VHDL BOOKS: TRAINING GROUNDS OR GROUNDMINES?

Joseph Pick
COMPASS Design Automation
5457 Twin Knolls Road
Columbia, Maryland 21045
(410) 992-5700

ABSTRACT

VHDL users at all levels of expertise rely on textbooks to enhance their coding techniques and to improve their understanding of this hardware description language. Unfortunately, many of these books contain, and hence perpetuate, a multitude of inefficient programming styles, false statements, and textual bloopers. Inefficient coding styles will needlessly slow down the model’s simulation. False statements about VHDL will result in lengthy and frustrating debugging sessions. Textual bloopers will confuse and disorient the reader. This paper presents a wide range of examples from the VHDL literature that are potentially detrimental to the reader’s growth as VHDL designers and engineers.

1. Introduction

"The more you know the happier and easier life will be". Applying this idiom to VHDL means that you should endeavor to be an avid reader of the VHDL literature. By so doing your knowledge of this subject should be enhanced. Moreover, there should be a marked improvement in your on-the-job modelling productivity. Unfortunately, the reality of this situation is not as idealistic and straightforward as one would like it to be. Instead, many of the current VHDL books provide a plethora of misinformation that may unintentionally hinder the reader's growth as a VHDL designer and engineer. These educational groundmines tend to fall into at least one of the following categories:

- Inefficient Coding Styles.
- Questionable / Poor Coding Styles.
- Unfulfilled Promises.
- Inaccurate Depiction of VHDL.
- "Welcome to the Twilight Zone".

The remainder of this paper will present numerous examples from each of these categories. In order to maintain a neutral, objective stance, this paper will not identify the owners of these infractions. All code segments will be written exactly as they appear in the respective books.

2. Inefficient Coding Styles

The problem with coding inefficiencies is that the reader will unwittingly perpetuate a modelling style that will needlessly prolong the model's simulation time. The list below highlights a sample of those inefficiencies that occur most frequently in the VHDL literature.

**Objection Raised**
A common inefficiency is the unnecessary re-computation of the same value.
An example of a wasteful computation is the following code segment:
wait for 2 * PERIOD;

On each iteration of the enclosing process the product, 2 * PERIOD, will always be evaluated. But in the author’s given example PERIOD is a generic parameter and hence the expression, 2 * PERIOD, will always have the same value. Consequently, there really is no need to re-compute it over and over again.

**Recommended Optimization**
Compute the expression just once, assign it to a constant, and then use this constant in the body of your code.

constant TWICE_PERIOD : TIME := 2 * PERIOD;

......................

wait for TWICE_PERIOD;

**Objection Raised**
The author presents a function body having a constant look-up matrix in its declaration region. This approach is very inefficient because every time the function is called this look-up matrix will dynamically be recreated on the stack of the host CPU that is executing the simulation.

**Recommended Optimization**
The constant look-up matrix should be placed outside of the function declaration region. There are several options for its location depending on the design unit that is enclosing the function body. If the function body is contained in a package body then just place the constant declaration anywhere prior to the function body’s source code. If the function body is in the declarative region of any design unit such as an architecture or process, then place the constant matrix in the same declaration region but before the function body. Note that in all cases the constant matrix has to precede the function body so that the matrix’s name will be visible to the function body.

**Objection Raised**
Typically authors use the expression 2 ** N to convert from BIT to INTEGER, where N is the for loop parameter. This exponential operation is very expensive in terms of CPU time.

**Recommended Optimization**
The 2 ** N algorithm should be replaced with the following computation:

```
Converted_Value := Converted_Value + Converted_Value +
BIT'POS(Input(N));
```

Note that this optimization further assumes that the compiler might not be smart enough to convert a multiplication by two (2 * Converted_Value) into a shift left operation. So instead, the variable, Converted_Value, is added to itself, which is a highly efficient operation requiring only one CPU cycle.

**Objection Raised**
Typically authors use a combination of division and the rem operator to convert from INTEGER to BIT. Both these binary operations are very expensive in terms of CPU time.

**Recommended Optimization**
This conversion should be implemented via the combination of a subtraction operation and a table look-up into an array containing the powers of two.

**Objection Raised**
Many authors use the expression (not S'\text{STABLE}) in a sequential region to check for an event on the signal S. This method is very inefficient since S'\text{STABLE} is a signal and hence must be queued, monitored, and managed by the VHDL simulation engine.

**Recommended Optimization**
S'\text{EVENT} should be used in a sequential region since it is not a signal and its value is always equivalent to (not S'\text{STABLE}).

