VHDL Coding Style and Guidelines

ISSUE 0.3

9th December 2003

Prepared by .....
M J MacIntosh Electronics Engineer

Approved by .....
D J Ives Head of Electronics Group

CHANGE RECORD

Issue	Date	Section affected	Change Description
0.1	21/01/03	All	First draft
0.2	3/12/03	2.1.2	Addition of CVS keywords to header file
0.3	18/12/03	2.1.1	Changed *.vhdl to *.vhd
		2.1.2	Add copyright
		2.2.3	Additional formatting
		2.3	Indent using spaces not tabs
		3.4	Elaborated on processes
		4 & 5	Add new section for Project variants and Appendices

TABLE OF CONTENTS

1. INTRODUCTION

This document presents the VHDL coding style to be used for any VHDL code generated at the UK ATC. The style is based on the guidelines presented in the 'Coding Standards' section of the 'VHDL Golden Reference Guide', Version 2.0, March 1997, Doulos Ltd (ISBN 0-9537280-1-3).

The document is split into two main parts: 1) Lexical coding standards which control text layout, naming conventions, commenting, indenting to improve readability, and revision control. 2) Synthesis coding standards which control VHDL style and help generate consistent designs.

This is intended to be a working document and it is expected that the document will be updated as experience with VHDL in the electronics group develops.

2. LEXICAL CODING STANDARDS

2.1 SOURCE FILES

2.1.1 General

Each VHDL source file should contain one of the following: one entity and its architectures, one package and its package body, or several related configurations.

Source file names should relate to the file contents (e.g. EntityName.vhd, PackageName.vhd or configs.vhd).

2.1.2 Header

Each source file should have a header containing at least the following information:
Check the Appendices for project specific information.

```
-- Copyright (c) 2003 UK Astronomy Technology Centre
                 All Rights Reserved

--  THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF THE UK ATC
--  The copyright notice above does not evidence any
--  actual or intended publication of such source code.

--  SOURCE CODE IS PROVIDED "AS IS". ALL EXPRESS OR IMPLIED CONDITIONS,
--  REPRESENTATIONS, AND WARRANTIES, INCLUDING ANY IMPLIED WARRANT OF
--  MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR
--  PURPOSE, OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT
--  THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

-- <Title>
--
-- <revision control keyword substitutions e.g. $Id$>
--
-- Project:       <project name>
-- Author:        <author name>
-- Organisation:  <organisation name>
--
-- Description:
-- <description text>
--
-- Revision history:
-- <date $Date$>  -           <text>            - <initials $Author$>
-------------------------------------------------------------------------
```

Items in angle brackets (e.g. <Title>) should be replaced as appropriate. CVS keywords can be used to automatically insert text e.g. $Date$ will automatically input date and time revision was checked in and who by $Author$. A template file (header.vhd) containing this information can be found in F:\Groups\Electronics\VHDL.

2.2 NAMING CONVENTIONS

2.2.1 VHDL keywords

All VHDL keywords should be in lower case.

2.2.2 Predefined names

All predefined names e.g. from packages, should be in upper case.

2.2.3 User defined names

User defined names should start with a capital letter. They should be meaningful and informative expect that local names e.g. loop variables may be terse.
The format My_Variable would be preferable to MyVariable.

2.2.4 Architecture names

Architectures should be named according to their functionality as follows:

Name	Description
Reference	Reference design
Behaviour	Behavioural model
RTL	RTL level design
GateLevel	Gate level design
TestBench	Test bench

2.2.5 Labels

All processes and concurrent assignments should be labelled with a meaningful and informative name. This provides additional documentation and aids debugging.

2.3 TEXT LAYOUT

Write only one declaration or statement per line.

Indent text by 3 spaces between pairs of related control statements e.g. begin and end, if and end if, etc. Do not use tabs as the tab spacing can be interpreted differently by various text editors.

Write comments to explain, not duplicate, VHDL code. In particular it is important to comment interfaces.

Write use clauses at the top of each entity, package or configuration to make dependencies easy to locate.

2.4 CONSTANTS AND ATTRIBUTES

Wherever possible use constants and attributes rather than directly embedding literal numbers or strings within the code. This will make the code easier to maintain and update.

3. SYNTHESIS CODING STANDARDS

3.1 DESIGN PARTITIONING

Partition the design into small functional blocks, and use a behavioural style for each block. Gate level descriptions should be avoided except for critical parts of the design.

3.2 CLOCKING STRATEGY

The clocking strategy for the design must be well defined and implemented explicitly in the VHDL code. Wherever possible separate blocks with different clocking strategies, e.g. single clock, multi-phase clocks, or gated clocks. Take care that clock and reset signals are clean and not generated from combinational logic or unintentionally gated.

Additionally, clock signals must not be renamed/reassigned in any file, except through the port-map, as this will lead to simulation delta-cycle errors where there are mismatches between simulation and real functionality. This also applies to the reset signal.

For example:
clk_local <=clk; -- this is bad, it introduces a delta in simulation time.
Port map (clk=> clk_local -- this is OK.

3.3 TESTING STRATEGY

Define a test strategy for the design and ensure that the VHDL code is written to support it. For example, make all flip-flops resettable and allow test access from external pins.

3.4 PROCESSES

Not all processes listed will be suited for every project so check project variants in the Appendices for preferred processes for particular projects. Where a process is used it should conform exactly to one of the standard synthesizable process templates shown below.

