CI Brasil - CT2 Campinas

Phase II Project

EMC08 - 8 Bit Microcontroller for Automotive Engine Application

Digital A

# Verilog Code Standard

Version # 1.0:

Last revision date: **17,Aug, 2010**

# Version Control

Table 1 – Version control

| Revision | Date | Author | Comments | Revisors |
|---|---|---|---|---|
| 1.0 | 17/Aug/2010 | Vinícius Amaral<br>Pedro Fausto Junior<br>Thiago Silva Santos | Initial version | |

# Summary


1 File names Conventions ..... 4
1.1 File types <type> ..... 4
2 HDL Code Items Naming Conventions ..... 4
2.1 Signal Names ..... 4
2.1.1 Suffixes <suffix> ..... 4
2.1.2 Internal Signals ..... 4
2.1.3 Top Module Inputs/Outputs: ..... 5
2.1.4 Sub-Module Signals ( signals that don't leave the top module) ..... 5
3 Comments ..... 5
3.1 File Headers Comments ..... 5
3.2 Function or tasks comments ..... 5
3.3 Other comments ..... 6
4 Code Style ..... 7
4.1 Use 3 spaces and empty lines to cascade code ..... 7
4.2 Use spaces to align code lines ..... 7
4.3 Module declaration ..... 8
4.4 Line length must not exceed 80 characters ..... 8
5 General Coding Techniques ..... 9
5.1 Parameterize your code ..... 9
5.2 Use parameters for state encodings ..... 9
5.3 Declare explicitly the size (in bits) and base of every number ..... 9
5.4 Connect ports by name in module instantiations ..... 9
6 Extra Information ..... 10

# Verilog Code Standard

## 1 File names Conventions

Follow the model: <module>_<sub module>_<type>.v

Example:

```
core_alu.v
core_alu_task.v
```

### 1.1 File types <type>

task: file consists of tasks

func: file consists of functions

defines: file consists of macros

## 2 HDL Code Items Naming Conventions

Names must describe the purpose of the item. Use meaningful names, in English.

Names must be composed by characters [a-z, 0-9, _ ]. Parameters and constants must be UPPERCASE, all others in lowercase.

Names composed with several words must be separated with underscore (_).
Names must not exceed 32 characters.

### 2.1 Signal Names

#### 2.1.1 Suffixes <suffix>

_i: input

_o: output

_b: active low

#### 2.1.2 Internal Signals

<signal_name>_<signal_suffix>

Example:

```
`define     ADDR_BUS_WIDTH 16;
parameter   DATA_BUS_WIDTH  = 8;
reg         [7:0] pc;
wire        [5:0] alu_selector;
```

### 2.1.3 Top Module Inputs/Outputs:

<top_level>_<signal_name>_<suffix>

Example:

```
reg [2:0] interrupt_addr_vector_o_b;
```

### 2.1.4 Sub-Module Signals ( signals that don't leave the top module)

<submodule>_<signal_name>_<suffix>

Example:

```
reg [15:0] ula_result_o;
reg [15:0] fsm_result_i;
```

# 3 Comments

## 3.1 File Headers Comments

Every RTL and behavioral Verilog file will be documented with the header as follow:

```
// ------------------------------------------------------------------
// CI Brasil - CT2 Campinas
// Phase II Project
// EMC08 - 8 Bit Microcontroller for Automotive Engine Application
// ------------------------------------------------------------------
// File Name: top_coding_style.v
// Module Name: module_name
// Author: Thiago Santos
// E-mail: tssantos@gmail.com
// ------------------------------------------------------------------
// Release History
// Version   Date        Description
// 1.0       29/07/2010  Initial version
// 1.1       30/07/2010  Modified module definitions
// 1.2       30/07/2010  Updated header
// ------------------------------------------------------------------
// Description
// This file describes the coding style to be used on rtl
// implementations.
// File names must be all lowercase separated by underscore if needed
// ------------------------------------------------------------------
```

## 3.2 Function or tasks comments

Every function or task must have the following comments:

```
// ------------------------------------------------------------------
// NAME: instrunction_fetch
// TYPE: TYPE can be func, task, primitive
// ------------------------------------------------------------------
// PURPOSE: Short description of functionality
// ------------------------------------------------------------------
// PARAMETERS
```

```
// PARAM_NAME:    RANGE:      DESCRIPTION:            DEFAULT:    UNITS:
// data_width:    [32,16]:    width of the data:      32:         n.a.
// stack_depth    [1,8]:      depth of stack          5:          n.a.
// -------------------------------------------------------------------
```

## 3.3 Other comments

- Each functional section of the code must be preceded by comments describing the code's intent and function.
- Comments in English too.
- Comments must be as complete as possible.
- One line comments (//) must be used. Do not use multiline (/*...*/) comments.
- Do not comment old code, or unused code. They must be deleted as opposed to commented out.
- For nets and variables, it is recommended to have a descriptive comment, preferably on the same line. If the comment is not on the same line, it should be on the preceding line.
- Always comment end and endcase statements greater than 10 lines. Use an annotation of the construct ended.