**Objection Raised**
The assert statement below is another flavor of the unnecessary re-computation of the same value. In this variation the function \text{NOW} is called repeatedly, even though the returned value has not changed in the interim between calls. A VHDL compiler might not be smart enough to optimize this scenario.

\[
\text{assert (NOW = 0 NS) or (NOW - R\_LAST\_EVT) > SPIKE\_WIDTH}
\]

**Recommended Optimization**
A temporary variable should be used to capture the output of this function. This variable may then be used repeatedly until the output of this function changes. For example, the given assert statement should be re-written as follows:

\[
\text{Now\_Value := NOW;}
\text{assert (Now\_Value = 0 NS) or (Now\_Value - R\_LAST\_EVT) > SPIKE\_WIDTH}
\]

**Objection Raised**
Another very common inefficiency in the VHDL literature is to implement a vector manipulating function by repeatedly calling its scalar version from within a for loop. The overhead of calling a function for each element of an array is very costly in terms of CPU time. An example of this inefficiency is the following for loop in which the scalar function called \text{wire} is repeatedly invoked:

\[
\text{FOR } i \text{ IN drivers'\text{RANGE LOOP}}
\text{ accumulate := wire (accumulate, drivers(i));}
\text{END LOOP;}
\]

**Recommended Optimization**
A look-up table should be used instead. Here again, this constant matrix table should not be defined within the function's declaration region.

\[
\text{FOR } i \text{ IN drivers'\text{RANGE LOOP}}
\text{ accumulate := WIRE\_CONSTANT\_MATRIX (accumulate, drivers(i));}
\text{END LOOP;}
\]

3. **Questionable / Poor Coding Styles**

The problem with questionable and poor coding styles is that the reader will unwittingly perpetuate a modelling technique that is either not necessary or may, under certain circumstances, produce unexpected side effects. The list below highlights a sample of questionable code segments that exist in the current VHDL literature.
Objection Raised
Because of readability and maintainability concerns it is not appropriate to assign a mode of inout to a port just so that the port's name may appear on both sides of a signal assignment statement. The mode inout should only be used in a VHDL model if the corresponding hardware pin is bi-directional. The following code segment is an example of this misuse. Incidentally, this example also belongs in the Twilight Zone category since the underscore is missing between the "rs" and "flop".

definition:
entity rs flop is
port(set, reset : in bit;
    q, qbar : inout bit);
end rs flop;

**Recommended VHDL Coding Style**
Declare the ports q and qbar to be of mode buffer. Another option is to declare the ports to be of mode out and then use intermediate signals to connect them. This latter approach is the more practical one since VHDL has the syntactical restriction that ports of mode buffer cannot be associated with an actual designator that is a port of mode out.

Objection Raised
This next example of poor coding style did not appear in a VHDL textbook. Instead it occurred in one of many VHDL conference proceedings which is even more dismaying. VHDL proceedings should only contain the apex of VHDL research and coding techniques. The following code excerpt is a disaster waiting to occur:

```vhdl
wait for 0 ns;
wait for 0 ns;
```

Every once in a while it might be necessary to suspend a process until the next simulation cycle with a `wait for 0 ns` statement. Unfortunately, when several of these statements appear consecutively, it is a sad indication that the VHDL modeler is counting delta delay units. The problem with this approach is that the resulting model is not easily extensible. Whenever any modification is made to the code, the VHDL engineer must then recount the number of deltas in order to determine if any `wait for 0 ns` statements should subsequently be included or deleted. Not only is this approach tedious to maintain it is also very error prone.

**Recommended VHDL Coding Style**
The overall data flow sequence of this model should be re-evaluated and re-designed so that the process interactions will not require the VHDL modeler to count delta delay units.

Objection Raised
This next example also comes from the proceedings of a VHDL conference. The author's objective is to interchange the values of two signals. However the method used is obscure and seems to indicate that the author is unaware of the fundamental characteristics of the signal assignment statement.

```vhdl
temp := list(j);
list(j) <= list(i);
list(i) <= temp;
wait for 0 ns;
```

This use of a temporary, intermediate variable mimics the technique required to interchange two variables. However because signal updates are always
scheduled to occur at a later time, it is more appropriate to write the following code sequence:

\[
\begin{align*}
\text{list}(j) & \leq \text{list}(i); \\
\text{list}(i) & \leq \text{list}(j); \\
\text{wait for } 0 \text{ ns;}
\end{align*}
\]

The `wait for 0 ns` is still needed because the algorithm implemented by this process requires the updated values of this array named `list`. But there still is another problem. Upon completing the algorithm, the process then has a wait on `X` statement. This is potentially dangerous since `X` may have had an event during the time that the process was suspended via the `wait for 0 ns` statement.

