Student Name:	Aviral Mittal
Student's Email	aviral.mittal@sli-institute.ac.uk, avimit@yahoo.com
ISLI Registration Number	2005/21
Module	IPB
Assignment Lecturer	Prof Trughul Arslan

This Report consist of following 6 Sections.

- Section 1: General Introduction

  1.1 Aims and Objectives of this exercise
- Section 2: Soft IP Evaluation:

  2.1 Introduction to Soft IP Evaluation
  2.2 Steps
  2.3 Rules and Guidelines, how far each rule was followed/not followed, and why with example corrections
    2.3.1 System Level Design Issues: Rules and Tools
    2.3.2 RTL Coding Guidelines
      2.3.2a: Comment upon overall results and quality of RTL
    2.3.3 Macro Synthesis Guidelines
    2.3.4 Verification Guidelines
    2.3.5 Deliverable Guidelines
  2.4 Results
- Section 3 Hard IP Generation and Evaluation

  3.1 Introduction to IP Hardening Process and Hard IP Evaluation
  3.2 Steps
  3.3 Results
- Section 4: Common or total Results and results discussion
- Section 5: Comment on OpenMore
- Section 6: Appendix

  A Filled OpenMore Spread sheet
  B Figures from Hard IP Generation Process
  C Scrips (Perl Code) used to evaluate some of the guidelines/Rules
  D Log files generated by the script, if there are too many violations corresponding to a rule. As it might not be possible to put all the violations corresponding to a single guideline in middle of the report.

Section 1: General Introduction.

As the IP business is rising, and more and more companies are adopting business models based on IP licensing. With the perpetual increase in IPs in the market, the problem of 'how good' and IP is also increasing with a great pace. There are various criteria of evaluating 'how good' an IP is, such as functionality, technology, performance, cost, area, support, power, reuse etc. The scope of this assignment is to focus on 'reuse'.

Now there can be various ways in 'reusability' can be evaluated, among them is OpenMORE assessment program, developed jointly my Synopsys and Mentor Graphics. OpenMore assessment program is based on the Reuse Methodology Manual (RMM). The focus of this exercise will be on the RMM Section 5 i.e RTL Coding Guidelines, however the RMM section 1 i.e Macro Design Guidelines, and RMM Section 3 i.e System-Level Design Issues: Rules and Tools, will also be addressed briefly.

Although a soft-ip has been provided, but to be able to 'evaluate' it to a degree of satisfaction, the IP will also be hardened. That is, the scope of this assignment will also cover a method to convert the soft-ip i.e RTL to hard-ip i.e layout or GDSII using industry standard tools. So this assignment report will also provide a section on RTL to GDSII flow that will be run on the soft-ip, which will produce a final hardened layout of the IP provided. Issues (if any) during the hardening process will also be discussed. The process of hardening the IP will be discussed in Section 3 "Hard IP Generation and Evaluation".

Section 1.1 Aims and Objectives of this exercise

Following is the point wise description of the objectives/aims of this assignment:

- Evaluate the IP against the OpenMORE assessment criteria: This will greatly help us in understanding the basic RTL coding guidelines that 'must' be followed while writing a industry standard IP.
- Harden the IP: This will greatly help us understand the industry standard process of converting an RTL to GDSII. This might also help in catching problems in the IP, which might be an issue in converting the RTL into a layout.
- To be able to read and understand an industry standard documentation and use it to understand the IP and the IP environment provided.
- To be able to learn the skills and tools used to evaluate an industry standard IP.

Notes: The Openmore spreadsheet submitted with this assignment will not have comments in SOFT IP evaluation section. This is because detailed comments on each guideline are made in this report. However in the HARD IP section, the spread sheet will have all the comments, as the evaluation is done only on the spread sheet.

Section 2: Introduction to Soft IP Evaluation

The evaluation of the soft-ip is done using the OpenMORE assessment program, which is itself based upon the RMM. The evaluation will focus mainly on Section 5 of RMM i.e on “RTL Coding Guidelines”, and will go through the other Sections briefly.

RTL stands for Register-Transfer-Level, which is used to code the hardware using a Hardware Description Language so that the resulting code can be ‘synthesized’ into a gate level netlist which should be ‘functionally’ equal to the HDL code. RMM Section 5 provides guidelines and rules which when followed make the HDL code or the IP more reusable, easy for others to understand, easy for others to modify, more synthesis friendly, more portable. These simple guide lines when followed can make the IP very much more marketable as compared to the one which do not follow these guidelines.

Section 2.1: Steps

- Read documentation and have as much information as possible about the IP environment, IP use, IP naming conventions. Comment upon the documentation.
- Run Simulation Scripts, Synthesis Scripts, post synthesis simulation scripts, and see if there are any problems with these. Comment upon the scripts and how far they succeed in performing the task they were designed to do.
- Focus on RTL code, (all RTL files), take one by one the ‘RTL coding guidelines’ and see if they are followed. For some guidelines/rules, it might be possible to write a simple perl script which will report any violations of the guidelines/rules. This will help in saving time, as reading the whole RTL code with respect to only one guideline can be very time consuming, this will also help in quickly evaluating IPs in future i.e. scripts can be re-used. But at the same time it is also recognised that writing script for all the guidelines is a complex task and is beyond the scope of this assignment. Also, The scripts written will be very simple and may not be considered as a 100% secure method to report a violation against a given guideline. A score for each rule/guideline will then be assigned. Appendix A will contain the source code of all perl scripts used along with information about which guideline/rule it refers to.
- For each guideline/rule, a table just like the one shown below will be used.

Table 1: G 5.2.1(example)

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1						

A detailed comment if needed will follow the table, describing if or not the guideline/rule was followed and why. A correction may also be suggested. Please note that if a rule/guideline is simple, no explanation will be given for what the given rule/guideline itself means, however for complex rules/guidelines a good explanation will be given.

It is to be noted that the scoring in each table refers to all the RTLs taken as one entity for the IP. In some cases if the guideline/rule as a number of violations, the full log of violations will be put in Appendix D. All the headings in the table are the same as the OpenMORE spread sheet, except ‘Script Used’ and ‘Score’. ‘Script used’ column will indicate the name of the script if used to

evaluate that particular guideline. If no script is used it will say ‘N/A’. The name of the table will be the RMM2 Section name. ‘Socre’ column tells the score given after assessment.

Section 2.3 Rules and Guidelines, how far each rule was followed/not followed, and why with example corrections
2.3.1 System Level Design Issues: Rules and Tools
2.3.2 RTL Coding Guidelines: Max Score 346

RMM2 5.2 Basic Coding Practices: Max Score 52
R5.2.1.1
..

Table 2: R 5.2.1.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.1	Documented naming conventions used consistently thought the design	R	A	10	10	N/A

Comment: Following are the documents provided with the IP
Readme_ipba_project.pdf
Environment_Strategy.pdf
Rapier_External_Memory_Controller.pdf
external_memory_programmers_guide.pdf
The document *Environment_Strategy.pdf* Section 3.3.1 does say about naming conventions used in verilog, and verilog actually uses those naming conventions.
The document also documents file naming conventions, directory structure in section 3.3.3. Overall it gives a good proof of documenting naming coventions and use of the same.
Hence the Assessment for this guideline is Always

G5.2.1.2
.

Table 3: G 5.2.1.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.2	Lower case letters for all signal names, variables, and user-defined types	G	A	2	2	lcase.pl

Comment: All the RTL(s) strictly followed the above guideline. Perl script ‘*lcase.pl*’ was used to evaluate it, and it reported no instance of lower case letters used in *ports, wires, regs*. There are no user-defined types in the design. Hence the Assessment for this guideline is Always.
Some examples from the RTL(s) are given below.

```
 reg         b_transfer_on_bus; // True is there is a valid transfer
// on the bus
 reg         transfer_type;   // Set to ‘AHB_WRITE if the transfer is
// a write transfer, ’AHB_READ otherwise

 wire        hsel;  // hsel is true when any of the hsel_mem bits are true

 wire        b_wait_states_left;  // True if there are more wait states
// left in this transfer
 wire        b_error_condition;
```

G5.2.1.3

.

Table 4: G 5.2.1.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.3	Upper case letters for names of constants and user defined types	G	A	2	2	ucase.pl

Comment: All the RTL(s) strictly followed the above guideline. Perl script ‘ucase.pl’ was used to evaluate it, and it reported no instance of lower case letters used in constants defined in the RTL using *‘define*. There are no user-defined types in the design. Hence the Assessment for this guideline is Always. Some examples from the RTL are given below:

```
‘define ST_IDLE        6’b000001
‘define ST_ERROR_START  6’b000010
‘define ST_READ_WAIT    6’b000100
‘define ST_WRITE_ADDR   6’b001000
‘define ST_WRITE_WAIT   6’b010000
‘define ST_READ_ADDR    6’b100000
```

G5.2.1.4

.

Table 5: G 5.2.1.4

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.4	Meaningful names for signals, ports, functions, and parameters	G	A	2	2	N/A

Comment: All the RTL(s) strictly followed the above guideline. Hence the Assessment for this guideline is Always. Some examples from the RTL are given below:

```
 input     hclk;           // AHB system clock. Only the rising edge of
// this clock is used throughout the module.
 input     hreset_n;       // Active low AHB sychronous reset.
 input [3:0] hsel_mem;     // Active high AHB memory bank select.
// MEM3, MEM2, MEM1, MEM0.
```

G5.2.1.4.a

.

Table 6: G 5.2.1.4a

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.4.a	Names do not differ in case only	G	A	2	2	lcase.pl

Comment: All the RTL(s) strictly followed the above guideline. Perl script 'lcase.pl' was used again to evaluate it, and it reported no instance of upper case letters in *'inputs, outputs, regs, wires'*. So there cannot be any duplication of signals using different case. There are no user-defined types in the design. Hence the Assessment for this guideline is Always.

G5.2.1.5

.

Table 7: G 5.2.1.5

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.2	Short but descriptive names for elaboration parameters	G	A	2	2	N/A

Comment: All the RTL(s) strictly followed the above guideline. Hence the Assessment for this guideline is Always. Below are given a few examples from the RTL(s)

```
parameter tm_prop = 20;
parameter par_little_endian = 1;
```

G5.2.1.6

.

Table 8: G 5.2.1.6

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.6	Name clk or prefix clk for the clock signals	G	N	2	0	clkname.pl

Comment: None of the RTL(s) followed the above guideline. Hence the Assessment for this guideline is Never. Below are given a few examples from the RTL(s). The clock used in the deign is called ‘hclk’ instead. Following is the line from the RTL(s) which shows the name as *‘hclk’*.
input hclk; // AHB system clock. Only the rising edge of
Perl script does not depend upon the comment, it checks for *‘always @ (posedge any_signal)’*, and then determines that the ‘*any_signal*’ is a clock. Since the IP is targeted on FPGA, it does not use any asyn resets, and hence anything following and ‘*posedge*’ in the design will be a clock signal.
Correction: The name of the clock can be ‘*clk_h*’ or even ‘*clk_hclk*’.

G5.2.1.7.

Table 9: G 5.2.1.7

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.7	Same name for all clock signals driven form the same source	G	A	2	2	N/A

Comment: All of the RTL(s) strictly followed the above guideline. All the RTL(s) use clock called ‘*hclk*’ which is sourced by a single ‘*hclk*’ input at the top level. Hence the Assessment for this guideline is Always. Below are given a few examples from the RTL(s).

G5.2.1.8

Table 10: G 5.2.1.8

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.8	Active low signal names end with an uderscore followed by a lowercase character consistently(_n)	G	A	2	2	N/A

Comment: All of the RTL(s) strictly followed the above guideline. All the signals which are described as ‘Active Low’ in comments use (_n) character. Hence the Assessment for this guideline is Always.Some examples from the RTL(s) are shown below:
input hreset_n; // Active low AHB sychronous reset.
output mem_output_enable_n_o; // Active low output enable signal

G5.2.1.9

Table 11: G 5.2.1.9

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.9	Name or prefix rst used for reset signals. If the reset signal is active low, user rst_n	G	N	2	0	N/A

Comment: None of the RTL(s) followed the above guideline. Hence the Assessment for this guideline is Never. Below are given a few examples from the RTL(s). The reset pin used in the deign is called ‘hresest_n’ instead. Following is the line from the RTL(s) which shows the name as ‘hreset_n’.
input hreset_n; // Active low AHB sychronous reset.
Correction: The name of the above reset signal can be ‘*rst_n*’ or ven ‘*rst_n_hreset*’

G5.2.1.10

Table 12: R 5.2.1.10

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.10	VHDL Guideline Not Applicable	R	N/A	2	2	N/A

R5.2.1.11

Table 13: R 5.2.1.11

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.11	If Verilog, always use (x:0) for multibit ports or signals, rather than (0:x)	R	A	10	10	downto.pl

Comment: All of the RTL(s) strictly followed the above guideline. Hence the Assessment for this guideline is Always. A script ‘*downto.pl*’ was used, it reported no violations.Below are given few examples from the RTL(s).
input [3:0] hsel_mem;
reg [1:0] hresp;

wire [1:0] hresp_reg;

G5.2.1.12

Table 14: G 5.2.1.12

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.12	Same name or similar names, for ports and signals that are connected that are not clocks	G	A	2	2	N/A

Comment: When binding an instance to wires, it is recommended that the names of the wires and the ports(of the instantiated module) must be simlar or same. All of the RTL(s) strictly followed the above guideline. Hence the Assessment for this guideline is Always. Following are the examples form the RTL(s).

```
i_ahb_ext_mem_con
  (.hclk          (hclk),
   .hreset_n      (hreset_n),
   .hsel_mem      (hsel_mem),
   .hwrite        (hwrite),
```

G5.2.1.13

Table 15: G 5.2.1.13

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.13	Signal naming conventions listed used consistently and exhaustively	G	N	2	0	N/A

Comment: None of the RTL(s) followed the above guideline. No signals were found wit suffixes *'_r'* , *'_a'*, *'_pn'* , *'_nxt'* , *'_z'* Hence the Assessment for this guideline is Never.

End of RMM2 Section 5.2.1 Max Marks 52: Marks Scored = 36. %age = 69%

RMM2 5.2.3 Architecture Naming Conventions

Only applicable to VHDL.Not applicable here. Max marks = 2, Marks scored = 2.

RMM2 5.2.4 Headers in Source Files : Max Score 10

R5.2.4.1

Table 16: R 5.2.4.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.4.1	Header included at the top of every source file, including scripts, containing RMM recommended elements.(author name is optional)	R	A	10	10	N/A

Comment: All of the RTL(s) strictly followed the above guideline. All the RTL(s) include a header with filename, author, description, date, modification history. Hence the Assessment for this guideline is Always.

RMM2 5.2.5 Use Comments Max Score 12
R5.2.5.1

Table 17: R 5.2.5.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.5.1	Comments used generously to explain all processes, functions, and declarations of types and subtypes	R	A	10	10	N/A

Comment: All of the RTL(s) strictly followed the above guideline. The RTL(s) are heavily commented. Hence the Assessment for this guideline is Always.

G5.2.5.2

Table 18: G 5.2.1.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.1.2	All ports, signals and variables or groups of signals or variables explained in comments.	G	S	2	1	N/A

Comment: All of the RTL(s) strictly followed the above guideline. The ports, signals, variables in RTL(s) are commented, but not all of them have comments. Hence the Assessment for this guideline is Sometimes. Following are some examples:
Uncommented Port Names:

input hwrite;
input [1:0] htrans;
input [2:0] hsize;

input hready;
input [31:0] haddr;
input [31:0] hwdata;

Commented Ports:
output hready_reg; // AHB register hready output.
output hready_mem; // AHB memory hready output.
output [1:0] hresp_reg;
// AHB response. These modules only provide two types of response:
// OKAY and ERROR
output [1:0] hresp_mem;
output [31:0] hrdata_reg; // AHB register data output for read cycles.
output [31:0] hrdata_mem; // AHB memory data output for read cycles.

