000000000000000000000000000000";
        ffc := "00000000000000000000000000000000";
        for i in n-2 downto 1 loop
            fbc(i) := fbc(i+1) + fbc(i)
            end loop;
            m(n) := inputs + fbc(n);
        end process;
end.

END.

VHDL Implementation of an Infinite Impulse Response Filter

entity IIRFILTER is
    port (input : in IEEE Real;
           output: out IEEE Real);
end;

architecture behaviour of IIRFILTER is
begin
    process (input)
        constant n: INTEGER := 4;
        type BV is array (1 to n) of INTEGER;
        type STRING is array (1 to 23) of STRING;
        begin
            for i in n-2 downto 1 loop
                (m(n) := m(n+1);
                fb(i) := fb(i+1) + fb(i)
                end loop;
                m(n) := inputs + fb(n);
            end process;
        end.
end.

END.

ELLA Implementation of a Infinite Impulse Response Filter

entity IIRFILTER is
    port (input : in REAL;
           output: out REAL);
end;

architecture behaviour of IIRFILTER is
begin
    process (input)
        constant n: INTEGER := 4;
        variable m, fb, fbc, ffc, I:
        begin
            (m(n) := m(n+1);
            fb(i) := fb(i+1) + fb(i)
            end loop;
            m(n) := inputs + fb(n);
        end process;
end.

END.