Example:

```
// procedural block that checks each state of FSM and ...
always @(posedge my_clk)
begin
   code line 1;
   code line 2;
   ...

   // checking 'a' flag condition. If a < b the module
   // must set flag_1 and flag_2 else ...
   if (a < b)
   begin
      code line 3;

      if (a < c)
      begin
         code line 4;
      end

      else
      begin
         code line 5;
         code line 6;
      end

   end // if (a < b)

end // always

// for each state of FSM given the ...
case (state)
   START:
   begin
      code line 7;
   end

   ...
```

```
   DEFAULT:
   begin
      code line 7;
   end

endcase // case (state)
```

# 4 Code Style

## 4.1 Use 3 spaces and empty lines to cascade code

As standard, always use 3 spaces and not tab key to cascade code.

Use one empty line before if, else, case, function, etc.

Use one empty line after end statement.

It improves readability. When writing a code block (begin, case, if statements, etc.), it is useful to complete the frame first, in particular to align the *end* of the code block with the *begin*.

Example:

```
// sum of dealer's cards
if (card == 4'b1) // card is an ace
begin

  if (dealer_ace == 1'b0) // no ace before
  begin
    dealer_ace = 1'b1;

    if (sum_dealer > 5'd10)
      sum_turn_dealer = sum_dealer + 1'b1;

    else
    begin
      sum_turn_dealer = sum_dealer + 4'd11;
      dealer_big_ace = 1'b1;
    end

  end // end if – no ace before

end // end if - card is an ace
```

## 4.2 Use spaces to align code lines

Example:

```
sum_player     = 5'd0;
sum_dealer     = 5'd0;
player_big_ace = 1'b0;
dealing        = 3'd3;
player_ace     = 1'b0;
dealing_end    = 1'b0;
dealer_big_ace = 1'b0;
dealing        = 3'd3;
```

```
dealer_ace      = 1'b0;
```

## 4.3 Module declaration

Follow the example to module declarations:

```
module prescaler(
                 core_32m,
                 system_clk,
                 scan_mode_test,
                 reset_b,
                 div16_clk,
                 div16_clk_b
                );
input  [31:0] core_32m;  // 32 MHz core signal
input  system_clk;       // system clock
input  scan_mode_test;   // scan mode clock
input  reset_b;          // active low hard reset, synch w/ system_clock

output div16_clk;        // input clock divided by 16
output div16_clk_b;      // input clock divided by 16 and inverted

reg[3:0] count;          // counter to make clock divider
reg div16_clk;           // input clock divided by 16
reg div16_clk_b;         // input clock divided by 16 and inverted

wire[3:0] count_ns;      // clock divider next state input
```

Use one port declaration por line, as shown above, Do not use:

```
input a, b;
```

Preserve port order, it is recommended that the port declarations be listed in the same order as the port list of the module declaration.

It is recommended that all wire declarations be grouped together in one section following the input/output/inout. Port nets need not be redeclared in wire declarations in addition to the input/output/inout declarations.

## 4.4 Line length must not exceed 80 characters

It is recommended that line length not exceed 80 characters. Use <Enter> to break long code lines.

Example:

```
if ((sum_dealer < 5'd17) || (sum_dealer == 5'd17 && dealer_big_ace == 1'b1)
|| (sum_player == 5'd17 && player_big_ace == 1'b1))
begin
   ...
end
```

# 5 General Coding Techniques

## 5.1 Parameterize your code

Use parameters instead of text macros for symbolic constants.

Example:

```
‘define DATA_WORD 8
‘define DATA_LONG (4 * ‘DATA_WORD)
```

## 5.2 Use parameters for state encodings

Example:

```
parameter [1:0] RESET_STATE;   // synthesis enum state_info
parameter [1:0] TX_STATE;      // synthesis enum state_info
parameter [1:0] RX_STATE;      // synthesis enum state_info
parameter [1:0] ILLEGAL_STATE; // synthesis enum state_info
```

## 5.3 Declare explicitly the size (in bits) and base of every number

Example:

```
dealing    = 3'd3;

if (sum_player > 5'd21)
   ...

else if (hit == 1'b1)
   ...
```

Do not use:

```
dealing    = 3;

if (sum_player > 21)
   ...

else if (hit == 1)
   ...
```

## 5.4 Connect ports by name in module instantiations

Example:

```
timers timers_instance (
                        .signal_a (signal_a),
                        .signal_b (signal_b)
                        );
```

# 6 Extra Information

Other code practices and rules can be found in the following reference:

- Freescale semiconductor, *Verilog HDL Coding, Semiconductor Reuse Standard*. SRS V3.2.