RMM2 5.2.6 Keep Commands on Separate Lines Max Score 10
R5.2.6.1

Table 19: G R5.2.6.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.6.1	Separate line used ofr each HDL statement	R	A	10	102	sepline.pl

Comment: All of the RTL(s) strictly followed the above guideline. No occurrence of two statements were reported by the script '*seplie.pl*' which was used. The logic for finding this was looking for more than one semicolons in a single line. Hence the Assessment for this guideline is Always

RMM2 5.2.7 Line Length Max Score 2

G5.2.7.1

Table 20: G 5.2.7.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.7.1	Line Length throughout consistently kept to 132 characters or less	G	A	2	2	chars132.pl

Comment: All of the RTL(s) strictly followed the above guideline. A script '*chars132.pl*' was used for this purpose. It reported no violations. Hence the Assessment for this guideline is Always The author has made efforts to follow this guideline, as is evident from the RTL code. For example the following line in the RTL code has been put into 4 separate lines, instead of just a single

line. This is a single statement and this statement could have violated the guideline, if proper care was not taken.

```
if(
  (current_state == 'ST_WRITE_ADDR && next_state == 'ST_IDLE) ||
  (current_state == 'ST_WRITE_ADDR && next_state == 'ST_WRITE_WAIT) ||
  (current_state == 'ST_WRITE_WAIT && next_state == 'ST_WRITE_WAIT) ||
  (current_state == 'ST_WRITE_WAIT && next_state == 'ST_IDLE))
```

There are numerous occurrences of such statements in the design which proves that this guideline has been take care of, and its just not a coincidence that it is followed automatically

RMM2 5.2.8 Indentation Max Score 12

R5.2.8.1

Table 21: G 5.2.8.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.8.1	Indentation used to improve the readability of continued code lines and nested loops.	R	A	10	10	N/A

Comment: All of the RTL(s) strictly followed the above guideline. Hence the Assessment for this guideline is Always. Following is an example of the code showing indentation

```
begin
  if (hsel && hready && (htrans[1] == 1'b1))
    // Transfer request currently on the bus
    if (error_condition)
      begin
        slave_state <= 'EMR_ERROR_WAIT;
        hready_resp <= 1'b0;
        hresp[1:0]  <= 2'b01;
      end // if (error_condition)
    else if (hwrite) //write
```

G5.2.8.2

Table 22: G 5.2.8.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.8.2	Indentation of 1 to 4 spaces per indent; number is consistent.	G	A	2	2	N/A

Comment: All of the RTL(s) strictly followed the above guideline. Hence the Assessment for this guideline is Always. The number of space characters used for indentation is 2.

RMM2 5.2.9 HDL Reserved Words not used in HDL description Max Score 10

R5.2.9.1

Table 23: G 5.2.9.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.9.1	VHDL or Verilog reserved words excluded for names of any elements in your RTL source files.	G	A	2	2	N/A

Comment: All of the RTL(s) seem to follow the above guideline. It is however not possible to check all the keywords because the list of keywords is very exhaustive. But the most common ones were checked for and RTL(s) did not show any occurrence of VHDL keywords in it. Hence the Assessment for this guideline is Always. The number of space characters used for indentation is 2.

RMM2 5.2.10 Port Ordering Max Score 16

R5.2.10.1

Table 24: R 5.2.10.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.10.1	Ports declared in a logical order, consistently within a given design	R	S	10	5	N/A

Comment: As per the RMM2, the ports should be declared in the following order
Inputs:Clocks,Resets, Enables, Other control Signals,Data and address Lines
Outputs: Clocks,Resets,Enables,Other control Signals, Data.
The author of the IP has tried to follow this guideline as much as possible. But at the same time, this guideline has also been violated, in order to group the ports together as per their function. Hence the Assessment for this guideline is Sometimes

G5.2.10.2

Table 25: G 5.2.10.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.10.2	Ports are declared one per line, preferable with a comment following it on the same line.	G	A	2	2	oneperline.pl

Comment: All the RTL(s) have declared only one port per line. Although not always followed by a comment on the same line. But the main point here is it seems that the ports should be one per line, which is found to be followed consistently. A perl script ‘oneperlin.pl’ was written which reported no violations. Hence the Assessment for this guideline is Always. Here is some example code showing that indeed only one port is declared per line.

```
input [1:0] htrans;
input [2:0] hsize;
input       hready;
input [31:0] haddr;         // AHB address bus bits.
input [31:0] hwdata;
```

G5.2.10.3

Table 26: G 5.2.10.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.10.3	Ports declared per recommended order	G	S	2	1	N/A

Comment: The RTL(s) of the IP, follow this guideline to a certain extent. Although the clocks are declared first, followed by reset. But next in line are the ‘enable’ signals, which does not appear in the desired order. For example in the file *‘ahb_external_memory_control.v_rtl’* the ports *‘enable’, ‘mem_chip_enable_n_o’, ‘mem_output_enable_n_o’, ‘mem_write_enable_n_o’* which are enable signals are declared after the signals like *‘read_wait_state0’, ‘read_wait_state1’, read_wait_state2’*, which are control signals.
Hence the Assessment for this guideline is Sometimes..

G5.2.10.4

Table 27: G 5.2.10.4

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.10.4	Comments used to describe groups of ports.	G	A	2	2	N/A

Comment: All RTL(s) follow this guideline.Following are some examples quoted from the RTL(s).

```
// AHB Inputs
//-----------

input     hclk;          // AHB system clock. Only the rising edge of
// this clock is used throughout the module.
```

```
 input     hreset_n;         // Active low AHB sychronous reset.

 input [3:0] hsel_mem;    // Active high AHB memory bank select.
// MEM3, MEM2, MEM1, MEM0.

 input       hwrite;     // AHB transfer direction indicator. High
// for write cycle, low for read cycle.

 input [1:0] htrans;     // AHB transfer typ

// AHB Outputs
//------------

 output      hready_resp;        // AHB hready output.
 reg         hready_resp;

 output [1:0] hresp;      // AHB response. This module only provides.
 reg [1:0]    hresp;      // two types of response:
// 00 = OKAY
// 01 = ERROR
// 10 = RETRY - not implemented
// 11 = SPLIT - not implemented

 output [31:0] hrdata;    // APB data output for read cycles.
 reg [31:0]    hrdata;


// Memory Device Outputs
//----------------------


 output [3:0]  mem_chip_enable_n_o; // Active low chip enable signals for
 reg [3:0]     mem_chip_enable_n_o; // external memory devices.
```

Note that there are 3 examples of group of ports i.e ‘AHB inputs’ , ‘AHB Outputs’ and ‘Memory Device Outputs’. Similarly the ports are described under ‘group of ports’ as far as possible in the RTL(s) Hence the Assessment for this guideline is Always

RMM2 5.2.11 Port Maps and Generic Maps Max Score 22

R5.2.11.1

Table 28: RG 5.2.11.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.11.1	VHDL rule Not applicable	R	N/A	10	10	N/A

R5.2.11.2

Table 29: R 5.2.11.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.11.2	If Verilog, always use explicit connection for ports using named association rather than positional association.	R	A	10	10	N/A

Comment: There are instatiations of objects in the top level verilog file. It strictly follows this rule as the port mapping is done using named association, as it is shown below.

```
ahb_external_memory_registers #(tm_prop)
i_ahb_ext_mem_reg
  (.hclk          (hclk),
   .hreset_n      (hreset_n),
   .hsel          (hsel_reg),
   .hwrite        (hwrite),
   .htrans        (htrans),
   .hsize         (hsize),
   .hready        (hready),
   .haddr         (haddr),
   .hwdata        (hwdata),
   .hready_resp   (hready_reg),
   .hresp         (hresp_reg),
   .hrdata        (hrdata_reg),
   .enable        (enable),
   .read_only     (read_only),
   .read_wait_state0 (read_wait_state0),
   .read_wait_state1 (read_wait_state1),
   .read_wait_state2 (read_wait_state2),
   .read_wait_state3 (read_wait_state3),
   .write_wait_state0 (write_wait_state0),
   .write_wait_state1 (write_wait_state1),
   .write_wait_state2 (write_wait_state2),
   .write_wait_state3 (write_wait_state3)
   );
```

```
i_ahb_ext_mem_con
  (.hclk              (hclk),
   .hreset_n          (hreset_n),
   .hsel_mem          (hsel_mem),
   .hwrite            (hwrite),
   .htrans            (htrans),
   .hsize             (hsize),
   .hready            (hready),
   .haddr             (haddr),
   .hwdata            (hwdata),
   .enable            (enable),
   .read_only         (read_only),
   .read_wait_state0  (read_wait_state0),
   .read_wait_state1  (read_wait_state1),
   .read_wait_state2  (read_wait_state2),
   .read_wait_state3  (read_wait_state3),
   .write_wait_state0 (write_wait_state0),
   .write_wait_state1 (write_wait_state1),
   .write_wait_state2 (write_wait_state2),
   .write_wait_state3 (write_wait_state3),
   .hready_resp       (hready_mem),
   .hresp             (hresp_mem),
   .hrdata            (hrdata_mem),
   .mem_chip_enable_n_o    (mem_chip_enable_n_o),
   .mem_output_enable_n_o  (mem_output_enable_n_o),
   .mem_write_enable_n_o   (mem_write_enable_n_o),
   .mem_byte_enable_n_o    (mem_byte_enable_n_o),
   .mem_address_o     (mem_address_o),
   .mem_datain_i      (mem_datain_i),
   .mem_dataout_o     (mem_dataout_o),
   .mem_invertbits_i  (mem_invertbits_i), //Added by SO on 2/8/01
   .mem_invertbits_o  (mem_invertbits_o), //Added by SO on 1/8/01
   .mem_dataout_en_o  (mem_dataout_en_o)
   );
```

Hence the Assessment for this guideline is Always

G5.2.11.3

Table 30: G 5.2.11.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.11.3	Blank line between the input and output ports to improve readability	G	A	2	2	N/A

Comment: All RTL(s) follow this guideline. There is always a blankline between input and output ports. Hence the Assessment for this guideline is Always. Following lines are quoted as examples form the RTL(s)

```
 output [31:0] hrdata_mem;   // AHB memory data output for read cycles.

//---------------------------------
 input [31:0]  mem_datain_i;       // Memory device read databus.
 input [3:0]  mem_invertbits_i;       // Memory device read for invertbits.


// Memory Device  Outputs
//---------------------------------

// Active low chip enable signals for external memory devices.
 output [3:0]  mem_chip_enable_n_o;
```

RMM2 5.2.12 VHDL Guideline, Not applicable Max Score 2

RMM2 5.2.13 Use Functions Max Score 2
G5.2.13.1

Table 31: G 5.2.13.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.13.1	Functions used whenever possible instead of repeating the same sections of code with comments to explain the function	G	A	2	2	N/A

Comment: There are functions which are being used in the RTL such as ‘bus_invert_coder’, ‘binary2gray32’, and they are used wherever possible, which suggests that the author of the IP has taken care of this guideline. A closer look of the RTL code shows no occurrence of repeated lines of RTL codes which might be put into a single function. Hence the Assessment for this guideline is Always

RMM2 5.2.14 Use Loops and Arrays Max Score 4

G5.2.14.1

Table 32: G 5.2.14.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.14.1	Loops and arrays used for improved read-ability of the source code	G	S	2	1	N/A

Comment: Although there are ample examples of arrays being used in the RTL code. But the Author has missed an opportunity to declare the following 4 ‘regs’ and 4 ‘wires’ as array.

```
wire        mem0_control_reg_sel,
            mem1_control_reg_sel,
            mem2_control_reg_sel,
            mem3_control_reg_sel;



reg [9:0]    mem0_control_reg,
            mem1_control_reg,
            mem2_control_reg,
            mem3_control_reg;
```

Correction: The declaration could have been as shown below:

```
reg [9:0] mem_control_reg [3:0];
wire[9:0] mem0_control_reg_sel;
```

This would have also given the author to use for loops instead of following multiple statements:

```
assign enable[0] = mem0_control_reg[0];
assign enable[1] = mem1_control_reg[0];
assign enable[2] = mem2_control_reg[0];
assign enable[3] = mem3_control_reg[0];
```

the following could have been a replacement code instead of the 4 lines above.

```
reg [9:0] mem_control_reg [3:0]; //delcare the array
```

In an always block use the following code:

```
  for(i=0;i<4;i=i+1)
  begin
    mem_control_temp = mem_control_reg[i];
    enable[i] = mem_control_temp[0];
    read_only[i] = mem_control_temp[1];
  end
```

Similarly for loops could have been used for the lines below.

```
assign read_only[0] = mem0_control_reg[1];
assign read_only[1] = mem1_control_reg[1];
assign read_only[2] = mem2_control_reg[1];
assign read_only[3] = mem3_control_reg[1];


assign read_wait_state0 = mem0_control_reg[5:2];
assign read_wait_state1 = mem1_control_reg[5:2];
assign read_wait_state2 = mem2_control_reg[5:2];
assign read_wait_state3 = mem3_control_reg[5:2];
assign write_wait_state0 = mem0_control_reg[9:6];
assign write_wait_state1 = mem1_control_reg[9:6];
assign write_wait_state2 = mem2_control_reg[9:6];
assign write_wait_state3 = mem3_control_reg[9:6];
```

the following could have been a replacement code instead of

```
for(i=0;i<4;i=i+1)
begin
  mem_control_temp = mem_control_reg[i]
  enable[i] = mem_control_temp[0]
  read_only[i] = mem_control_temp[1]
end
```

Hence the Assessment for this guideline is Sometimes.

G5.2.14.2

Table 33: G 5.2.14.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.14.2	Vector operation on arrays rather than for loops whenever possible.	G	A	2	2	N/A

Comment: The RTL(s) have followed this as far as possible. Following are some examples where vector operations have been done. They could have been done in loops also. Although it seems quite obvious to use vector operations in the following code, but these are quoted here to prove that the above guideline is being followed in RTL. There are no instances of any looping assignments in the RTL(s) which could have been done in vector operations.

```
4'b1000 : hrdata[31:0] <= {22'b0, mem0_control_reg[9:0]};
4'b0100 : hrdata[31:0] <= {22'b0, mem1_control_reg[9:0]};
4'b0010 : hrdata[31:0] <= {22'b0, mem2_control_reg[9:0]};
```

4'b0001 : hrdata[31:0] <= {22'b0, mem3_control_reg[9:0]};

Hence the Assessment for this guideline is Always.

RMM2 5.2.12 Use meaningful lables Max Score 34

R5.2.15.1: VHDL Not Applicable, Score = 10
G5.2.15.2: VHDL Not Applicable, Score = 2

R5.2.15.3

Table 34: G 5.2.15.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.15.3	Each instance labeled with a meaningful name	R	A	10	10	N/A

Comment: There are 2 instances in all in the RTL, and each of them have been given meaningful names as it is evident form the following lines of code:
ahb_external_memory_registers #(tm_prop)
i_ahb_ext_mem_reg
ahb_external_memory_control #(tm_prop, par_little_endian)
i_ahb_ext_mem_con

Hence the Assessment for this guideline is Always

G5.2.15.4

Table 35: G 5.2.15.4

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.15.4	Each Instance labeled U_<name>	G	N	2	0	N/A

Comment: Although the Author has tried to do something like this, but instead of 'U_' he has used 'i_' , so this guideline is not followed. The instantiations are quoted while describing the R5.2.15.3 above, and is not being repeated here.
Hence the Assessment for this guideline is Never

R5.2.15.5

Table 36: R 5.2.15.5

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.2.15.5	Signal, Variable or entity names are not duplicated	G	S	10	8	uniq.pl

Comment : The RTL(s) follow this guideline to a great extent, however, following were 3 violations reported by the script *'uniq.pl'*.
The 3 signals *'hready_resp','hresp', 'hrdata'*, which are declared as 'reg' type appear in both the 2 files: *ahb_external_memory_registers.v_rtl* and *ahb_external_memory_control.v_rtl*.
Hence the Assessment for this guideline is Sometimes

RMM2 5.3 Coding for Portability: Max Score 42

RMM2 5.3.1 For VHDL Only Not Applicable Max Score 24

RMM2 5.3.2 No hare coded Numeric Values Max Score 2

G5.3.2.1

Table 37: G 5.3.2.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.3.2.1	No Hard-Coded Numeric Values in your design (with possible exception of 1 and 0)	G	N	2	0	hard-code.pl

Comment: None of the RTL(s) follow this guideline strictly. All the 3 files use numeric value in abundance. Its very difficult to check this manually, so a script 'hardcode.pl' was used for reporting all the violations form all the files. The log generated by the script is given in the appendix D showing all the violations reported by the script.
This point needs some elaboration. There can be many types of hard coding. In all 4 types of hard-coding were checked in the RTL(s)
1) Using a Numeric value in signal width declaration: Example
reg [7:0] mysig; //'7' is hardcoding
2).Using a Numeric value before the tick(') while assigning a vector type: Example:
mysignal <= 10'd255;// '10' is hardcoding.
3). Using a Numeric value after the tick(') while assigning a vector type: Example:
mysignal <= 10'd100;// '255' is hardcoding.
4) Using a Numeric value to expand a vector: Example:

mysignal <= 100{1'b1};// '100' is hardcoding

All the four types of hardcoding were checked by the script.

