IEEE 200X Fast Track Change Proposal

ID: FT10A

Proposed By: Jim Lewis jim@synthworks.com
Analyzed By: Jim Lewis jim@synthworks.com
With significant help/input from John Ries

Status: Analyzed
Proposed: 12/03
Analyzed: 07/11

Enhancement Summary:

Add nnary expressions

Revisions:

Rev4: Made nnary expression left recursive. Changed n-nary to nnary.
Fixed error in example. Added to further study section.
Rev3: Split into 10A nnary expressions and 10B sequential assignments
Rev2: Initial pdf/word release. Changed conditional_expression to ternary_expression.
Removed proposed changes to conditional waveforms due to ambiguity.
Combined conditional assignments with selected assignments.
Rev1: Initial release

Related issues:

Relevant LRM sections:

7.1

Enhancement Detail:

This enhancement is intended to bring a generalized form of conditional expressions, termed nnary expressions. In a generalized form, a nnary expression is as follows:

```
expression1  if  condition1,
expression2  if  condition2,
      •
      •
      •
expressionN-1 if  conditionN-1,
expressionN
```

Code example:

```
Y <= A if Asel='1', B if Bsel='1', C if Csel='1', D if Dsel='1', E ;
```

The syntax is deliberately selected different from conditional signal assignment to avoid ambiguity.

Use in initialization:

```
Signal A : integer := 7 if GEN_VAL = 1, 15 ;
```

Used in port range:

```
entity fifo is
    generic (word_size : Natural := 8);
    port ( data : std_logic_vector(
             (7 if word_size <= 8,
              15 if word_size <= 16, 31) downto 0) ;

    -- Added parentheses to demonstrate evaluation order:
    port ( data : std_logic_vector(
             ( 7 if word_size <= 8,
             (15 if word_size <= 16, 31)) downto 0);
```

Nnary expressions are similar in precedence to conditional signal assignment:

```
EX_1A:  Y <=  A and B  if S = '1',  C and D ;
EX_1B:  Y <= (A and B) if S = '1', (C and D) ;
```

Parentheses required due to low precedence:

```
EX_1C:  Y <= A and (B if S = '1', C) and D ;
```

Must be differentiated from both conditional signal assignment and selected signal assignment

```
EX_2:  Y <= A if S1='1', B   when S2='1' else C ;
           ^^^^^^^^^^^^^^   vvvvvvvvvvvvvvvvvvv
           nnary            conditional waveform

EX_3:  with MuxSel select
         Y <= A if Asel='1', B when '0',
              C if Csel='1', D when '1',
              'X' when others ;
```

The following two are equivalent:

```
EX4A:  AReg <=   '0' if nReset = '1', A   when rising_edge(Clk) ;
EX4B:  AReg <=  ('0' if nReset = '1', A)  when rising_edge(Clk) ;
```

A similar expression, however different implication with conditional signal assignment:

```
EX5:   AReg <= '0' when nReset = '1' else A when rising_edge(Clk) ;
```

With unaffected (future extension), similar to EX5 in semantics:

```
EX6A:   AReg <=   '0' if nReset = '1', A if rising_edge(Clk) ;
EX6B:   AReg <=   '0' if nReset = '1', (A if rising_edge(Clk)) ;
```

Use of unaffected (future extension), equivalent to EX4A and EX4B:

```
EX7:    AReg <=  ('0' if nReset = '1', A) if rising_edge(Clk) ;
```

Analysis

The goal of this proposal is to find a solution that maximizes the following conditions:

1. Work in sequential and concurrent context
2. Work in initializations and general expressions
3. Each condition can test different signals
4. The new syntax cannot conflict with existing syntax
5. Be concise but also maintain the nature and spirit of VHDL
6. No parentheses

First the following syntax for nnary expression was considered. This syntax requires parentheses in many contexts (selected signal assignment and conditional signal assignment). It has been rejected due to its similarity to conditional signal assignment, it appears not to have parentheses in some situations.

```
expression1  when  condition1 else
expression2  when  condition2 else
     •
     •
     •
expressionN-1 when conditionN-1 else
expressionN
```