**Recommended VHDL Coding Style**

List should be declared as a variable instead of a signal. The interchanging of the `i`th and `j`th element of the array would now legitimately require an intermediate temporary variable. The resulting model would also be more efficient due to the usage of variables instead of signals. Also the enclosing process would no longer have to be suspended and reactivated for only one delta time unit.

**Objection Raised**

`return State(1);`

The author wrote the above statement assuming that the enclosing resolution function is always called with an array length of one. However such an approach is very risky and unportable. There is no guarantee that the actual parameter associated with the formal unconstrained parameter `State` will always have the value 1 as its only index.

**Recommended VHDL Coding Style**

The following portable statement should be used instead:

`return State(State'LEFT);`

**Objection Raised**

The author wrote the following use clause even though it was not required:

`use work.all;`

Because of the special role of the library called `work`, the reader might erroneously deduce that this use clause is required by every VHDL model.

**Recommended VHDL Coding Style**

In general, use clauses should only be written when required.

**Objection Raised**

`MAR <= B'00000000" after MAR_DELAY;`

Upon seeing this the reader might erroneously deduce that the base B is always required which is not the case.

**Recommended VHDL Coding Style**

The base B may be omitted. Incidentally, note that the base is required when underscores are used for readability as in:

`MAR <= B'0000_0101" after MAR_DELAY;`
Objection Raised
MIR <= B"00000000_00000000_00000000_00000000";

It is very time consuming to write out thirty-two zeroes.

**Recommended VHDL Coding Style**
It would be easier to write

MIR <= (others => '0');

Incidentally, MIR is supposed to represent a microinstruction register. The author’s model would be more manageable and less error prone, if the various fields of this register would be aliased. Under this aliasing method the various fields would be distinguishable via a meaningful mnemonic. Moreover it would no longer be necessary to remember the indices that correspond to each MIR field.

Objection Raised
process (CLK)
begin
if CLK = '1' then
  case STATE is
  ..........  
end case;
end process;

This code is intended to model a state machine that transitions on the leading edge of the signal CLK. Unfortunately, the given approach is very risky. What if CLK has an initial value of '1'? Under this circumstance, the state machine will be updated one delta after the VHDL initialization phase even though the first leading edge of CLK has not yet occurred. All subsequent state machine updates will be handled correctly even though the first one will not agree with the hardware.

**Recommended VHDL Coding Style**
Either of the following two techniques will correct this potential error:

Technique_1:
process (CLK)
begin
  if CLK = '1' and CLKEVENT then
    case STATE is
    .......... 
  end case;
end process Technique_1;

Technique_2:
process
begin
  wait until CLK = '1'
  case STATE is
  .......... 
end case;
end process Technique_2;

Objection Raised
S <= A and B;
NS <= not S'DRIVING_VALUE;

This code is intended to illustrate a new VHDL'92 feature. Unfortunately, naming a signal NS may lead to disastrous results because this user defined identifier will now be visible instead of the unit ns from the STANDARD package. If the next line would be
wait for 10 ns;
then a compile error would be generated since ns is no longer interpreted by the compiler as a timing unit.

**Recommended VHDL Coding Style**
Never use a name from any of the pre-defined VHDL packages.

**Objection Raised**
Consider the following code segment of a function where the identifier VEC is one of its formal input parameters:

```vhdl
if VEC'right <= VEC'left then
    for N in VEC'right to VEC'left loop
        ...
    end loop;
else
    for N in VEC'right downto VEC'left loop
        ...
    end loop;
end if;
```

A VHDL novice reading this code would conjecture that it is always necessary to deduce the direction of a function's input arrays and then repeat any for loops but with different range directions. Besides being difficult to understand, such an approach is also inefficient in terms of both execution speed and memory utilization. Perhaps the problem is that the author's VHDL tool suite does not currently support the alias construct. This non-compliance with the VHDL-1076 IEEE standard may have forced the author to use this poor coding scenario.