Corrections: Following are a few examples which are hard coded, and their corrections.
input [31:0] haddr; //: Not Recommended.
input [wi-1:0] haddr; // Recommended, where 'wi' is a parameter which is equal to 32

Another Example
reg [3:0] reg_addr; //Not Recommended
reg_addr <= 4'b0000;//Not Recommended
reg [width-1:0] reg_addr;//Recommended
reg_addr <= {(width){1'b0}};//Recommended

Hence the Assessment for this guideline is Never

RMM2 5.3.3 For VHDL Only Not Applicable Max Score 2

RMM2 5.3.4 Include Files: Max Score 2

G 5.3.4.1
G5.3.4.1

Table 38: G 5.3.4.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.3.4.1	If Verilog, then keep the 'define statements for a design in a single separate file	G	N	2	0	N/A

Comment: Two out of 3 RTL files use 'define statements locally. Here is an example coded from one of the RTL(s)
'define EMR_IDLE 3'b000
'define EMR_ERROR_WAIT 3'b001
'define EMR_ERROR_READY 3'b010
'define EMR_READ_WAIT 3'b011
'define EMR_READ 3'b100
'define EMR_WRITE 3'b101
Whereas there should be a separate file containing all the 'define statements, as per the guideline.

Hence the Assessment for this guideline is Never

RMM2 5.3.5 Avoid Embedding dc_shell Scripts Max Score 2

G5.3.5.1

Table 39: G 5.3.5.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.3.5.1	No dc_shell scripts in design(except for noted exceptions in RMM2)	G	A	2	2	N/A

Comment: No occurrence of ‘dc_shell’ commands reported in any of the RTL(s). The IP is independent of these.
Hence the Assessment for this guideline is Always

RMM2 5.3.6 Technology-Independent Libraries Max Score 4

G5.3.6.1

Table 40: G 5.3.6.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.3.6.1	Technology Independent Library(e.g DesignWare Foundation Library) used to maintain technology independence.	G	A	2	A	N/A

Comment: Its recommended that there shouldn’t be any instantiations of gates from a library which is technology dependent. However the designer may choose to instantiate gates from a technology independent library, such as G-TECH (synopsys) so that the RTL code remains portable.
However no gate/component instantiations were found form any kind of library in any of the RTL(s), which means that the code is portable, technology independent.
Hence the Assessment for this guideline is Always

G5.3.6.2

Table 41: G 5.3.6.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.3.6.2	No instantiated gates in the design, or module-isolated technology specific gates if instantiated gates are absolutely necessary	G	N	2	0	N/A

Comment: If a designer has to instantiate gates in a design, he should write a isolated module, which will have all the instantiations to make the code portable as far as possible. However in this IP since there are not instantiated gates, it can be said that this guideline was followed strictly. Hence the Assessment for this guideline is Always

RMM2 5.3.7 Coding for translation. for VHDL RTL Not Applicable here Max Score 6

RMM2 5.4 Guidelines for clocks and resets Max Score 38

RMM2 5.4.1 Avoid Mixed clock edges Max Score 24
G5.4.1.1

Table 42: G 5.4.1.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.1.1	Single clock phase flip-flops(eigher +ive or -ive edge) used throughout the design	G	A	2	2	N/A

Comment: All the RTL(s) fo the IP follow this guideline strictly. A positive edge is always used throughout the IP. There are no occurrences of *@ (negedge clk)* in any of the RTLs
Below are some of the examples form RTL files of the IP, which show only *@ (posedge clk)* has been used.

```
always @(posedge hclk)
  begin
  if (~hreset_n)
    begin
      mem0_control_reg[9:0] <= {4'b1111, 4'b1111, 2'b11};
    end // if (~hreset_n)
  else
    begin
      if (mem0_control_reg_sel && write_strobe)
        begin
          mem0_control_reg[9:0] <= hwdata[9:0];
        end // if (mem0_control_reg_sel && write_strobe)
    end // else: !if(~hreset_n)
  end // always @ (posedge hclk)
```

Hence the Assessment for this guideline is Always

R5.4.1.2

Table 43: R 5.4.1.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.1.2	If both positive-edge and negative-edge triggered flip-flops are used, then the worst case duty cycle is modelled for timing analysis and synthesis	R	N/A	10	10	N/A

Comment: Only positive edge used, So not applicable here.

R5.4.1.3

Table 44: R 5.4.1.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.1.4	If both positive-edge and negative-edge triggered flip-flops are used, then the assumed duty cycle is documented for the user.	R	N/A	10	10	N/A

Comment: Only positive edge used, So not applicable here.

G5.4.1.4

Table 45: R 5.4.1.4

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.1.4	If both positive-edge and negative-edge triggered flip-flops are used, then they are separated into different modules.	G	N/A	2	2	N/A

Comment: Only positive edge used, So not applicable here.

RMM2 5.4.2 Avoid Clock Buffers Max Score 2

G5.4.2.1

Table 46: G 5.4.2.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.2.1	No clock buffers in design; inserted after synthesis in physical design stage	G	A	2	2	N/A

Comment: Clock buffers are usually desired when a single gate is seen to drive high loads. Clock buffers are also used to balance the clock skew throughout the design. But they are never coded in RTL or instantiated in RTL. They are automatically inserted where ever needed in RTL to GDSII flow.
In the given IP, no occurrence of clock buffers are reported.
Hence the Assessment for this guideline is Always

RMM2 5.4.3 Avoid Gated Clocks Max Score 2

G5.4.3.1

Table 47: G 5.4.3.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.3.1	No gated clocks in design	G	A	2	2	N/A

Comment: Gated clocks are useful to save power in an IP. While a block of sequential logic is not active at a point of time, the clock of this block can be shut off, which saves power. In the design flow these can be inserted by the tools.hence manual clock gating is not recommended In the given IP, no occurrence of gated clocks are reported.
Hence the Assessment for this guideline is Always

RMM2 5.4.4 No Internally generated clocks Max Score 2

G5.4.4.1

Table 48: G 5.4.4.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.4.1	No internally generated clocks in design	G	A	2	2	N/A

Comment: In the design flow, all the clocks which are required are generated mostly in ‘analog’ area, where a pll is used to generate all the desired clocks with different frequencies. Hence, it is recommended, that there should not be any internally generated clocks, which usually a problem in DFT.
In the given IP, no occurrence of internal division or multiplication of any clocks are reported. Hence the Assessment for this guideline is Always

RMM2 5.4.5 Gated clocks and low power design Max Score 4

G5.4.5.1

Table 49: G 5.4.5.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.5.1	If a gated clock, or an internally generated clock or reset, must be used, then the clock and/or reset generation circuitry is a separate module at the top level of the design	G	N/A	2	2	N/A

Comment: In the given IP, no occurance of internally generated resets or clocks are reported. So this guideline is not applicable here.

G5.4.5.2

Table 50: G 5.4.5.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.5.1	If design requires a gated clock, then model it using synchronous load registers, as recommended (RMM2, P.95)	G	N/A	2	2	N/A

Comment: In the given IP, no occurance of internally generated resets or clocks are reported. So this guideline is not applicable here.

RMM2 5.4.6 Avoid Internally generated resets Max Score 4

G5.4.6.1

Table 51: G 5.4.6.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.6.1	No internally generated, conditional resets. Entire macro resets at one time	G	N/A	2	2	N/A

Comment: In the given IP, no occurance of internally generated resets are reported.
So this guideline is not applicable here.

G5.4.6.2

Table 52: G 5.4.6.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.4.6.1	If a conditional reset is required, then create a separate signal for the reset line and isolate its generating logic in a separate module	G	N/A	2	2	N/A

Comment: In the given IP, no occurance of internally generated resets are reported.
So this guideline is not applicable here.

RMM2 5.5 Coding for Synthesis Max Score 50

RMM2 5.5.1 Infer Registers Max Score 2
G5.5.1.1

Table 53: G 5.5.1.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.1.1	Technology-independent RTL style infers registers(flip-flops) for sequential logic	G	A	2	2	N/A

Comment: RTL coding style should be such that, it should be independent of any type of flip-flops. It should be left up to the synthesis tool to decide what flip-flops will fit for the RTL provided.

This guideline has been strictly followed by the RTL(s) as we do not see any instantaitions for flip-flop types in the design.
Hence the Assessment for this guideline is Always

RMM2 5.5.2 Avoid Latches Max Score 2
R5.5.2.1

Table 54: R 5.5.2.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.2.1	No Latch inference in RTL, especially avoiding inferring R-S latches.	R	A	10	10	N/A

Comment: RTL coding style should be such that, there should be no unintentional latches inferred by the synthesis tools. Usually an combinational process with ‘if’ or ‘case’ statement, within a combinational procedure, when not written carefully results in unwanted latches. That is to say that if in a combinational process, an *‘if’* statement has not been given an *‘else’* statement, or in a *‘case’* statement, if all the possible choices of *‘case’* are not mentioned and there is no *‘default’* corresponding to the case, then the synthesis tool tries to preserve the values, as it has no info as what to do, and it puts latches in the design. For example:

```
always @ (mysignal1 or enable)
begin
  if((mysignal1) && (enable))
  begin
    output1 <= mysignal1;
  end
end
```

In the above example, the synthesis tool has no info on what to do if the condition *if((mysignal1) && (enable))* is false, and it tries to preserve the value assigned to *‘output1’* when the above said condition is true. To preserve the values, it puts a latch.

This guideline has been strictly followed by the RTL(s) as every *‘if’* statement has and *‘else’* and every *‘case’* statement has a *‘default’*
Hence the Assessment for this guideline is Always

G5.5.2.2

Table 55: G 5.5.2.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.2.2	Consistent coding techniques as recommended to avoid latch inference (RMM2, P.100)	G	A	2	2	N/A

Comment. Please see sec 5.2.2.1 above.
Since in every combinational process/block, output is specified for all input conditions, it can be said that the IP follows this guideline strictly.
Hence the Assessment for this guideline is Always

RMM2 5.5.4 Avoid combinational feedback Max Score 2

G5.5.1.1

Table 56: G 5.5.4.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.4.1	No combinational feedback; that is, the looping of combinational process.	G	A	2	2	N/A

Comment: Feedback in the design without any register/latch are not recommended. All the RTL(s) follow this guideline strictly. There are no combinational feedback reported in any of the RTL of the given IP.
Hence the Assessment for this guideline is Always

RMM2 5.5.5 Specify Complete Sensitivity lists Max Score 2

G5.5.1.1

Table 57: R 5.5.5.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.5.1	Complete sensitivity list in each porcess(VHDL) or always(Verilog) blocks	R	A	10	10	N/A

Comment: Sensitivity lists are very important to simulation tools, where as synthesis tools completely ignore them. So in order to match post synthesis simulations to RTL simulations, it is recommended that all the signals/variables which are read in a given ‘always’ block, must be present in sensitivity list of that ‘always’ block. For example: consider an always block shown below

always
begin

myoutput <= (input1 || input2) && (input3 || input4)
end
will synthesize correctly, where as when it will be simulated, 'myoutput' will never change. Its because this 'always' block reads 4 inputs and none of them are present in the sensitivity list of the 'always' block. So there will be a potential mismatch between pre and post synthesis simulations.
In the given IP, no incomplete sensitivity lists are reported.
Hence the Assessment for this guideline is Always

G5.5.5.2

Table 58: G 5.5.5.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.5.2	Only necessary signals in each process sensitivity lists, as defined (RMM2, P.105)c	G	A	2	2	N/A

Comment: If a process(VHDL) or a 'always' block in verilog is sequential, then the sensitivity list should ONLY contain clock signal and a reset signal. Reset signal can only be present if it is a async reset.
In the given IP, this guideline is followed strictly. The IP does not make use of any async reset, so all the sequential process have ONLY 'hclk' in their sensitivity lists.
Hence the Assessment for this guideline is Always

RMM2 5.5.6 Blocking and Non-Blocking Assignments Max Score 2
R5.5.6.1

Table 59: R 5.5.6.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.6.1	If Verilog, then nonblocking assignments always used in always @ (posedge clk) blocks for synthesis	R	A	10	10	blocking.pl

Comment: Out of the two types of assignment statements in verilog, blocking and non blocking assignments, it is recommended that always use non-blocking assignments in any process/block which is sensitive to clock. i.e that synthesizes in clocked registers. A perl script was used to check this called *'blocking.pl'* which reported no occurrence of blocking assignments in any *'always'* block which is sensitive to the system clock *'hclk'*.
This guideline has been strictly followed by the RTL(s) : Here are some examples form the RTL(s) which shows that this guideline has been followed.
always @ (posedge aviclock)

```
  begin
    current_state <= #tm_prop next_state;
    if(hreset_n == 0)
      begin
        current_state <= #tm_prop 'ST_IDLE;
      end
  end
always @(posedge hclk)
  begin
  if (~hreset_n)
    begin
      mem0_control_reg[9:0] <= {4'b1111, 4'b1111, 2'b11};
    end // if (~hreset_n)
  else
    begin
      if (mem0_control_reg_sel && write_strobe)
        begin
          mem0_control_reg[9:0] <= hwdata[9:0];
        end // if (mem0_control_reg_sel && write_strobe)
    end // else: !if(~hreset_n)
  end // always @ (posedge hclk)
```

We can see that assignment has always been done using '<=' which signifies a non-blocking assignment, which is recommended. No occurrence of '=' which signifies a blocking assignment which is not recommended.
Hence the Assessment for this guideline is Always

RMM2 5.5.7 Infer Registers Max Score 2
G5.5.7.1

Table 60: G 5.5.7.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.7.1	If VHDL, then signals used instead of variables for synthesizable RTL	G	N/A	2	2	N/A

Comment: For VHDL not applicable here.

RMM2 5.5.8 Case Statement instead of if-then-else Statements Score 2

G5.5.8.1

Table 61: G 5.5.8.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.8.1	Case statements used rather than an if-then-else statement wherever appropriate	G	S	2	1	N/A

Comment: It is recommended that while writing a conditional statement, a ‘case’ statement is used instead of several ‘if-then-else’. A vector is formed by concatinating the variables on which the output depends, and then the newly formed vector is used in a case statement to cover all the possible input conditions. For example consider the following if statement

```
if(myvar1)
  output1 = input1;
else if(myvar2)
  output1 = input2;
else if(myvar3)
  output1 = input3;
else if(myvar4)
  output1 = input4;
else
  output1 =1’bx;
end
```

Now using this has a disadvantage that input1 is prioritised against all other inputs, *input2* is priortised against *input2* and *input3*, *input3* is prioritised against *input4*, and *input4* gets least priority. Unless the priorities are deliberately needed, we would use the following ‘case’ statement to implement this:

```
case({myvar1,myvar2,myvar3,myvar4})
  4’b1000: output1=input1;
  4’b0100: output1=input2;
  4’b0010: output1=input3;
  4’b0001: output1=input4;
  default: output1=1’bx;
endcase
```

this will give equal priority to *‘input1’,’input2’.’input3’,’input4’*.
Post synthesis of the ‘if’ implementation will result in a longer critical path as shown in Figure 1, whereas post synthesis of the ‘case’ implementation will result in a shorter critical path as shown in Figure2. Also ‘if’ implementation is prioritised, whereas ‘case’ statement implementation is parallel.