The following sets of syntax were also considered. The first one selected was selected due to its conciseness while maintaining readability:

Form1 (accepted):

```
expression1  if  condition1,
expression2  if  condition2,
     •
     •
     •
expressionN-1 if conditionN-1,
expressionN
```

Form2 (rejected: else does not provide additional benefit over ","):

```
expression1  if  condition1 else
expression2  if  condition2 else
     •
     •
     •
expressionN-1 if conditionN-1 else
expressionN
```

Form3 (rejected: ? has reduced readability and not enough benefit over if):

```
expression1   ?  condition1,
expression2   ?  condition2,
     •
     •
     •
expressionN-1 ? conditionN-1,
expressionN
```

Approach Overview:

In section 7 added nnary expression. Since nnary expressions have lower precedence than logical operators, they need to be first in the BNF. Nesting of nnary expressions occurs through a primary having a choice of "(expression)".

Add section 7.6 that defines nnary expressions and how they are evaluated. Note that nnary expressions are not operators and hence are not overloaded. Their result value is a function of the result value of the enclosed expressions.

Further Study

Can unaffected be used in nnary expressions? How would constants and inputs to subprograms be handled?

```
nnary_expression ::=
    { logical if condition, }
    logical  [if condition]
```

From Jim: This means we would have to deal with the dangling else given that condition is allowed to contain a nnary expression.

Is "A if B if C, D" the same as "A if (B if C, D) " or is it "A if (B if C), D"? Other languages seem to use the first one.

From John Ries:
If unaffected is a primary then we need to define how it works in all cases. In a number of cases this is very unclear. For example.

a(3 downto 0) := s1 & s2 & s3 & ('1' if (cntrl = '1'), unaffected);

Does this mean that the value of a(0) is unchanged? Lets make it slightly more complicated

integer_variable := convert_to_integer(s1 & s2 & s3 & ('1' if (cntrl = '1'), unaffected));

What gets passes to convert_to_integer? What is the result?

LRM Changes

Changes to Clause 7.1

Modify the BNF in section 7.1 as follows:

```
expression ::=  nnary_expression

nnary_expression ::=
  { logical if condition, } logical

logical ::=
  relation { and relation }
  | relation { or relation }
  | relation { xor relation }
  | relation [ nand relation ]
  | relation [ nor relation ]
```

```
      | relation { xnor relation }
```

Add new Clause 7.6

7.6 Nnary Expression

A nnary expression is an expression that selects a result value from one of the enclosed expressions. To select a result value, each condition is evaluated in succession until one evaluates to TRUE. The expression preceding the TRUE condition is evaluated and its result value is the result value of the nnary expression. If each condition evaluates to FALSE the last expression is evaluated and its value is the result value of the nnary expression.

Note: A nnary expression is not an operator or subprogram and it cannot be overloaded.

Examples:

Use of a nnary expression to initialize a signal:

```
Signal A : integer := 7 if GEN_VAL = 1, 15 ;
```

Use of a nnary expression to constrain the size of an array:

```
entity fifo is
    generic (word_size : Natural := 8);
    port ( data : std_logic_vector(
              ( 7 if word_size <= 8,
               15 if word_size <= 16, 31) downto 0) ;

    -- Added parentheses to demonstrate evaluation order:
    port ( data : std_logic_vector(
              ( 7 if word_size <= 8,
              (15 if word_size <= 16, 31)) downto 0);
```

Nnary expressions allow a richer set of expressions than currently available in conditional signal assignment:

```
Y <= A and (B if S = '1', C) and D ;
Y <= ('1' if en = '1', 'Z') after 5 ns ;
```

Editing note for 7.6

If nnary expressions are expanded as suggested in Further Study, the following text replaces the last sentence of the first paragraph in clause 7.6

If all of the conditions evaluate to FALSE and there is an expression following the last condition, this expression is evaluated and its result value is the result value of the nnary expression. If all of the conditions evaluate to FALSE and there is not an expression following the last condition, then the result value of the nnary expression is unaffected. It is an error for the expression to result in unaffected where a value is required.