**Recommended VHDL Coding Style**
An aliasing technique should be used instead which will allow the direction of the for loop bounds to be controlled by the VHDL modeller as in the following coding excerpt:

```vhdl
alias Vec_Alias : BIT_VECTOR (VEC'LENGTH downto 1) is VEC;
        ...
for N in 1 to  Vec_Alias'LEFT loop
```

4. **Unfulfilled Promises**

A book's jacket and table of contents establish a Contract of Trust between the author and the reader. The author is professionally bound not to betray this trust. I will not delve too much on this matter since there really is only one book that does not fulfill its advertised promises. This book contains chapters on the following topics: Microprocessors, Control Units, Caches, Memory Management, Pipelines and Superscalar Architectures, Bus Interface Units, Busses and Protocols, and Applications of Finite State Machines to Microprocessor Control Units, Memory Control Units, Bus Interface Units and Protocols. These headings sound great and being in a VHDL book one would naturally expect these topics to be accompanied by VHDL code illustrating how to implement these computer subsystems. Unfortunately, these chapters contain only a very shallow preamble for these topics. There is no supplementing VHDL code. In many cases, the reader is already well versed in these subjects and so does not need a superficial introduction to them. The VHDL reader is expecting to see lots of VHDL code! But even more shocking is that the book's title and cover jacket imply that the book
"uses a code example-based approach which illustrates principles as well as advanced
features of VHDL - using real-world problems." Perhaps the author sincerely intended
to fulfill these promises. Unfortunately something went wrong because the final
product does not even come close.

5. Inaccurate Depiction of VHDL

Erroneous VHDL related statements have a disastrous, far reaching effect. The
perpetuation of VHDL falsehoods will result in a model that will behave in an
unexpected manner. Either the model will fail to compile or its response to injected
stimuli will be contrary to the reader's expectations. Much time will then be wasted
trying to resolve this discrepancy between the model's behavior and the book's alleged
VHDL dogma. The reader will not know whether to believe the book on this matter or to
blame his/her VHDL tool suite. The reader might even ask the VHDL tool builder to
support this erroneous feature. This section presents a sample of inaccurate statements
regarding the VHDL language. In some cases the author is clearly in error, whereas in
others the author's mistaken beliefs may have originated with his VHDL tool suite not
being fully compliant with the VHDL-1076 IEEE standard.

Objection Raised: Author is Incorrect
Author is somewhat confused about the relationship between component
declarations, instantiations, and binding. He states that the corresponding
design entity must exist at the time that the component is instantiated. This
confusion is a running theme throughout his book.

VHDL Fact
At the time of component instantiation the eventual, corresponding
design entity does not yet have to exist as a compiled unit in a VHDL
resource library.

Objection Raised: Author's VHDL Tool Suite May Be Incorrect
Author believes that the following code is correct since it was probably accepted
by his VHDL tool suite:

C := BIT_VECTOR('1', B, others => '0');

VHDL Fact
BIT_VECTOR is an unconstrained type and the reserved word others may
not appear in any aggregate of an unconstrained type.

Objection Raised: Author is Incorrect
Author claims that "A qualified expression can cast the type of a literal"

VHDL Fact
Author is misusing the term "casting" which implies type conversion. A
type qualification does not convert from one data type to another. Rather,
a type qualification assists the compiler in resolving the type of a VHDL
expression that would otherwise be ambiguous.

Objection Raised: Author is Incorrect
When discussing signal updating, the author writes "Hence, they are delayed
(even if zero delay is specified) until a wait is executed."

VHDL Fact
Consider the following code segment:
A <= B after 80 ns;
wait for 10 ns;

Note that signal A will not be updated during this particular wait since the signal is scheduled to be updated well after the duration of the 10 ns waiting interval.

**Objection Raised: Author is Incorrect**
Author states that the term sensitivity list only refers to processes.

**VHDL Fact**
The author is confused about the terms sensitivity list and static sensitivity list. The latter directly corresponds to a process. But other VHDL constructs may have sensitivity lists. The wait statement is one such construct.

**Objection Raised: Author is Incorrect**
While discussing the characteristics of concurrent procedures, the author writes that a concurrent call has more than one return value.

**VHDL Fact**
Though the intentions are good the words are not quite right. Only functions return a value. Procedures may modify those parameters that are of mode out or inout.

**Objection Raised: Author is Incorrect**
The author erroneously uses the following example to illustrate the process that is equivalent to a concurrent procedure named vector_to_int:

```vhdl
architecture
process (bitstuff, number)
begin
  vector_to_int(bitstuff, number)
end process;
end
```

**VHDL Fact**
A process having a static sensitivity list cannot also contain an explicit wait statement. Since the procedure `vector_to_int` may contain a wait statement it is incorrect to embed it inside a process having a static sensitivity list. Hence, the author should have written the following instead:

```vhdl
process
begin
  vector_to_int(bitstuff, number);
  wait on bitstuff, number;
end process;
```

Incidentally, the original code excerpt is also a prime candidate for the Twilight Zone section of this paper since the architecture declaration is incomplete and both the procedure call and the final end delimiter must be followed by a semi-colon.