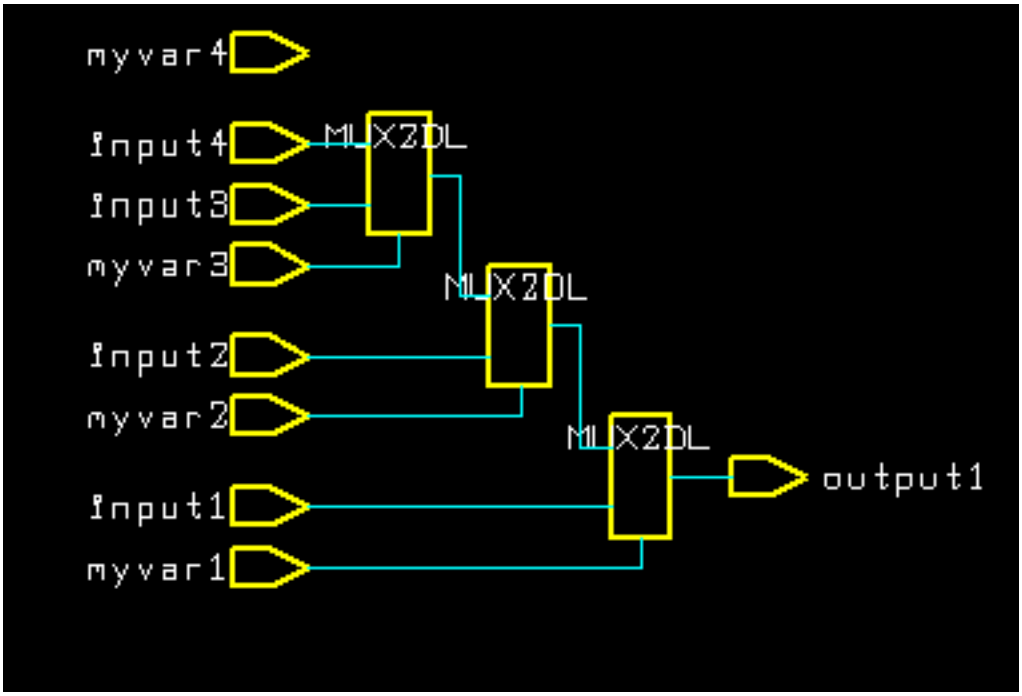


Figure 1: ‘if’ statement Implementation. Note that myvar4 is not used.

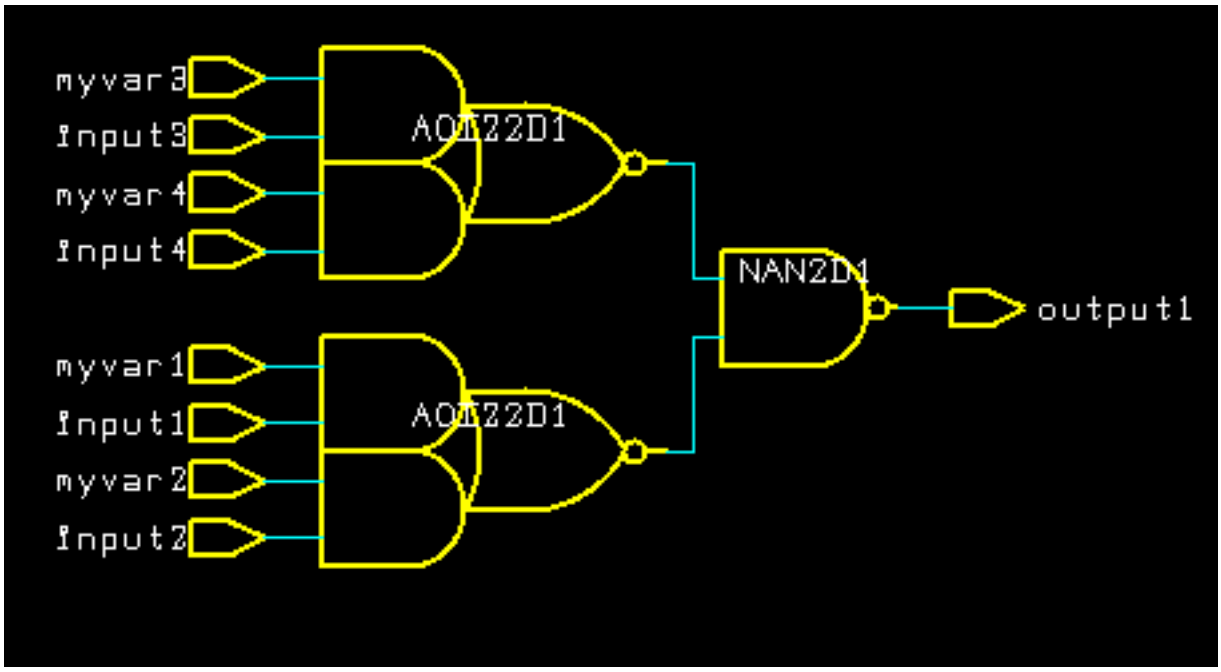


Figure 2.: ‘case’ statement Implementation.

This guideline has been followed by the RTL(s) but not always. Here are some examples where ‘else if’ can be noticed, which may have been written using case statement.

```
if (error_condition)
 begin
  slave_state <= ‘EMR_ERROR_WAIT;
  hready_resp <= 1’b0;
  hresp[1:0]  <= 2’b01;
 end // if (error_condition)
else if (hwrite) //write
 begin
  slave_state    <= ‘EMR_WRITE;
  hready_resp    <= 1’b1;
  hresp[1:0]     <= 2’b00;
  write_strobe   <= 1’b1;
  reg_addr[3:0]  <= haddr[3:0];
 end // if (hwrite)
else
 begin
  slave_state    <= ‘EMR_READ_WAIT;
  hready_resp    <= 1’b0; // wait state
  hresp[1:0]     <= 2’b00;
  read_strobe    <= 1’b1;
  reg_addr[3:0]  <= haddr[3:0];
 end // else: !if(hwrite)
```

Hence the Assessment for this guideline is Sometimes

RMM2 5.5.9 Infer Registers Max Score 2
G5.5.9.1

Table 62: G 5.5.9.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.9.1	HDL description for state machines separated into two processes, one for the combinational logic and one for the sequential logic	G	S	2	1	N/A

Comment: The way state machines should be coded in RTL is such that, the combinational part should be made independent to the sequential part as far as possible. That is there should be different processes for sequential part and the combinational part. This gives a better chance to the synthesis tool for better optimization. Figure 3 below shows the structure clearly.

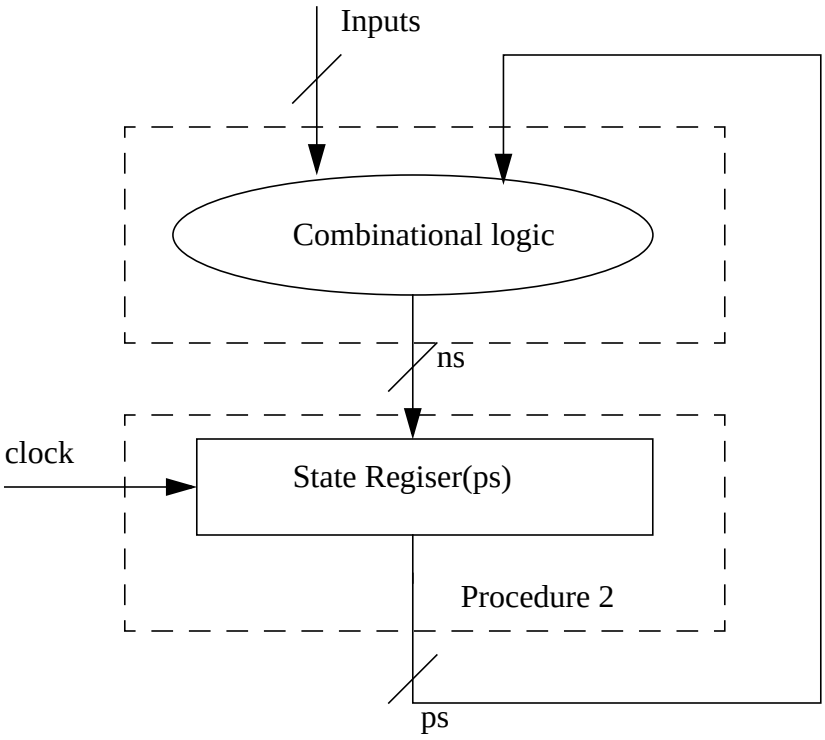


Figure 3 Recommended structure of a State Machine coded/realized in RTL

Procedure 1 should be coded like this:

```
always @ (all_inputs or ps)
begin
  ns <= f(ps,inputs);
end
```

Where f(ps,inputs) means function of ‘ps’ i.e present state, and ‘inputs’ i.e all the inputs to the state machine

Procedure 2 should be code like this:

```
always @ (posedge clk)
begin
  ps <= ns;
end
```

Where ‘ps’ is the present state of the state registered, and ‘ns’ is the calculated next state that would be loaded in the present state register at each clock edge.

Not all the state machines in the given IP, follow this guideline.

Following is an example where this guideline is followed.

```
always @ (b_error_condition or b_transfer_on_bus
        or b_wait_states_left or current_state or transfer_type)
  begin
    next_state <= ‘ST_ERROR_START;
    case (current_state)
     .
     .
```

The above code corresponds to Procedure1, in figure 3.

Following is an example where this guideline is NOT followed:

```
// State update
// -------------------------------------------------------------------
always @ (posedge hclk)
  begin
    current_state <= #tm_prop next_state;
    if(hreset_n == 0)
      begin
        current_state <= #tm_prop ‘ST_IDLE;
      end
  end
```

The above code corresponds to Procedure2, in figure 3.

```
always @(posedge hclk)
  if (~hreset_n)
    begin
      slave_state     <= ‘EMR_IDLE;
      hready_resp     <= 1’b1;
      hresp[1:0]      <= 2’b00;
      read_strobe     <= 1’b0;
      write_strobe    <= 1’b0;
      reg_addr        <= 4’b0000;
    end
  else
    case (slave_state)
      ‘EMR_IDLE:
        //
        // When a valid bus cycle is taking place (htrans not
        // IDLE or BUSY) on the AHB, if the hsel signal is
        // asserted when hready is high then the target for the
        // bus cycle is the AHB slave.
        //
        // If an error condition exists then the transfer is
        // not allowed and an ERROR response will be issued.
        // Otherwise the transfer must be a valid read or
        // write cycle, and is handled accordingly.
```

```
//
// The state machine remains in the IDLE state while no
// transfers are pending.

begin
  if (hsel && hready && (htrans[1] == 1'b1))
    // Transfer request currently on the bus
    if (error_condition)
      begin
        slave_state <= 'EMR_ERROR_WAIT;
        hready_resp <= 1'b0;
        hresp[1:0]  <= 2'b01;
      end // if (error_condition)
    else if (hwrite) //write
```

We can clearly see that this is an implicit state machine. There is a single sequential procedure as opposed to 2 procedures which is not in line with what is recommended in this guideline.
Hence the Assessment for this guideline is Sometimes

G5.5.9.2 VHDL guideline, Not applicable here

G5.5.9.2.a

Table 63: G 5.5.9.2a

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.9.2a	In Verilog, use 'define statements to define the state vector..	G	A	2	2	N/A

Comment: All RTL(s) follow this guideline strictly. 'define statements are used to define the state vector always. As an example, following code is re-produced here showing that the guideline is indeed followed.

```
'define ST_IDLE        6'b000001
'define ST_ERROR_START  6'b000010
'define ST_READ_WAIT    6'b000100
'define ST_WRITE_ADDR   6'b001000
'define ST_WRITE_WAIT   6'b010000
'define ST_READ_ADDR    6'b100000
```

Hence the Assessment for this guideline is Always

G5.5.9.3

Table 64: G 5.5.9.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.9.3	FSM logic and non-FSM logic separated into different modules.	G	N	2	0	N/A

Comment: 2 out of 3 RTL files has state machines in them, and both of the modules if, they have to follow this guideline, must not have anything else in them. All the logic apart from the FSM must be done in new module(s). It is also seems difficult and may be a bit un-necessary to follow this guideline, because this will need a creation of another module for no good reason other than just following this guideline.
Since this guideline is not followed in the given IP, the Assessment for this guideline is Never

G5.5.9.4

Table 65: G 5.5.9.4

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.5.9.4	Assign a default state for the state machine	G	A	2	2	N/A

Comment: A ‘default’ assignment is always expected in a state machine, for the state vector, which facilitates to define the power up value of the state-vector. There are 2 FSMs in the design, and a ‘default’ assignment is seen in both of them. Following is the code re-produced here from the file ‘*ahb_external_memory_control.v_rtl*’ showing the ‘default’ assignment. Note that the use of a verilog ‘default’ key word is **not** being made here, yet there is a default assignment. The ‘**bold**’ text highlights the line which does the ‘default’ assignment.

```
always @ (b_error_condition or b_transfer_on_bus
        or b_wait_states_left or current_state or transfer_type)
  begin
    next_state <= 'ST_ERROR_START;
    case (current_state)

      // -------------------------------------------------------------
      // Idle
      // -------------------------------------------------------------
      'ST_IDLE: begin
```

While the code from the other file ‘*ahb_external_memory_registers.v_rtl*’, shown below, in fact uses verilog ‘default’ key word for the default assignment.

```
default:
  // The default case is included to ensure correct
  // recovery from illegal states.

  begin
    slave_state <= ‘EMR_IDLE;
    hready_resp <= 1’b1;
    hresp[1:0]  <= 2’b00;
  end // case: default
// surefire coverage_on
//verisureon
```

Hence the Assessment for this guideline is Always

RMM2 5.6 Partitioning for Synthesis Max Score 22

RMM2 5.6.1 Register all outputs Max Score 2
G5.6.1.1

Table 66: G 5.6.1.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.2.1	For each block of a hierarchial design, all output signals from the block come directly from registers	G	A	2	2	N/A

Comment: The RTL should be written such that, all outputs from it must directly come out from registers. This will then ensure that all outputs from the block are glitch free, and have a defined relationship with the clock. All RTL(s) in the IP follow this guideline strictly. Here is an examples of output port which are seen coming directly from an register. This example shows that a port ‘*read_only’*, which is connected directly to ‘*mem0_control_reg[1]’*, which in turn is being updated in a clocked procedure. This example is deliberately chosen, to show that it is not necessary to declare a port as a ‘reg’ type(every port is a ‘wire’ type by default in verilog, unless a ‘reg’ is declared using the same name as of the port), if it needs to be coming from a register.

```
output [3:0]  read_only;    // Active high level outputs. These are used
assign read_only[0] = mem0_control_reg[1];
always @(posedge hclk)
  begin
  if (~hreset_n)
    begin
      mem0_control_reg[9:0] <= {4’b1111, 4’b1111, 2’b11};
    end // if (~hreset_n)
  else
```

```
  begin
    if (mem0_control_reg_sel && write_strobe)
      begin
        mem0_control_reg[9:0] <= hwdata[9:0];
      end // if (mem0_control_reg_sel && write_strobe)
  end // else: !if(~hreset_n)
end // always @ (posedge hclk)
```

No occurrence of any output port in RTL(s) are reported with do NOT come directly from clocked register.
Hence the Assessment for this guideline is Always

RMM2 5.6.2 Related Combinational logic in a single module Max Score 2
G5.6.2.1

Table 67: G 5.6.2.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.2.1	Related combinational logic placed together in the same module	G	A	2	2	N/A

Comment: To give the synthesis tool a better chance to epitomize the combinational logic, it is recommended that all the related combinational logic should be placed together in a single module. In the given IP only one file '*ahb_external_memory_control.v_rtl*' has combinational logic inside it, which suggests that the author has followed this guildeline strictly.
Hence the Assessment for this guideline is Always

RMM2 5.6.3 Separate Modules That Have Different Design Goals Max Score 2
R5.6.3.1

Table 68: G 5.6.3.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.3.1	Critical path logic isolated in a separate module from non critical path logic	G	N/A	2	2	N/A

Comment: To give the synthesis tool, different constraints for different requirements, it is recommended that the part of design which is timing critical should be separated in a different module with the part of the design which might be area critical. The area critical part can be kept separate module, so that the designer can assign different constraints to different modules as per the reqiurement. If this guideline is followed, then the timing critical module can be epitomized for speed,

which might require more area. Other parts of the design can then be epitomized for area separately, giving a better design overall.
The given IP doesn't seem to have very long critical paths. No multiplication or addition or any other arithmetic function is seen in the IP, which suggests that this guideline is more or less not applicable here.
Hence the Assessment for this guideline is Not Applicable.