Process 1.

```
process (Inputs)                    -- All inputs in sensitivity list
begin
        …                           -- Outputs assigned for all input conditions
        …                           -- No feedback
end process;                        -- Gives pure combinational logic
```

Process 2.

```
process (Inputs)                    -- All inputs in sensitivity list
begin
        if Enable = '1' then
                …                   -- Latched actions
        end if;
end process;                        -- Gives transparent latches + logic
```

Process 3.

```
process (Clock)                          -- Clock only in sensitivity list
begin
        if Rising_edge(Clock) then       -- Test clock edge only
                …                        -- Synchronous actions
        end if;
end process;                             -- Gives flip-flops + logic
```

Process 4.

```
process (Clock, Reset)                   -- Clock and reset only in sensitivity list
begin
        if Reset = '0' then              -- Test active level of asynchronous input
                …                        -- Asynchronous actions
        elsif Rising_edge(Clock) then    -- Then test clock edge only
                …                        -- Synchronous actions
        end if;
end process;                             -- Gives flip-flops + logic
```

Process 5.

```
process                                  -- No sensitivity list
begin
        wait until Rising_edge(Clock);
                …                        -- Synchronous actions
        end if;
end process;                             -- Gives flip-flops + logic
```

Combinational process must not contain incomplete assignments i.e. all outputs must be assigned for all combinations of input values. Otherwise unwanted latches will be produced.

When designing combinational logic avoid creating a latch unintentionally due to the HDL design style. For example, when CASE or IF statements do not cover all possible input conditions, combinational feedback can generate latches to hold the output in the case when a new output value is not assigned.
Omitting the final ELSE clause or WHEN OTHERS clause from an IF or CASE statement, respectively, can also generate a latch. 'Don't care' assignments on the default conditions tend to prevent latch generation

All processes describing combinational and latched logic must have all of the inputs in the sensitivity list and must not contain feedback i.e. signal and variables assigned as outputs from the process must not be read as inputs to the process.

Clocked processes with a sensitivity list must have only the clock and any asynchronous inputs e.g. reset or set in the sensitivity list. Clocked process without a sensitivity list must have only one wait statement of the form wait until clock_edge_expression, as the first executable statement in the process. Flipflops are synthesized when signals are assigned in a clocked

process, or when variables assigned in a clocked process retain their value between process executions. These operations should be avoided wherever possible.

All internal state registers must be resettable in order that the Register Transfer Level and gate level descriptions can be reset into the same known state for verification. This does not apply to pipeline or synchronization registers.
Finite state machines and other sequential circuits with unreachable states, e.g. a decade counter, must explicitly define behaviour for all 2^N states, including the unreachable states, if the behaviour of the hardware in all states is to be controlled.

Delays in signal assignments should be avoided except where necessary to avoid the problem of a delta delay clock skew at the RTL level.

Signals and variables should only be initialized to either 'U' or 'X' so that initialization problems are revealed in the VHDL simulator.

Do not write VHDL code which relies on the order of values within an enumeration type. The order might be changed during optimization and the code will be harder to maintain.

Signals and variables of type INTEGER should have a range constraint, otherwise they will synthesize to 32 bit busses.

Take care to check VHDL code which uses dynamic indexing, loop statements, or arithmetic operators. Large numbers of gates can be synthesised which can be difficult to optimise.

4. PROJECT SPECIFIC VARIATIONS

This guideline is not project specific and not all projects and collaborations may want to use all instances of the coding style so this is an area to record project specific variations.

If we find over time that most projects are rejecting common areas of the style then that portion can be removed from the general coding style.

4.1 SCUBA2

- SCUBA -2 will use header file in Appendix 5.1
- No latches to be used – registers more efficient and robust in FPGA designs so do not use Process 2.
- Do not use Process 5.
- Do not use enumeration types – too hard to maintain.
- Avoid using delays in anything that will be synthesized. Delays in testbench OK
- As far as possible limit data types to 'std_logic' and 'std_logic_vector'.

5. APPENDICES

5.1 SCUBA-2 HEADER FILE

```
-- Copyright (c) 2003 SCUBA-2 Project
                  All Rights Reserved

--  THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF THE SCUBA-2 Project
--  The copyright notice above does not evidence any
--  actual or intended publication of such source code.

--  SOURCE CODE IS PROVIDED "AS IS". ALL EXPRESS OR IMPLIED CONDITIONS,
--  REPRESENTATIONS, AND WARRANTIES, INCLUDING ANY IMPLIED WARRANT OF
--  MERCHANTABILITY, SATISFACTORY QUALITY, FITNESS FOR A PARTICULAR
--  PURPOSE, OR NON-INFRINGEMENT, ARE DISCLAIMED, EXCEPT TO THE EXTENT
--  THAT SUCH DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

-- For the purposes of this code the SCUBA-2 Project consists of the
-- following organisations.

-- UKATC, Royal Observatory, Blackford Hill Edinburgh EH9 3HJ
-- UBC,   University of British Columbia, Physics & Astronomy Department,
--        Vancouver BC, V6T 1Z1

-- <Title>
--
-- <revision control keyword substitutions e.g. $Id$>
--
-- Project:       <project name>
-- Author:        <author name>
-- Organisation:  <organisation name>
--
-- Description:
-- <description text>
--
-- Revision history:
-- <date $Date$>  -           <text>            - <initials $Author$>

--
------------------------------------------------------------------------
```