**Objection Raised: Author's VHDL Tool Suite May Be Incorrect**
Author declares VCC to be a constant and then proceeds to use it in the following port map association:
port map (A => A, B => VCC, Z => Z);

Author's error may be due to the fact that this code may have been accepted by his VHDL tool suite:

**VHDL Fact**
VHDL'87 requires that a port actual must be a signal. The author's given method will be valid in VHDL'92.

**Objection Raised: Author is Incorrect**
Author repeatedly states that functions can have a wait statement.

**VHDL Fact**
Functions cannot have a wait statement.

**Objection Raised: Author is Incorrect**
Author presents a process without a wait statement.

**VHDL Fact**
Processes must have a wait statement.

**Objection Raised: Author is Incorrect**
Author claims that a resolution function will be called when any of its drivers has an event.

**VHDL Fact**
Resolution function will be called whenever any of its drivers has a transaction.

**Objection Raised: Author is Incorrect**
Author claims that Enable is also in the sensitivity list associated with the statement

wait on A, B until Enable = '1';

**VHDL Fact**
The wait on construct explicitly defines the members of the sensitivity list. In this particular example they are the signals A and B.

**Objection Raised: Author is Incorrect**
Author believes that the null transaction may be assigned to any signal.

**VHDL Fact**
Null transaction may be assigned only to guarded signals.

**Objection Raised: Author is Incorrect**
Author writes "Signals are of two types, bus and register, the former being the default type."

**VHDL Fact**
Bus and register are signal kinds and there is no default.

**Objection Raised: Author is Incorrect**
Author presents the following code segment:

```vhd
type STATE is (Set, Reset);
architecture Behave_Latch of Latch is
```
**VHDL Fact**
The location of the type declaration is illegal.

**Objection Raised: Author is Incorrect**
Author writes, "Note that we have given each component instantiation a unique label. This is optional".

**VHDL Fact**
Component instantiations must be labelled. There is no option.

**Objection Raised: Author is Incorrect**
Author writes, "Here we have a block and optional label".

**VHDL Fact**
Blocks must always be labelled. There is no option.

**Objection Raised: Author is Incorrect**
Author presents the following type declaration:

```vhdl
type Opcode is INTEGER range 0 to 255;
```

**VHDL Fact**
The correct type declaration is

```vhdl
type Opcode is range 0 to 255;
```

**Objection Raised: Author is Incorrect**
Author presents the following subtype declaration:

```vhdl
subtype Opcode of INTEGER is range 0 to 255;
```

**VHDL Fact**
The correct subtype declaration is

```vhdl
subtype Opcode is INTEGER range 0 to 255;
```

**Objection Raised: Author is Incorrect**
Author claims that one can conveniently convert from Big_Endian to Little_Endian by applying the following aliasing technique:

```vhdl
variable XDRInteger : BYTE_VECTOR(0 to 3);
alias LEinteger : BYTE_VECTOR is XDR(3 downto 0);
```

**VHDL Fact**
First of all, this excerpt is also a good candidate for the Twilight Zone since XDRInteger is declared but only the identifier XDR is later referenced. But there is an even more serious flaw here. The author mistakenly believes that an array object can be declared with an ascending direction and then later referenced with a descending slice. This assumption is completely false.

**Objection Raised: Author is Incorrect**
Author claims that:

S'VALID (0 NS) = not S'ACTIVE

**VHDL Fact**
S'VALID (0 NS) = not S'EVENT
Objection Raised: Author's VHDL Tool Suite May Be Incorrect
Author believes that the following code is correct since it was probably accepted
by his VHDL tool suite:

case N1 & N0 & M1 & M0 is
  when '0000' =>

  VHDL Fact
  Though a case selector may reference a function the returned value cannot
  be an unconstrained array. But the concatenation function returns a
  BIT_VECTOR type which is unconstrained. Note that the author is really
  aware of the correct VHDL syntax since he had earlier written the
  following case selector statement where TWOBiT is a constrained subtype:

  case TWOBiT ' (currentstate & nextstate)

Objection Raised: Author is Incorrect
Author gives the following code sequence as an example of an incomplete
declaration:

TYPE element_ptr;

TYPE element_rec IS
  RECORD
    data : data_type;
    nxt : element_ptr;
  END RECORD;

TYPE element_ptr IS ACCESS element_rec;

  VHDL Fact
  The author should have written the following correct sequence:

  TYPE element_rec;

  TYPE element_ptr IS ACCESS element_rec;

  TYPE element_rec IS
    RECORD
      data : data_type;
      nxt : element_ptr;
    END RECORD;

Objection Raised: Author is Incorrect
The author claims that an edge triggered device may be modelled by using the
attribute 'EVENT' in the GUARD expression associated with a block construct. In
particular he wrote:

block (clocking = '1' and clocking'EVEN'T)

  VHDL Fact
  Actually many authors are guilty of perpetuating this error. Since
  clocking'EVEN'T is not a signal it will not be in the sensitivity list for this
  block's GUARD. So even though clocking'EVEN'T becomes FALSE one delta
  after clocking transitions to '1', this change will not initiate the updating
  of GUARD. Hence GUARD will remain TRUE and the model will behave
  like a latch. To achieve the desired edge triggered effect you must write:
block (clocking = '1 and (not clocking'STABLE'))

**Objection Raised: Author is Incorrect**
The author claims that the following `concurrent` assert statement may be used to identify a hold violation:

```vhdl
assert NOT ((data'EVENT) AND (clock = '1) AND (NOT clock'STABLE(hold_time)))
```

**VHDL Fact**
This error is another variant of the previous error in which 'STABLE must be used in the GUARD expression to model an edge triggered device. Analogously, since data'EVENT is not a signal it will not cause the `concurrent` assert statement to be awaken whereas data'STABLE would. Hence the correct `concurrent` assert statement should be:

```vhdl
assert NOT ((not data'STABLE) AND (clock = '1) AND (NOT clock'STABLE(hold_time)))
```

**Objection Raised: Author's VHDL Tool Suite May Be Incorrect**
The author believes that the following code is correct since it was probably accepted by his VHDL tool suite:

```vhdl
type RESOLVED_BIT is WIRED_OR_BIT;
```

**VHDL Fact**
Only a subtype declaration may reference a resolution function. Hence the reserved word "type" should be replaced by "subtype".

6. "Welcome to the Twilight Zone"

This section describes those textual bloopers that might confuse and disorient the novice VHDL user. Most of these examples may easily have been avoided by a thorough proof-reading session.

**Objection Raised**
The author presents a procedure called vector_to_int which has an unconstrained input parameter. However the body of this subprogram contains a for loop with a loop parameter that is specified to span the range 1 to 8. This fixed loop bound is incompatible with the unconstrained input. One would have expected something like Input'LENGTH. Moreover, the procedure's name is a misnomer since the given algorithm will only yield natural numbers. Negative integers will never be produced.

**Objection Raised**
The following poorly constructed English statement implies the existence of a return statement. "In Figure 4-2, is return 'x' ambiguous type?" Unfortunately, the illustrated function did not contain a return statement.

**Objection Raised**
Another example of a questionable remark is "When you declare a BIT_VECTOR, signal, or variable, you must specify an index constraint". The comma after BIT_VECTOR is very confusing and misleading.

**Objection Raised**
The author writes (muxval <= muxval + 1;) but then later refers to the statement (muxval <= 1;).
Objection Raised
Author writes "The variable a could have been tested in the CASE statement". But there is no variable named a.

Objection Raised
Author declares a port called New, but in the corresponding architecture body uses NEWS. Clearly, the compiler will not accept this misspelling. Had this object been spelled correctly as NEW, I would have still raised two objections. Because VHDL is case insensitive the compiler would treat both New and NEW as the same object. Nonetheless the reader will be somewhat disoriented by the fact that they are not visually identical. The VHDL modeler should strive to ensure that all identifiers have a consistent appearance. Another complaint with this code excerpt is that New is a VHDL reserved word and hence should not be used as the identifier of an object.

Objection Raised
The next code segment contains many humorous bloopers. The included comments are mine and they indicate what the code should have been.

Entity BCD_DECODER is
port (D: in BIT_VECTOR (3 downto 0);
    NINE, EIGHT, ONE, ZERO : out BOOLEAN);
end BCD_DECODER -- ***** MISSING SEMI-COLON *****
architecture MY_FIRST of BCD_DECODER is
begin
    NINE <= (D = "1001");
    EIGHT <= (D = "0001"); -- ***** SHOULD HAVE BEEN "1000" *****
    ONE <= (D = "1000"); -- ***** SHOULD HAVE BEEN "0001" *****
    ZERO <= (D = "0000");
end BCD_DECODER; -- ***** SHOULD HAVE BEEN end MY_FIRST; *****

Objection Raised
Here is another blooper that unfortunately speaks for itself.