RMM2_5.6.4 Asynchronous Logic Max Score 2
G5.5.2.1

Table 69: G 5.6.4.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.4.1	Asynchronous logic is avoided	G	A	2	2	N/A

Comment: No Async logic in the design. ALL RTL(s) follow this guideline strictly.
Hence the Assessment for this guideline is Always
G5.5.4.2

Table 70: G 5.6.4.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.4.2	If Asynchronous logic is used, then it is partitioned into a separate module and modelled in a behavioural rather than a structural style as suggested(RMM, P.93)e	G	N/A	2	2	N/A

Comment: No Async logic in the design.
Hence the Assessment for this guideline is Not Applicable.

RMM2_5.6.5 Arithmetic Operators: Merging resources Max Score 2
G5.6.5.1

Table 71: G 5.6.5.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.5.1	Partition arithmetic equations to leverage automatic resource sharing in synthesis	G	N/A	2	2	N/A

Comment: No arithmetic operators found in the IP.

Hence the Assessment for this guideline is Not Applicable.

RMM2_5.6.7 Avoid Point-to_poing Exceptions and False Paths Max Score 8

G5.6.7.1

Table 72: G 5.6.7.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.7.1	No Multicycle paths in your design	G	A	2	2	N/A

Comment: Consider a simple block of combinational logic between two registers R1 and R2, with the delay T1, where the time period of the clock is Tclk. In some cases, if T1>T2, we may be able to get away with this timing violation by declaring the path between R1 and R2 as what is called a multicycle path. This of course requires that the output of R2 is not used in each clock cycle. A multicycle path is a risky design practice, which can easily produce serious errors if it is not done very carefully. The guideline recommends that these should simply be not present. In the given IP, there is no comment in the RTL or a mention of multicycle path in the documentation.So we can say that there are no multicycle paths in the given IP.

Hence the Assessment for this guideline is Always

G5.6.7.2

Table 73: G 5.6.7.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.7.2	If a multicycle path must be used, then point-to-point exceptions maintained within a single module and well-commented	G	N/A	2	2	N/A

Comment: Since there are no multicycle paths in the given IP, the above guideline does not apply here.

Hence the Assessment for this guideline is Not Applicable.

G5.6.7.3

Table 74: G 5.6.7.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.7.3	No false paths in the design	G	A	2	2	N/A

Comment: A false path is a datapath, which is effectively not active in a design. An example of false path is shown below:

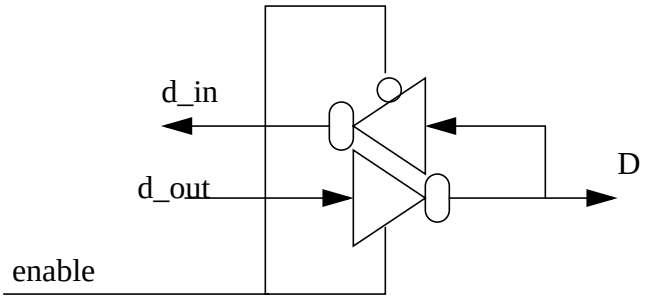


Figure 4. The path shown above from d_out to d_in is a false Path

In the figure 4 shown above, the path from *d_out* to *d_in* will never be active in the design, because only one of the tristate buffers can be enabled at a given time. But the static timing analysis tool has no way of knowing this automatically. These kind of paths are called false paths, and must be explicitly told to the STA tool, wherever they are present. There may be several other kinds of false paths in the design, the above example is just one of those. Again using them in the design is risky, as inappropriate use can easily result in setting a 'false_path' to a path which is not actually a false path. The guideline says simple avoid them. Since there is no mention of any false path either in the document or in the RTL or synthesis scripts, we can say that this guideline has been strictly followed. It might be that there has been no need of declaring false paths, and this is just a coincidence that there are no paths which are declared as false, but for the IP evaluation purposes, there is no false paths no matter what the reason is.

Hence the Assessment for this guideline is Always

G5.6.7.4

Table 75: G 5.6.7.4

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.7.4	If a false path exists, it is documented*	G	N/A	2	2	N/A

Comment: No false paths reported either in the document, RTL or synthesis scripts.
Hence the Assessment for this guideline is Not Applicable.

RMM2 5.6.8 Eliminate Glue Logic Max Score 2

G5.6.8.1

Table 76: G 5.6.8.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.6.8.1	No instantiated gate-levle logic at the top level of the design hierarchy	G	A	2	2	N/A

Comment: The top level RTL file is not expected to have gate level instantiations. This is because this kind of gate level logic inhibits the proper optmization by the synthesis tool.
There is no glue logic in the top level file 'ahb_external_memory.v_rtl'
Hence the Assessment for this guideline is Always

RMM2 5.7 Designing with Memories Max Score 2

Table 77: G 5.7

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.7.1	Address and data registers and the write enable logic partitioned into separate modules	G	A	2	2	N/A

Comment: IF an IP is written to deal with memories then in order that the IP works with both types of memories sycn and async, it is recommended that the write enable logic is in a different module to the module haveing address and data registers.The given IP does deal with memories. The registers are in one module '*ahb_external_memory_registers*' and the control(i.e wiret enable etc) are in a different module called '*ahb_external_memory_control*'
Hence the Assessment for this guideline is Always

RMM2 5.8 Code Profiling Max Score 2

Table 78: G 5.8

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
5.8.1	Use code profiling to improve RTL	G	A	2	2	N/A

Comment: After an RTL is written, simulation is performed to check the functionality of the design. But it is often advised profiling tools, which can measure the RTL code coverage using

your testbenche(s) or testvector(s). These tools report the frequency at which a line in RTL code is hit using your testvectors. Now if the frequency comes out to be 0 for any line, then either this line is redundat or the test vectors are not exhaustive enouth. This information can give the designer a hint of any problems that might be there in the IP.
The RTL code does suggest that code profiling is used. The comments like *verisureoff and* ver-isureon suggests, that the author of the IP has take proper care of improving the code coverage. A script is also provided for measuring the code coverage called '*rtl_coverage.script*', however due to the expired license of Verisure(from Verisity) the script did not run, and code coverage results cannot be analysed.
Hence the Assessment for this guideline is Always

2.3.2a: Comment upon overall results and quality of RTL

The total Score obtained from the openmore excel sheet for this section "RTL Coding Guidelines" is evaluated to be **201/240 = 83.75 %.**
Over the quality of RTL was found to be OK. I will discuss the negative and positive remarks here. First I will discuss negative remarks and finish off positive remarks will be discussed.

Negative Remarks:
1). Use of constants in RTL for conditional assignments:
There exists a serious problem in the RTL which is unprofessional, and which may jeopardise the market-ability of the RTL. The point brought here is **very important** to improve the quality of the RTL. That is the way constants are being used in the RTL of the given IP, and assigning signals depending upon the value of those constants. The following 'always' procedure is re-produced here which shows the problem.

```
always @ (haddr or hsel_mem)
  begin
    if ('ADDRESS_LOW_POWER_SELECT_0 ==1 && hsel_mem[0] == 1)
      begin :blk_low_power_adress_cap0
        if ('DATA_SIZE_0==001)
          begin
            haddr_gray_capture <=  binary2gray32(haddr); //Gray coding for 16 bit addressing
          end // if ('DATA_SIZE_0==001)
        else if ('DATA_SIZE_0==010)
          begin
            haddr_gray_capture <=  binary2gray16(haddr); //Gray coding for 16 bit addressing
          end // if ('DATA_SIZE_0==010)
        else if ('DATA_SIZE_0==100)
          begin
            haddr_gray_capture <=  binary2gray8(haddr); //Gray coding for 8 bit addressing
          end // if ('DATA_SIZE_0==100)
        else
          begin
            haddr_gray_capture <=  haddr;
          end
      end // block: blk_low_power_adress_capt0
```

```
    else if (‘ADDRESS_LOW_POWER_SELECT_1 ==1 && hsel_mem[1] == 1)
.
.
.
.
..
  end // always @ (haddr)
```

Now, we can see the use of two constants in the above procedure i.e ADDRESS_LOW_POWER_SELECT_0 and DATA_SIZE_0, and depending upon the value of these constants the author is trying to assign the signal ‘*haddr_gray_capture*’. This is never done in any professional IP. It doesn’t make any sense.
The problem here is to make the same IP useable in different scenarios. The proper way of doing this is given below:

First write a generic function binary2gray, instead of 3 different functions binary2gray8, binary2gray16, binary2gray32. Parameterise the function, and put it in a separate module. as given below:

```
module bin2gray(haddr_bin, haddr_gray);
 parameter width = 32;
 parameter data_size = 2;
 input [width-1:0] haddr_bin;
 output [width-1:0] haddr_gray;


function [width-1:0] binary2gray;
input [31:0] haddr_bin;    // Binary code address from amba ahb bus
reg [width-1:0]   haddr_gray;
integer i;

begin
  haddr_gray = {(width + 1){1’b0}};//Design Compiler complains that variable is not initialised
                // if I don’t put this in

  haddr_gray[width-1] = haddr_bin[width-1];
  for(i = 0; i < data_size; i = i + 2) //i+2 because it should exit out of loop after 1 itration
    haddr_gray[data_size-1] = haddr_bin[data_size-1];
  for(i = 1; i < data_size; i = i + 1)
    haddr_gray[data_size-2] = haddr_bin[data_size-2]^haddr_bin[data_size-1];
  for(i = data_size; i < width-1; i = i + 1)
    haddr_gray[i] = haddr_bin[i] ^ haddr_bin[i + 1];

  binary2gray = haddr_gray;
end
endfunction //binary2gray
```

assign haddr_gray = binary2gray(haddr_bin);

endmodule

Then use this function/module as shown below instead of the long not very useful 'always' block shown above

'ifdef ADDRESS_LOW_POWER_SELECT_2
bin2gray #(width, data_size) bin2gray_u0(.haddr_bin(haddr_bin), .haddr_gray(haddr_out));
'else
assign haddr_out = haddr_bin;
'endif

The first line '*ifdef ADDRESS_LOW_POWER_SELECT_2*' checks if somewhere the word '*ADDRESS_LOW_POWER_SELECT_2*' is defined, if yes, then it '*haddr_out*' gets the value from the generic function '*bin2gray*', otherwise '*haddr_out*' gets the value of '*haddr_bin*' directly.Note that it is not required to check the value of constant, just weather it is defined somewhere or not will suffice. So a if a low power solution is required, *ADDRESS_LOW_POWER_SELECT_2* will be defined somewhere in the environment, if it is not defined, then a low power solution will automatically be dropped.

The parameter '*data_size*' replaces the definition of the constant 'DATA_SIZE_2', which is passed to the module containing the function '*binary2gray*' and depending upon the value of the parameter '*data_size*' the generic function '*binary2gray*' actually performs one of the old three functions '*binary2gray8*','*binary2gray16*' or '*binary2gray32*'

We see how the 82 lines of code in the RTL is now changed to 5 lines, which is one of the unobjectionable way to do the things which are desired.

Following are the disadvantages or problems with the coding style used in the IP which are addressed by the alternative way suggested above

- The old code produces several warning messages when read in a synthesis tool, complaining about several branches which it says will 'never be reached'. That is true because constants are constants, they will never change.
- The code coverage will also be greatly reduced because again, several branches will never be hit by any sets of test vectors.
- Meaningless synthesis results may result, because the code does not make sense in terms of hardware mapping.

Conclusion: Never let RTL check a value of a constant and do things depending upon the value of a constant. Its unprofessional and does not make any sense.

There are no other serious negative remarks.

Positive Remarks:

Leaving the point discussed above, the RTL quality is found to be good, with nearly 80% guidelines followed.

Another thing which is pointed out here is the use of comments whenever a 'always' block ends. For example:

always @ (haddr or hsel_mem)

begin

.
.
.

end // always @ (haddr)

Note the use of '*// always @ (haddr)*' where the 'always' block ends. This is a very good practice and makes the code very much more readable. This is an extra feature, which the author as introduced apart from the guidelines, which helps the evaluator a lot.

Decision on IP based on RTL coding guidelines

Purchase of the IP is recommended based on RTL coding guidelines, as long as the problem discussed above(i.e in negative remarks) are removed.