U1 : nand2
port map (a => set, c => q_; -- ***** SEMI-COLON NOT REQUIRED *****
    b => qbar);

Objection Raised
Yet another blooper is the following code excerpt.

ENTITY Linear is
    generic (m,B:real);
    port (W: in real, Y : out real); -- ***** SEMI-COLON REQUIRED BEFORE Y *****
end Linear;

architecture only of Linear is
Function c_to_f (W:real) return real is -- *** OMIT SEMI-COLON AFTER real ***
begin
    Return (M *W) + B; -- ***** WHY USE UPPER CASE M NOW ? *****
end c_to_f;
-- ***** THE RESERVED WORD BEGIN IS OMITTED. *****
end Linear; -- ***** SHOULD HAVE BEEN only INSTEAD OF Linear *****

Objection Raised
Author writes

Use my_defall; -- ***** SHOULD HAVE BEEN Use my_defall; *****
**Objection Raised**
In the following code segment the second commented line should also contain two leading dashes. It seems that the author's original one line comment mysteriously got wrapped around to a second line:

```
-- it is illegal to have both R(eset) and S(et) inputs true or '1'
```

Also, the statement "true or '1'" is misleading since these signals can only be of type BOOLEAN or of type BIT. They cannot be both.

**Objection Raised**
Author presents a VHDL model for a latch and then states that "the latch instantly sets the values of the outputs, without any propagation delays". However, the latch is implemented with signals and hence can never be updated immediately.

**Objection Raised**
variable Cray_Word : Word (1 : 64);

The author should have written (1 to 64) instead.

**Objection Raised**
Author includes 'X' as a member of an enumeration list but then later references 'x' in his coding example. Note well that character literal 'X' is not the same as 'x'.

**Objection Raised**
On numerous occasions, the author's word processing control codes ended up in the final book. Pity the VHDL novice who comes across the following:

```
@CODE = Sig <= transport WavForm1;
```

**Objection Raised**
In one book, several paragraphs were totally out of order. For instance, the section titled CONSTANTS contained the following passage:

"For each access type defined, the function Deallocate() is implicitly defined for this type."

Clearly, this statement about access types does not belong in the CONSTANTS section of the book.

**Objection Raised**
Author writes "returns a BOOLEAN value TRUE ..... otherwise zero". This sentence should have been "otherwise FALSE".

**Objection Raised**
Author writes "NOW - S'LAST_EVENT will give us the duration for which the signal has been stable". Actually S'LAST_EVENT is sufficient since it already is the desired time interval.

**Objection Raised**
In the following code excerpt the author references a resolution function's name before it is visible:

```
signal State : Sresolve State_Values register := State1;
function Sresolve (State : State_Vector) return State_Values is
```
Objection Raised
Author first makes the following declarations:

```plaintext
type State_Values is (State1, State2, State3);
signal State : Sresolve State_Values register := State1;
Author then uses the signal State in the following erroneous manner:

Block (State = 1 and Input = 1)
The comparison of State to 1 is illegal since these two objects are not of the same type.
```

Objection Raised
Author mysteriously writes:

```plaintext
B?type Out_Values is (Out1, Out2, Out3);
The characters 'B' and '?' are totally inappropriate.
```

Objection Raised
Author writes Process (Input) but in this process' body he erroneously references the signal In instead of Input.

Objection Raised
Author writes the following function:
```plaintext
function Int_Bit(BitArray: BIT_VECTOR) return INTEGER is
variable Sum : Integer :=0, Index; begin for
Index in Bit_Array'LENGTH downto 1 loop
    Sum := Sum + BitArray(Index);
    Value := Value *2;
end loop;
end Int_Bit;
```

This code is very obscure. What will happen if a poor unsuspecting VHDL novice attempts to incorporate this alleged conversion function into his/her modelling efforts? To begin with this code will not even pass the compilation phase. Specifically, the following line will be rejected by the VHDL compiler:

```plaintext
variable Sum : Integer :=0, Index; begin for
```

Moreover the general approach of this function is not safe at all. An alias should be used since there is no guarantee that the input array actually can be indexed by the range Bit_Array'LENGTH downto 1. Another problem is that the textual layout of this source code is visually very confusing. The textual positioning of many of the reserved words do not honor their semantical role within the framework of the VHDL language. For instance, the reserved word "begin" should be by itself on its own line and the reserved word "for" should be on the same line in which the range for the looping parameter is specified.