Section 2.3.3 Macro Synthesis Guidelines

RMM2 6.2 Macro Synthesis Stratigy

RMM2 6.2.1 Macro Timing Budget

R6.2.2.1

Table 79: R 6.2.1.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
6.2.2.1	Timing budget for the macro developed as part of the specification process, before the design is partitioned into blocks and before coding has begun.	R	N	10	0	N/A

Going through all the documentation, there were no mention of either timing budget or the clock frequencies. There is no evidence of even the maximum or minimum frequency the design supports in any documentation, however in one of the synthesis script ‘synthesis.prj’ the frequency is mentioned to be 200.00. But it definitely needs to be documented.
Hence the Assessment for this guideline is Never

RMM2 6.2.2 Sub block Timing Budget

R6.2.2.1

Table 80: R 6.2.2.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
6.2.2.1	Timing budget for each subblock in the macro developed at the time the design is partitioned into subblocks, and before coding has begun.	R	N	10	0	N/A

Going through all the documentation, there were no mention of either timing budget or the clock frequencies. There is no evidence of even the maximum or minimum frequency any of the sub block supports
Hence the Assessment for this guideline is Never

RMM2 6.2.4 Sub block synthesis Process 3 phases

G6.2.4.1

Table 81: G 6.2.4.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
6.2.4.1	The following subblock synthesis process is used: 1) Compile subblock, using constraints based on budget; 2) Characterize-compile whole subblock, to refine timing constraints and re-synthesize subblock; 3) Iterate if required.	G	N	2	0	N/A

Going through all the documentation, scripts, it can be said, that the above guideline is not followed. There is no evidence of reading the design block by block in synthesis script. All the blocks are read at once, and one pass synthesis is done.
Hence the Assessment for this guideline is Never

RMM2 6.2.5 Macro Synthesis Process 3 phases

G6.2.5.1

Table 82: G 6.2.5.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
6.2.5.1	The following macro-level synthesis process is used: 1) Compile each of the subblocks, using constraints based on budget; 2) Characterize-compile whole macro to improve area and timing; 3) If necessary, incremental compile performed.	G	N	2	0	N/A

Again same as above.
Hence the Assessment for this guideline is Never

RMM2 6.2.7 Preserve Clock and Reset Networks

G6.2.7.1

Table 83: G 6.2.7.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
6.2.7.1	dont_touch_network specified on clock and asynchronous reset networks and included in the synthesis scripts for the design - dc_shell scripts	G	N/A	2		N/A

No dc_shell scripts provided. The design synthesis is limited to FPGA target only.
Hence the Assessment for this guideline is Not Applicable

RMM2 6.5 Coding Guidelines for Synthesis Scripts

This section is omitted because ASIC synthesis script are not provided for the IP.
Hence the Assessment for this guideline is Not Applicable

Section 2.3.4 Verification Guidelines

RMM2 7 : Macro Verification
RMM2 7.1 Overview of Macro Verification
7.1.3 : Subblock Simulation

R7.1.3.1

Table 84: R 7.1.3.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
7.1.3.1	All response checking is done automatically (not by viewing waveforms)	R	A	10	10	N/A

Comment: Verification strategy seems to be quite exhaustive, as documented in 'Environment_Statigy.pdf'. Although the small blocks does not use the self-checking system, i.e text based system, but on the macro level, the testbench work with self-checking capability. Giving a high degree of satisfaction. Their monitors, Scoreboards and also random test strategy, which all gave text results.
Hence the Assessment for this guideline is Always

G7.1.3.2

Table 85: G 7.1.3.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
7.1.3.2	All subblock test suites achieve 100% statement and path coverage as measured by a test coverage tool.	G	N/A	2		N/A

Comment: Since the license of Verisure(from Verisity) has expired, the script given to do the coverage analysis i.e '*rtl_coverage.script*' did not return results.
Hence the Assessment for this guideline is Not Applicable

RMM2 7.3 RTL Testbench Style Guide

RMM2 7.3.1: General Guidelines

G7.3.1.2

Table 86: G 7.3.1.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
7.3.1.2	Testbench code is partitioned into synthesizable and behavioral sections. Behavioral code is used to generate clocks and resets and synthesizable code is used to model a finite state machine that manipulates and generates stimulus for the design.	G	A	2	2	N/A

Comment: A high degree of partitioned is observed in the test bench. Very clear RTL style code is observed in the testbench file. '*ahb_external_memory.v_tb*'. Clearly clock generation and initial values in 'initial' blocks are separated in the testbench. The rest of the code is RTL style.
Hence the Assessment for this guideline is Always

RMM2 7.3.2: Generating Clocks and Resets

G7.3.2.2

Table 87: G 7.3.2.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
7.3.2.2	Separate processes used for clock generation, data generation, and reset generation.	G	A	2	2	N/A

Comment: Clock and Resets are indeed generated in separate processes. As it is evident from the following lines quoted from the testbench file '*ahb_external_memory.v_tb*'

```
    repeat(6)@(posedge hclk);
    hreset_n <= 1;

 always
   begin
     #(CYCLE/2) hclk <= ~hclk;
   end // always begin
```

Hence the Assessment for this guideline is Always

G7.3.2.3

Table 88: G 7.3.2.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
7.3.2.3	If multiple asynchronous clocks, then a separate process for each clock generation is used.	G	N/A	2		N/A

Comment: No evidence of multiple clocks
Hence the Assessment for this guideline is Not Applicable

G7.3.2.4 : VHDL coding guideline Not Applicable here

G7.3.2.5

Table 89: G 7.3.2.5

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
7.3.2.5	Testbenches read and apply one vector only per clock cycle.	G	N/A	2		N/A

Comment: This guide line needs detailed study of the testbench, so this is omitted from evaluation.
Hence the Assessment for this guideline is Not Applicable

G7.3.2.6

Table 90: G 7.3.2.6

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
7.3.2.6	Clocks used to synchronize stimulus generation. All data applied once every cycle boundary with as few individual process waits as possible.	G	A	2	2	N/A

Comment: The testbench is coded in RTL style. All procedures are clocked, stimulus is therefore generated with respect to the clock. For example the following code reproduced from the testbench file ahb_external_memory.v_tb' which clear shows this:

```
always@(negedge hclk)
  begin
    if(hreset_n == 0)
      begin
        hgrant_m0 <= 1;
```

hgrant_m1 <= 0;
hgrant_m2 <= 0;

Hence the Assessment for this guideline is Always

RMM2 7.5 Timing Verification

7.5.1 : Use static timing analysis for timing verification

Comment: This guideline is followed, as it is evident from the document 'Environment_Statigy.pdf' . The tool used is given to be Synopsys' Prime Time. However the scripts are not given with the IP.
Hence the Assessment for this guideline is Always

System Level Verification

RMM2 11.5 Prototyping

G 11.5.1 The macro has been implemented and verified on FPGA

The IP is basically targeted on FPGA
Hence the Assessment for this guideline is Always

G 11.5.3 The macro is silicon proven at speed

The IP is a soft IP, and is yet to go on silicon
Hence the Assessment for this guideline is Not Applicable

RMM2 11.6 Gate Level Verification

11.6.2 Formal Verification

G11.6.2.1

Table 91: G 11.6.2.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
11.6.2.1	Macro follows naming convention defined on p74, avoiding small or similar name	G	S	2	1	N/A

Comment: This guideline is followed to an extent. There are naming conventions used in verilog coding, which are also documented, but the naming conventions do not match with the ones given in RMM2.
For example RMM2 recommends '*_r' for output of registers. Which is not there in RTL, however, the use of similar type of conventions are present in RTL such as use of '* _o' for output signals.
Hence the Assessment for this guideline is Sometimes

R11.6.2.2

Table 92: R 11.6.2.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
11.6.2.2	The functionality of the macro is not changed by synthesis script	R	A	10	10	N/A

Comment: ASIC synthesis scripts are not provided, however the FPGA scripts are simple and there is no evidence of any commands which can/may change the functionality of the design
Hence the Assessment for this guideline is Always

R11.6.2.3

Table 93: G 11.6.2.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
11.6.2.3	Pragmas (compiler directives) are used in RTL code instead of synthesis script commands	G	A	2	2	N/A

Comment: No such commands present in synthesis scripts
Hence the Assessment for this guideline is Always

G11.6.2.4

Table 94: G 11.6.2.4

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
11.6.2.4	Avoid any complex retiming at gate level	G	A	2	2	N/A

Comment: No evidence suggesting that
Hence the Assessment for this guideline is Always

R11.6.2.5

Table 95: R 11.6.2.5

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
11.6.2.5	Avoid combinational loop	R	A	2	2	N/A

Comment: No combinational feedbacks found in RTL
Hence the Assessment for this guideline is Always

11.6.3: Gate level Simulation:

R11.6.3.1

Table 96: R 11.6.3.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
11.6.3.1	The test bench provided with RTL works on gate level netlist	R	A	10	10	N/A

Comment:*ahb_external_memory.v_tb*: is the testbench file, which is used in both RTL and Gate level Simulations.
Hence the Assessment for this guideline is Always

G11.6.3.2

Table 97: G 11.6.3.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
11.6.2.2	The gate level simulation works with and without SDF	G	N/A	2		N/A

Comment: No sdf provided.
Hence the Assessment for this guideline is Not Applicable

G11.6.3.3

Table 98: G 11.6.3.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
11.6.2.3	If macro has memory elements, all memory elements are reset during reset phase. If not, it is documented and the X propagation is controlled in test plan.	G	N/A	2		N/A

Comment: The IP is targeted on FPGA, which provide a reset for registers automatically. So this guideline is not applicable here
Hence the Assessment for this guideline is Not Applicable

RMM2 11.7 Specialized Hardware for System Verification

11.7.2 RTL Acceleration

G11.7.2.1

Table 99: G 11.7.2.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
11.7.2.1	Avoid collection of small testbenches where the compilation time is larger than the execution time	G	A	2	2	N/A

Comment: One testbench file used. No evidence of small testbenches found
Hence the Assessment for this guideline is Always

11.7.6 Design guidelines for accelerated verification

G11.7.6.7

Table 100: G 11.7.6.7

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
11.7.6.7	Maintain hierarchical, modular design to help reduce routing between processors.	G	A	2	2	N/A

Comment: Design is found to be modular.
Hence the Assessment for this guideline is Always

Section 2.3.4 Deliverable Guidelines

RMM2 Section 9 RMM Deliverables

9.1 Soft Macro Deliverables
9.1.1 Product Files

R9.1.1.1.1

Table 101: R 9.1.1.1.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.1.1	If Verilog, synthesizable Verilog RTL Source for the macro and its subblocks.	R	A	10	10	N/A

Comment: Synthesis Verilog present.
Hence the Assessment for this guideline is Always
R9.1.1.1.2: VHDL guideline Not Applicable

R9.1.1.1.3

Table 102: R 9.1.1.1.3

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.1.3	Application Notes with VHDL and Verilog Design Example.	R	A	10	10	N/A

Comment: Application Notes present: Documents like *external_memory_programmers_guide.pdf* provide them
Hence the Assessment for this guideline is Always
R9.1.1.1.4

Table 103: R 9.1.1.1.4

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.1.4	Synthesis scripts & timing constraints	R	A	10	10	N/A

Comment: Synthesis scripts & timing constraints file present '*synthesis.prj*'
Hence the Assessment for this guideline is Always

R9.1.1.1.5

Table 104: R 9.1.1.1.5

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.1.5	Scripts for scan insertion and ATPG.	R	N/A	10		N/A

Comment: Design Target is FPGA. So these are not needed here.
Hence the Assessment for this guideline is Not Applicable

R9.1.1.1.6

Table 105: R 9.1.1.1.6

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.1.6	Reference library.	R	N	10	0	N/A

Comment: Reference Library not given
Hence the Assessment for this guideline is Never

R9.1.1.1.7

Table 106: R 9.1.1.1.7

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.1.7	Installation scripts.	R	N/A	10		N/A

Comment: No complex installation needed. All files present in one compressed file.
Hence the Assessment for this guideline is Not Applicable

R9.1.1.2: Verification Files

R9.1.1.2.1

Table 107: R 9.1.1.2.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.2.1	Bus functional model/monitors used in testbench.	R	A	10	10	N/A

Comment: Monitors are used in the testbench, and also documented.
Hence the Assessment for this guideline is Always

R9.1.1.2.2

Table 108: R 9.1.1.2.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.2.2	Testbench files including representative verification tests.	R	A	10	10	N/A

Comment: Testbench files present.
Hence the Assessment for this guideline is Always

R9.1.1.3: Documentation

R9.1.1.3.1

Table 109: R 9.1.1.3.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.3.1	User guide / Functional specification.	R	A	10	10	N/A

Comment: User Guide '*external_memory_programmers_guide.pdf*' and '*Rapier_External_Memory_Controller.pdf*' Present
Hence the Assessment for this guideline is Always

R9.1.1.3.2

Table 110: R 9.1.1.3.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.3.2	Datasheet.	R	N	10	0	N/A

Comment: No Data Sheet Given.
Hence the Assessment for this guideline is Never

R9.1.1.4: System Integration Files

G9.1.1.4.1

Table 111: G 9.1.1.4.1

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.4.1	Cycle-based/emulation models as appropriate for macro and/or its testbenches and BFMs.	G	N	2	0	N/A

Comment: No Cycle based/Emulation models provided.
Hence the Assessment for this guideline is Never

G9.1.1.3.2

Table 112: G 9.1.1.3.2

RMM Sec	Guideline	Type	Assessment	Max Score	Score	Script used
9.1.1.3.2	If macro has significant software requirements, such as microcontrollers and microprocessors, then a list of system integration tools (compilers, debuggers, real-time operating systems and software drivers) that support the macro is provided.	G	N/A	2		N/A

Comment: The design do not contain any software or software based function. Not Applicable
Hence the Assessment for this guideline is Not Applicable

Section 2.4 : Results on Soft IP Evaluation and discussion:

Overall score of the soft IP evaluation was found to be 342/584, which is assessed as 67%, given the weighted scores of different sections are different.
This score includes
#1. **A score of 201/240 i.e 83% for RTL coding guidelines**, which was the main criteria for the IP assessment in this exercise. A detailed explanation and comment has already been given.
#2. A score of 0/56 for system level Issues : Rules and Tools: Most of the guidelines does not apply for in this exercise. So this doesn't mean that the IP has failed badly with respect to this topic, but its just not enough info is given to be able to evaluate the IP again this topic
#3. A score of 0/108 for macro synthesis Guidelines: Again not much info has been given with the IP to be able to evaluate the IP against this topic. The document said that the main target of the IP was made to be FPGA instead of ASIC. So no dc_shell/ac_shell scripts were provided. Which makes the evaluation of the IP against this topic as Not Applicable.
#4. **A score of 61/68 i.e 90% for Verification guidelines**: This is a very good score. This tells that the verificaiton of the IP has been done largely as per guidelines.
#5. **A score of 80/112 ie 62% on Deliverable Guidelines**: Which is a reasonably good score.

Conclusion: Leaving the points #2 and #3 because they are mostly Not Applicable, the IP conforms to the Openmore Guidelines to a high degree.

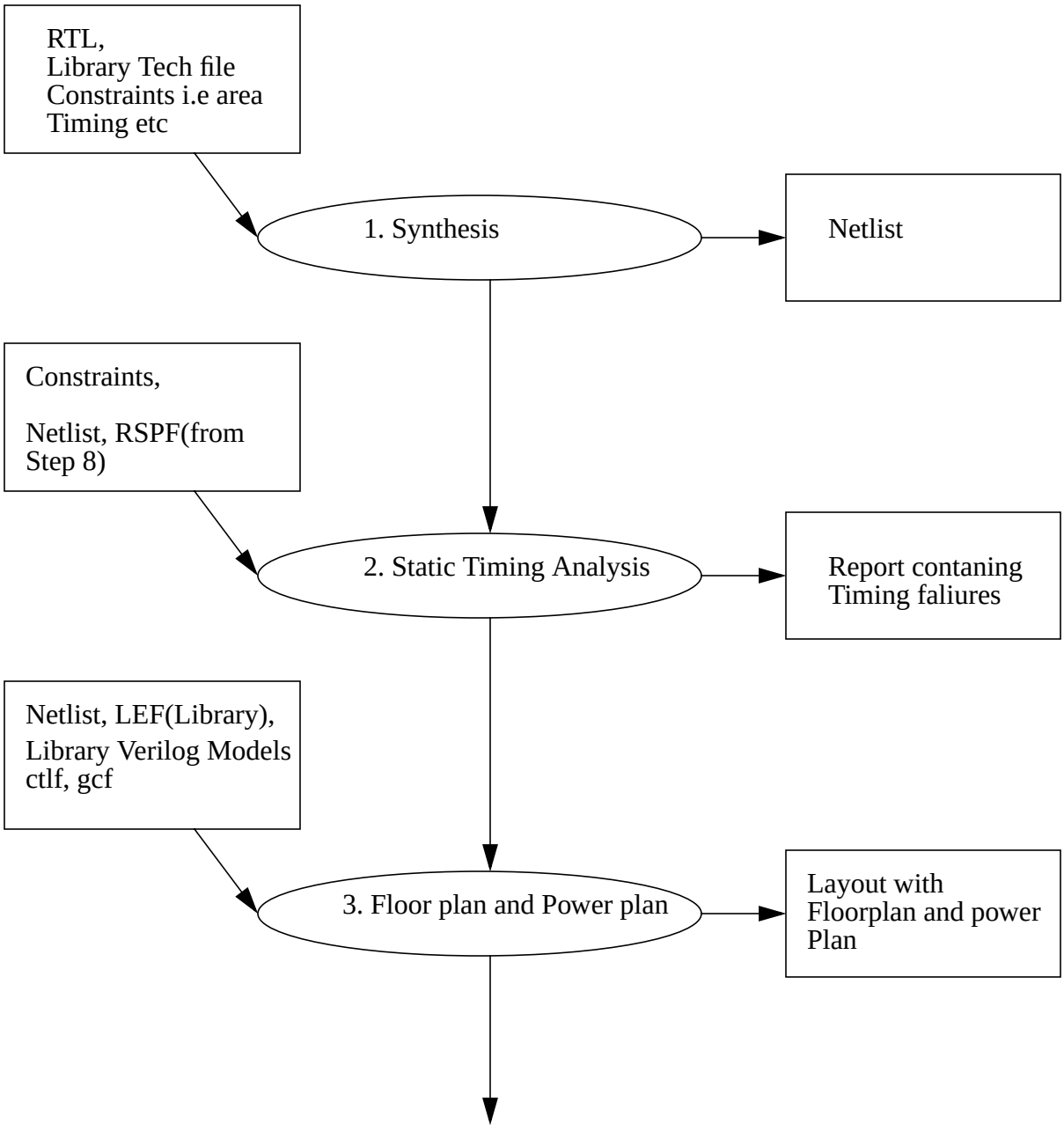
Section 3 : IP Hardening Process: RTL to GDSII