Objection Raised
Author writes the following function:
```plaintext
function Bit_Int(Int, Length : Integer)
return BitArray is
variable Sum : BIT_VECTOR;
variable Bit : BOOLEAN;
begin for Index in 1 to Length loop
    Bit(Index) = Int and 1; Int := Int/2; -- or
```
Int SRA 1 in VHDL-92, or shift up '1'
    If Int = 0 then exit;
end loop;
    return Value;
end Bit_Int;

This source code is very obscure. It definitely will not achieve the author's intended conversion. Furthermore, the code will not even compile. Here are the problems that I noted before I, along with the compiler, finally gave up:

- Visually very unappealing and confusing.
- Where did the returned object called Value come from?
- The variable named Bit will hide the type Bit that is declared in the STANDARD package.
- Comments continue onto a new line without the necessary two dashes.
- Based on the author's type declarations the following statement is totally absurd:

  Bit(Index) = Int and 1;

Objection Raised

Author references a two dimensional type, called STRN_MVL7TABLE, without first showing its declaration. Perhaps, this omission was intentional so that the author could bypass the explanation of how an enumeration literal may be used to index into an array. In any event the following type declaration should have been included along with an accompanying explanation of its interpretation:

  type STRN_MVL7TABLE is array (STRENGTH, MVL7) of MVL7;

Objection Raised

Author defines an enumeration type called CONTROL_EVENT but then uses EVENT as a type-mark (VHDL terminology) in an object declaration. Incidentally, this mistake is also from the proceedings of a VHDL conference.

Objection Raised

On several occasions, the author's word processing control codes ended up in the final book. Imagine the VHDL novice's confusion upon studying the following code segment:

```vhdl
ENTITY shifter_unit IS
    PORT (alu_side : IN byte; arith_shift_left, arith_shift_right : IN qit;
        in_flags : IN nibble; obus\kern1pt_side : OUT byte;
        out\kern1pt_flags : OUT nibble)
END shifter\kern1pt_unit;
```

Objection Raised

Author used the following code segment to highlight that the component name does not hide the entity of the same name:

```vhdl
entity AND2 is
    port (X, Y : in Bit; Z : out Bit);
end AND2;

entity EXAMPLE is ... end;
architecture STRUCTURE of EXAMPLE is
    component AND2 is
    ```
port (X, Y : in BIT; Z : out BIT);
end component;
signal S1, S2, S3 : BIT;
begin
  C1 : AND2_COMP port map (S1, S2, S3);
end STRUCTURE;

The author had intended to write:

C1 : AND2 port map (S1, S2, S3);

**Objection Raised**
When describing a VHDL'92 feature the author wrote:

*Integer'IMAGE(Val)*

But the object *Val* was declared to be of type *Boolean*, and hence the author should have written:

*Boolean'IMAGE(Val)*

**Objection Raised**
Author erroneously wrote the phrase "carry-look-mahead".

**Objection Raised**
The cover jacket of a VHDL book erroneously uses the acronyms MUL9 and STAG instead of MVL9 and JTAG respectively.

7. **Summary**

As a VHDL educator I am saddened by the errors outlined in this paper. The purpose of a VHDL book is to share the insights and techniques that its author has accumulated over the years. The reader should never feel confused or disoriented about what he or she has just read. A VHDL book should be a positive, enlightening, and perhaps even an entertaining experience. A VHDL book should strive to become an integral part of the reader’s growth as a VHDL designer and engineer.

Though this paper has highlighted a wide variety of flaws, it, nonetheless has also produced two very positive byproducts. Firstly, there is an educational benefit since, in the final analysis, this paper explored many important VHDL related concepts. Secondly this paper may also be viewed as a friendly warning to aspiring VHDL authors. Based on what you have seen in this paper, you now know what can potentially go wrong when writing a book on this subject. Anyone contemplating the authorship of a VHDL book should make certain that

- The VHDL Fact are correct.
- The Twilight Zone is never entered.
- The final manuscript is thoroughly proof read by a trusted VHDL colleague.

As formidable as these requirements seem, the beneficial outcome will be very rewarding for both the author and the VHDL community as a whole. By writing a high quality book the VHDL author will be contributing towards the positive and enthusiastic growth of this IEEE standard hardware description language.