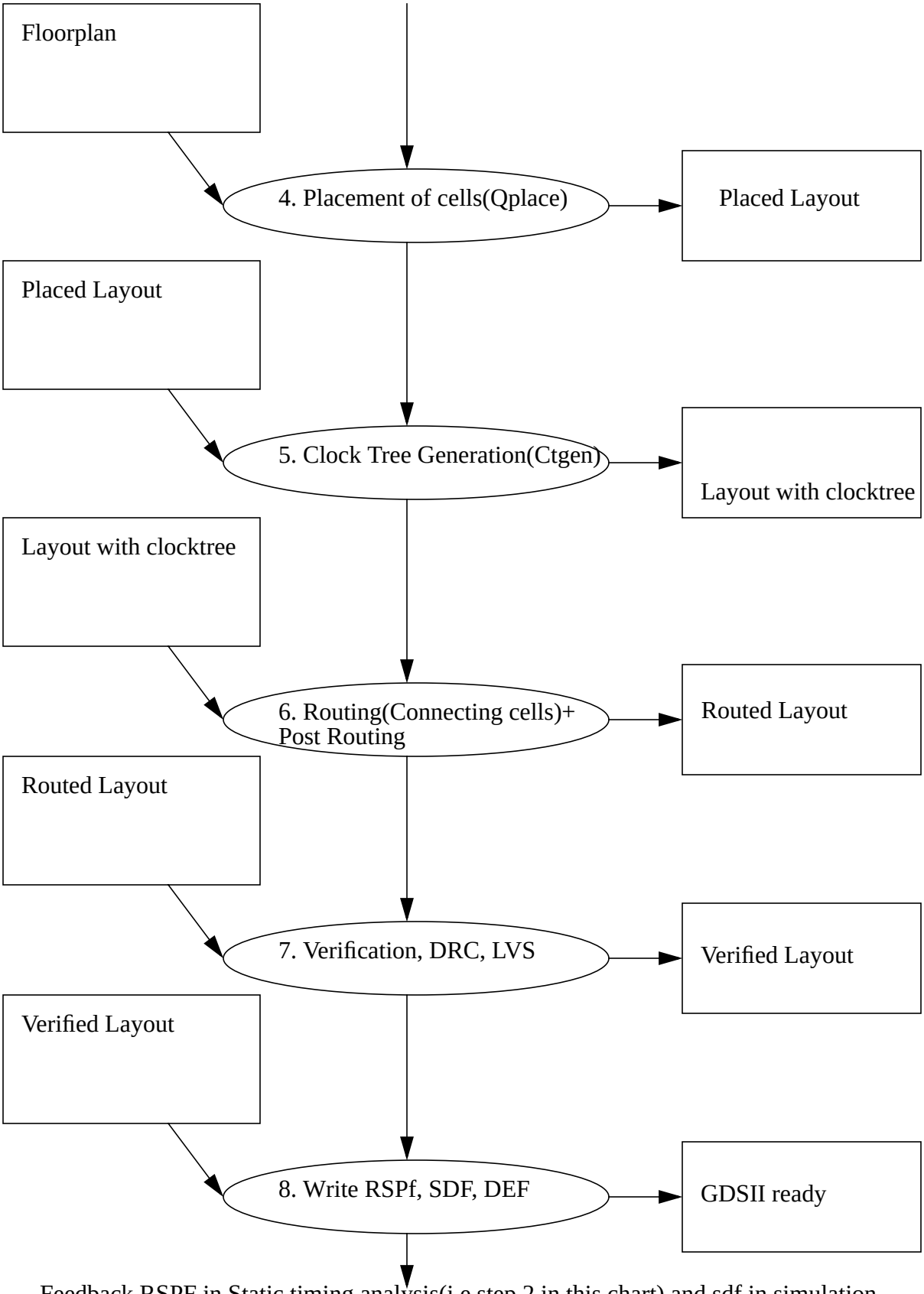
3.1 Introduction to IP Hardening Process and Hard IP Evaluation

This section gives details of how the IP was hardened and evaluated. Since the IP given is a soft-IP and it is to be hardened locally, many of the guidelines in Hard IP evaluations does not apply. The Hardening Process:

3.2 Steps

A soft IP given to us as RTL can be converted into final layout using the following flow chart. The Left hand box shows the inputs of the step, the ellipse in middle shows the step and the box on right hand side shows the result of that step.




Floorplan
4. Placement of cells(Qplace)
Placed Layout
Placed Layout
5. Clock Tree Generation(Ctgen)
Layout with clocktree
Layout with clocktree
6. Routing(Connecting cells)+
Post Routing
Routed Layout
Routed Layout
7. Verification, DRC, LVS
Verified Layout
Verified Layout
8. Write RSPf, SDF, DEF
GDSII ready
Feedback RSPF in Static timing analysis(i.e step 2 in this chart) and sdf in simulation

It is to be noted that this is a simplified flowchart giving only main steps in the layout generation. The IP was hardened using the steps shown above. The tool used for steps 3 to steps 8 is Cadence's Silicon Ensemble.

Step wise discussion of the flow: Starting from step 3 ending at step 8.

Step 3. Floor Plan and Power Plan:

This steps creates the floor plan of the design. This step needs the following as inputs
1). The verilog netlist files: The netlist which is to be hardened
2). The verilog library files: The netlist contains just the name of the cells and their connections, however more information about each cell is required, which is present in library verilog file, containing detailed model of each cell in the library, to which the design has been mapped.
3). The library LEF file: LEF stands for Library Exchange format. It is a text representation of the Abstract views, which are derived from the layouts of cells. It contains the information about the position of pins, the pin definition for each cell in design, pin capacitance, blockages. Blockages in a cell is a restricted area to which router is not allowed to route.

The purpose of this step is to create a black box of the design, which will define the Aspect Ratio, the area of the chip, the number of 'rows' in which the cells will be placed. Row utilization is also defined in this step.
It is a very difficult step to decide the area and row utilization. Because there are chances that later in steps, the area can be found to be inadequate, or it can be a lot more than needed. Usually an iterative process is followed to decide the proper area and row utilization. Experience from previous design also helps in this step a lot.

The core area is defined in this step.
The distance of the 'rows' in which cell will be placed from the IPs is defined in this step.
Results of the step:
Putting I/O to core distance of 30u and Row Utilization to 80% , and Aspect ratio to 1.0 following were the results.
Number of cells = 1308
Number of blocks = 0
Number of I/O pads = 0
Number of I/O Pins = 266
Number of Cornet Pads = 0
Number of Nets = 1619
Width of chip = 507.267u
Height of chip = 507.267u
Number of standard cell rows = 37
Area of Cells(sq microns) 160038.000
Chip Area = 257319.809 sq microns

Row Utilization was then changed to 90%
Following were the results of changing row utilization to 90%
Number of cells = 1308(same)
Number of blocks = 0(same)
Number of I/O pads = 0(same)

Number of I/O Pins = 266(same)
Number of Cornet Pads = 0(same)
Number of Nets = 1619(same)
Width of chip = 481.687u(different)
Height of chip = 481.687u(different)
Number of standard cell rows = 35(different)
Area of Cells(sq microns) 160038.000(same)
Chip Area = 232022.366 sq microns

Comment on affect of changing row utilization.
Increasing row utilization will place more cells in one row, so the number of rows will reduce. This will also reduce the chip area. But it will make the routing more difficult because there will be less space between the cells for routing metal to pass. So if the row utilization is increased very much, the chip will more likely produce problems in physical verification i.e DRC and LVS, but on the other hand keeping it very low will be a waste of area. So a trade-off must be made and using some iterative processes, row utilization should be set to a reasonable value.
I/O pins in the design are also placed in this step along the periphery of the chip. Usually the placement of the I/O is defined using a separate file, which defines where exactly an I/O will be placed. But for the purpose of this exercise, the I/Os were placed randomly.

This step also involves the definition of power stripes. To supply power to each and every row, a power grid is usually made, which thicker metal. It is very important to maintain the VDD levels throughout the chip, so that the delays are consistent, and the noise margins are as expected. To distribute power uniformly across the chip, a grid of power stripes is made, which connects to the 'rows' of cells in later stages.
The tool allows us to define the width of the metal used for power ring. Which was selected to 5 microns for metal 1 and metal 2 both.
Power stripes were then made, and the width(5u), Spacing(2u) of the power stripes and number of power stripes(3 sets) were also defined. The offset, i.e the distance from the power ring was also defined to be 100u form left as well as from right.
This step thus created a gird of power distribution for the whole chip, which helps in uniform distribution of power i.e VDD and GND thought the design. For power stripes M2 i.e metal2 was chosen.

At the end of this step we get a layout which has cell rows defined and power stripes made. Please see the attached Figures 1,2,3 which shows the area, number of I/Os, cells, cornerpads, pins etc for row utilization value of 80%, 90%, and 85%. Figure 4 shows the the layout with rows which have been created. Figure 5 shows the Power Stripes and power rings as well.

Step 4. Placement of cells

The rows created in the layout is the place where the cells in the netlist will be placed. The placement of cells in DSM flow is extremely important as this affects the timing of the design to a large extent in DSM methodology. There are various algorithms to place the cells such as Manhattan algorithm. All these algorithm helps in placing the cells in the rows created in Step3 to minimise the interconnect wire length. They help in placing those cells together which are more related to each other.

In ASIC design methodology the placement was not as critical as in DSM flow. The new tools perform what is called ‘Timing Driven Placement’, for the placement of cells in rows to minimise the critical path delay. Since the timing are also being considered in this step, the tool requires the timing information of each cell in the library as well. This is usually supplied as ‘ctlf’ file i.e ‘compiled timing library format’. Unlike the ASIC design flow, in DSM design flow, the layout tools have the power to change the netlist logically, using more appropriate cells as required. For example, the layout tool may change the drive strength of the cells as required.
Once the placement is done, usually in DSM flow, a more accurate ‘wire load models’ are generated, and the information is fed back into step 2) i.e static timing analysis. STA is then performed to see if the generated placement of cells is more likely to meet the timing requirements.
The placement of cells can also be ‘flipped’ in each row, so that the power pins/lines of cells in a row can be abutted with the power lines in the adjacent row.

Cells in the netlist were placed in the rows created in step3 above.
Figure 6 clearly shows the placed cells in the layout.

Step 5. Clock Tree Generation(CTgen)

Clock tree is an important part of the design. Clock is one single net which has to go to almost every place in the layout. Also the clock should be laid out in such a manner so that the skew in the clcok is within acceptable limits. This net usually has very large capacitive load because there can be thousands of flip-flops connected to a single clock net.
So there is a need for proper buffers in clock path, which will make sure that the clock net doesn’t loose strength, and also it will put equal timing delay from the clock pin on the boundary of chip to each flip-flop in the design, so that ‘skew’ is controlled.
This step then modifies the existing the clock tree, and optimize it according to the placement done in Step 4 above. In the practical exercise, this step failed given an unknown error.

Step 6. Routing(Connecting cells)+ Post Routing
The cells placed in steps above now needs to be connected together by physical wires. The connections are defined in the netlist file. To avoid shorts, several metal layers can be used to perform routing. Also the layers have defined directions. For example a given layer will have a defined direction i.e weather it runs horizontally or vertically.
If M1 is Left to Right or vice-versa, M2 will be top to bottom or vice versa, M3 will be Left to Right or vice-versa, M4 will be top to bottom or vice versa and so on. This is a valid scheme of metal routing direction, and very much used in practice.
All these metals ie M1 M2 M3 M4 are assigned different horizontal planes. ‘Vias’ are used to make connections between two metals on different horizontal planes.
Not only the cells are connected together in this step, the power rails are also connected to supply power to the full chip.
The first step was to connect the power rings, rails and power signals to pad/blocks. For this ‘connect ring’ command was used. Global routing was then done to make the connections between the cells.
In this exercise 5 Metal layers were used to make connections i.e M1, M2, M3, M4, M5
Fig 7 shows a clear picture of connected cells. Fig 8 shows a zoomed area from Fig 7 which shows the metal routing in a more elaborate way

Step 7. Physical Verification DRC and LVS:
This step verifies the layout created in step 6 above. Design Rule Check is run, to see if all the design rules are followed. A layout versus schematic check is also performed to see that the layout generated corresponds to the input netlist. This step however was not performed as it was not a part of the exercise

Step 8: Write RSPF, SDF, DEF:
Parasitic extraction is performed, and put in a industry standard format i.e RSPF(Reduced Standard Parasitic Format), which is then fed back into step 2 i.e Static Timing Analysis. It is very important that STA passes with this parasitic information. SDF is also written which is a delay file(Synopsys Delay Format or Standard Delay Format) which is used by post netlist gate level simulations. The final design is then written out as a DEF(Design Exchange Format).

Answers to asked questions:
1). How many cells, blocks, IO pads, IO pins, Corner Pads and nets are contained in design
Answer :
No of I/O cells = 1308
No of blocks = 0
No of I/O pads = 0
No of I/O pins = 266
No of Corner Pads = 0
No of nets = 1619
2). What is the area of the design
Area of chip = 234170.824 square microns, area of cells = 160038.000 square microns
3). How many layers are there in the design
Answer : 5
4). How many stripes were added to the design
Answer : 3
5). How many Stripe connections to core rings are there in the design
Answer 6. £ each for VDD and GND
6). Define block ring width
Answer: The width of the power ring is being referred to: It is defined to be 5 microns for both metal1 and metal2
7). If you need to add new stripes to a design, do you need to completely reset the floorplan
Answer No: New stripes will be added, and 'Delete All Existing Stripes' button can be switched ON while doing that. So there is no need to reset the floorplan
8).If so why
Answer: No the floor plan doesn't needs a reset. See explanation in answer to question 7
9) How is the aspect ratio calculated.
Answer: Aspect ratio is the ratio of Height/Width of the design. It is not calculated, it is given as a input field.
10). What is the desired value of aspect ratio
Answer : 1.
11). What is the die size of the design
Answer : Die size = chip area = 234170.824

12). Name the power wires in the design:
Answer : vdd! and gnd!
13). Name the metal layers used for routing the power and ground wires
Answer: metal1 and metal2
14). Explain the role of power rings, power stripes and power rails on the chip.
Answer: This is done to distribute power uniformly throughout the chip. The value of VDD should remain constant throughout the chip as noise margins, and delays depend upon this value. So there must not be any drop in VDD in the chip. To maintain VDD a power grid is made using rings, stripes, and rails.
15) Which one of the wires in 14 can easily left out without causing problems on the routed chip.
Answer: Not sure, guess, stripes can be left out, so that the horizontal rails supply power to each cell in the design
16). What are regular wires
Answer: The wires used to make connections from cell to cell
17). Distinguish between regular wires and special wires
Answer: Special wires are VDD and GND ,which has special arrangements. Regular wires are ordinary wires running form cell to cell.
18). How many metal layers...
Answer: 5 layers
19) What are vias:
Answer: Vias are the connections in vertical plane on a chip, used to make connections between 2 metal layers in different horizontal planes. For example M1 and M2 can only be connected using vias.
20). Why are they needed for routing on a chip.
Answer: On a chip several routing metal layers are employed in different horizontal planes. Vias are used to make connections between different layers in different horizontal planes.
21). What information can you obtain from the report RC file:
Answer: An RC files gives more accurate values of parasitic capacitance and resistance after a layout has been completed. Since the layout has been done, accurate lengths of metals are now defined. So the related parasitic resistance and capacitance can now be calculated. This info is stored in the RC file.
22).What is the difference between the stripes added in tutorial and in this exercise
Answer: Only difference is in the width of rings. which is 5 microns in this exercise and 10microns in the tutorials. Width and spacing of stripes is 5 and 2 microns repsectively in both tutorial and in this exercise.

Section 3.3: Results: HARD IP Generation.

The soft ip was converted into hard ip successfully. However it is to be noted that, CTgen failed and clock tree optimization therefore was not done. Also DRC/LVS was not done as it was not a part of this exercise, but it is a very important step in physical design.

The HARD IP assessment was also done using OpenMORE. The result was that the IP scored 103/504. That is the results of assessment was not found to be satisfactory. As this is just an exercise and not a real design, most of the steps in OpenMORE had not been taken into account. For example, first of all no hard ip specs were present, so no area or power goals. LVS/DRC is not a part of this exercise, but OpneMORE has significant score related to them.
But the important part is that the exercise gave an insight about how RTL to GDSII flow is done in industry, and what are the main steps/tools/requirements/results in the flow

Section 4: Common or total Results and results discussion

The total result of the IP changed significantly after evaluation of HARD IP. Soft IP scored very good result of around 80%(when evaluated against the main criteria i.e RTL coding guidelines), where as the result of HARD IP was not found up to a level of satisfaction. It produced a score of 29% only. The reason is simple, the IP given is given as a soft ip, and all the related documentation, specs, code, scripts refer to the soft ip only. Hard IP was generated locally and there was no information related to generation of hard ip. Also the soft ip was mainly targeted on FPGA so it made more difficult to hardened in. So the poor result in evaluation of HARD IP is quite obvious.

Section 5 :Comment on the OpenMore environment.

OpenMORE environment was found very useful in evaluating the IP. Following OpenMORE can give a very high degree of re-use, understad ability of code, quality of code, good synthesis practices for the code. It can quickly give an overall assessment on the quality of RTL code. The guidelines in OpenMORE also avoids common mistakes which a designer can do: such as simulation and synthesis mismatches, un-intentional inference of latches, unstability due to asynchronous feedbacks in the design.
It not only provides good practices for RTL, but also for Verification, System Design issues, and Physical Design.

Main Advantages of OpenMORE:
1). Gives high degree of re-use
2). Helps in avoiding common mistakes in RTL
3). Gives a way to code which is quickly understandable.
4). Gives marketability to the IP
6). Gives a very good industry standard common checklist, so that its difficult to forget things.
7). Gives a good way to implement verification plan

However there are some disadvantages too
1). OpneMORE is very exhaustive.
2). It can be very time consuming to follow each and every guideline
3). Following each and every guideline is not always possible.
4). Some guidelines may require unnecessary change in the design, giving not as much advantage.

Addition to OpenMORE.
In Section 5.3.7 ie. Coding For Translation (VHDL to Verilog), following guideline is recommended to be added:
IF VHDL, no use of attributes like ‘HIGH, ‘LOW, ‘LENGTH.

These attributes are used commonly by VHDL designers to write functions in vhdl, which are synthesizeable. And there is no replacement of these attributes in verilog.

Section 6 APPENDIX:

B: Figures from Hard IP generation Process:

C : Scripts used for some guidelines

blocking.pl

```
#!/usr/bin/perl
################################################################################
###
# To extract WARNINGS/VIOLATIONS in RTL for R5.5.6.1 of Openmore
# Always use non blocking assignemnts in always blocks
# Assumptions:
# 1). There are no 'assign' statements inside an always statements
################################################################################
###

if(@ARGV[0] eq "") {
  print "ERROR: insufficient fields\n";
  print "Usage:unix> stuff.pl <input_file> \n";
  exit;
}

$begin = 0;
$always = 0;
$edge = 0;
$mystr = "";
$start_cat = 0;

open(out,">@ARGV[0].R5_5_6_1") || die "Couldn't open mapfile for @ARGV[0]\n" ;
open(infile0,"<@ARGV[0]") || die "Couldn't open infile0. @ARGV[0]\n ";
#system("awk '/module/,/endmodule/' @ARGV[0]>@ARGV[0].module")
  while ($line1 = <infile0>) {
    if(($line1 =~ ///\*/) && ($ignore == 0)) {
      $ignore = 1;
    }
    if(($ignore == 1) && ($line1 =~ /\*///)) {
      $ignore = 0;
    }
    $nn++;
    if($ignore == 1) {
      next;
    }
    chomp($line1);
    @lines = split(///, $line1);
    $no_of_ele = $#lines;
    $oneline = $lines[0];
    $oneline =~ s/(\s+)//g;
    if(($oneline =~ /edge/)&&($edge == 0)&&($oneline =~ /always/)) {
      $start_cat = 1;
      $edge = 1;
```

```
  }
  elsif(($oneline =~ /(assign|always|endmodule)/) && ($edge == 1)) {
    $start_cat = 0;
    $edge = 0;
  }
  if($start_cat == 1) {
     $oneline =~ s/<=/##/g;
     $oneline =~ s/==/##/g;
     if($oneline =~ /for‹/) { #filter out statements with for loops
       next;
     }
     if($oneline =~ /=/) {
       print out "VIOL:R5.5.6.1: blocking assignment found in always @ (anyedge) block\n";
       print out "$nn: $line1\n\n";
     }
  }
  if(($oneline =~ /edge/)&&($edge == 0)&&($oneline =~ /always/)) {
    $start_cat = 1;
    $edge = 1;
  }
 } #while ($line1 = <infile0>)
```

#The logic: if any always begins with ’edge’ ie. posedge or negedge in it
#try to find a ’=’ till you get to enter another always or till you see
#a assign or till you see a endmodule.
#Since it was not possible to find out when an always block ends, the above
#stpes were taken thinking, that
#1). If another alwyas is fouund, the prev one must have ended
#2). If an assign is found, the prev always must have ended
#3). If an endmodule is found, the prev always must have ended

file: uniq.pl

```
#!/usr/bin/perl
################################################################################
###
# To extract WARNINGS/VIOLATIONS in RTL for R5.2.15.5 of Openmore
# To find duplicate signals/veriables in two RTL files
################################################################################
###

if(@ARGV[0] eq "") {
  print "ERROR: insufficient fields\n";
  print "Usage:unix> stuff.pl <input_file> \n";
  exit;
}

$cat = 0;
$ii=0;
open(out,">file.R5_2_7_1") || die "Couldn't open file.R5_2_7_1\n" ;
foreach $file (@ARGV) {
  print "$file\n";
  open(infile0,"<$file") || die "Couldn't open infile0. $file\n ";
  while ($line1 = <infile0>) {
    if(($line1 =~ //\*/)) {
      $ignore = 1;
    }
    if(($ignore == 1) && ($line1 =~ /\*//)) {
      $ignore = 0;
    }
    $nn++;
    if($ignore == 1) {
       next;
    }
    chomp($line1);
    @lines = split(///, $line1);
    $oneline = " $lines[0]";
    if(($oneline =~ /\s+module\s+|\s+reg\s+|\s+wire\s+/ )) {
      $cat = 1;
    }
    if(($oneline =~ /(;|‹)/)&&($cat==1)) {
      $cat = 0;
      $myline = $myline.$oneline;
      #print "$myline\n";
      if($myline =~ /(module)\s+(\w+)/) {
        $fline = $2;
      }
      else {
      $fline = $myline;
```

```
      }
      $myline = "";
      $fline =~ s/;//g;
      $fline =~ s/,/ /g;
      $fline =~ s/\[.*\]//g;
      $fline =~ s/(\s+module\s+|\s+wire\s+|\s+reg\s+)//g;
      @plist = split(/\s+/, $fline);
      #print "f=$fline\n";
      if($ii==0) {
        push(@parray_vec0, @plist);
        #@parray_vec0 = @parray;
      }
      elsif($ii==1) {
        push(@parray_vec1, @plist);
        #@parray_vec1 = @parray;
      }
      @plist = "";
    }
    if($cat ==1) {
      $myline = $myline.$oneline;
    }

  } #while ($line1 = <infile0>)
  close(infile0);
  #print "closing file\n";

  $ii++;
}

  foreach $signal0 (@parray_vec0) {
    foreach $signal1 (@parray_vec1) {
      if($signal0 eq $signal1) {
        print out "VIOL:R5.2.15.5 Duplicate signal/module/variable '$signal0' in $ARGV[0] and
$ARGV[1]\n";
        $jj++;
      }
    }
  }
  print out "Total Number of Duplicate signals/variables/modules = $jj\n";
```

file: ucase.pl

```
#!/usr/bin/perl
################################################################################
###
# To extract WARNINGS/VIOLATIONS in RTL for G5.2.1.3 of Openmore
# To find a lower case letter in constant definition
# Only ’define statements are checked here
################################################################################
###
#WARNING:WARNING: This file is not ready to be used

if(@ARGV[0] eq "") {
  print "ERROR: insufficient fields\n";
  print "Usage:unix> stuff.pl <input_file> \n";
  exit;
}

open(out,">@ARGV[0].G_5_2_1_3") || die "Couldn’t open mapfile for @ARGV[0]\n" ;
open(infile0,"<@ARGV[0]") || die "Couldn’t open infile0. @ARGV[0]\n ";
#system("awk ’/module/,/endmodule/’ @ARGV[0]>@ARGV[0].module")
   while ($line1 = <infile0>) {
        if($line1 =~ /module/) {
          $ignore = 1;
        }
        if(($line1 =~ /;/) && ($ignore == 1)) {
          $ignore = 0;
        }
        $nn++;
        if($ignore == 1) {
          next;
        }
        chomp($line1);
        @lines = split(///, $line1);
        $no_of_ele = $#lines;
        $lines[0] = " "."$lines[0]";
       if($lines[0] =~ /(‘define\s+(.*)\s+)/) {
          if($2 =~ /[a-z]/) {
             print out "VIOL:G.5.2.1.3 Lowercase letter found in Constant Name \n";
             print out "$nn: $lines[0]\n";

          }
       }
   } #while ($line1 = <infile0>)
```

file: sepline.pl

```
#!/usr/bin/perl
################################################################################
###
# To extract WARNINGS/VIOLATIONS in RTL for R5.2.6.1 of Openmore
# To extract multiple statements in one line
# Checked for 2 semicolons in one line
################################################################################
###

if(@ARGV[0] eq "") {
  print "ERROR: insufficient fields\n";
  print "Usage:unix> stuff.pl <input_file> \n";
  exit;
}

open(out,">@ARGV[0].R5_2_6_1") || die "Couldn't open mapfile for @ARGV[0]\n" ;
open(infile0,"<@ARGV[0]") || die "Couldn't open infile0. @ARGV[0]\n ";
#system("awk '/module/,/endmodule/' @ARGV[0]>@ARGV[0].module")
    while ($line1 = <infile0>) {
         if(($line1 =~ /\*/)) {
           $ignore = 1;
         }
         if(($ignore == 1) && ($line1 =~ /\*/)) {
           $ignore = 0;
         }
         $nn++;
        if($ignore == 1) {
         next;
        }
         chomp($line1);
         @lines = split(///, $line1);
         $no_of_ele = $#lines;
        $oneline = $lines[0];
        $oneline =~ s/(\s+)//g;

          if($oneline =~ /;(.*);/ ) {
            print out "VIOL:R5.2.6.1: Two or more statements in one line found\n";
            print out "$nn: $line1\n\n";
          }
    } #while ($line1 = <infile0>)
```

file lcase.pl

```
#!/usr/bin/perl
###############################################################################
###
# To extract WARNINGS/VIOLATIONS in RTL for G5.2.1.2 of Openmore
# To find Uppercase letters in signals,variables,port Names
# Items checked : input,output,inout,reg,wire,integer
###############################################################################
###
#WARNING:WARNING: This file is not ready to be used

if(@ARGV[0] eq "") {
  print "ERROR: insufficient fields\n";
  print "Usage:unix> stuff.pl <input_file> \n";
  exit;
}

open(out,">@ARGV[0].G_5_2_1_2") || die "Couldn't open mapfile for @ARGV[0]\n" ;
open(infile0,"<@ARGV[0]") || die "Couldn't open infile0. @ARGV[0]\n ";
#system("awk '/module/,/endmodule/' @ARGV[0]>@ARGV[0].module")
   while ($line1 = <infile0>) {
        if($line1 =~ /module/) {
          $ignore = 1;
        }
        if(($line1 =~ /;/) && ($ignore == 1)) {
          $ignore = 0;
        }
        $nn++;
        if($ignore == 1) {
          next;
        }
        chomp($line1);
        @lines = split(/// , $line1);
        $no_of_ele = $#lines;
        $lines[0] = " "."$lines[0]";
       if($lines[0] =~ /(\s+input\s+|\s+output\s+|\s+inout\s+|\s+reg\s+|\s+wire\s+|\s+integer\s+)/) {
          if($lines[0] =~ /[A-Z]/) {
             print out "VIOL:G.5.2.1.2 Uppercase letter found in signal/variable/port Name \n";
             print out "$nn: $lines[0]\n";

          }
        }
   } #while ($line1 = <infile0>)
```

file: hardcode.pl

```
#!/usr/bin/perl
###############################################################################
###
# To extract WARNINGS/VIOLATIONS in RTL for G11.2.10.2 of Openmore
###############################################################################
###

if(@ARGV[0] eq "") {
  print "ERROR: insufficient fields\n";
  print "Usage:unix> stuff.pl <input_file> \n";
  exit;
}

open(out,">@ARGV[0].G5_3_2_1") || die "Couldn't open mapfile for @ARGV[0]\n" ;
open(infile0,"<@ARGV[0]") || die "Couldn't open infile0. @ARGV[0]\n ";
#system("awk '/module/,/endmodule/' @ARGV[0]>@ARGV[0].module")
    while ($line1 = <infile0>) {
         if(($line1 =~ //\*/)) {
           $ignore = 1;
         }
         if(($ignore == 1) && ($line1 =~ /\*//)) {
           $ignore = 0;
         }
         $nn++;
        if($ignore == 1) {
         next;
        }
         chomp($line1);
         @lines = split(///, $line1);
         $no_of_ele = $#lines;
        $oneline = $lines[0];
        $oneline =~ s/(\s+)//g;
        if($oneline =~ /(parameter|define|')/) { #to aviod stuff like if('DATA_SIZE = 100)
                                        #because this is realy not hardcoding
          next;
        }
        #if($oneline =~ /(parameter|define)/) {
        #  next;
        #}
        if($oneline =~ /(\[(†+):(†+)])/ ) {
          print out "VIOL:G5.3.2.1: hardcoding found type1\n";
          print out "$nn: $line1\n\n";
          next;
        }
        if($oneline =~ /((†+)'(b|h|d))/ ) {
          if(!($2==1)) {
```

```
          print out "VIOL:G5.3.2.1: hardcoding found type2\n";
          print out "$nn: $line1\n\n";
          next;
        }
      }
      if($oneline =~ /('(b|h|d)(†+))/ ) {
        if(!(($3==0)||($3==1))) {
          print out "VIOL:G5.3.2.1: hardcoding found type3\n";
          print out "$nn: $line1\n\n";
          next;
        }
      }
      if($oneline =~ /('(h)([A-Fa-f]))/ ) {
        print out "VIOL:G5.3.2.1: hardcoding found type4\n";
        print out "$nn: $line1\n\n";
        next;
      }
      if(($oneline =~ /(=(†+))/ )&&(!($oneline =~ /(for|while)/))) {
        if(!(($2==0)||($2==1))) {
          print out "VIOL:G5.3.2.1: hardcoding found type5\n";
          print out "$nn: $line1\n\n";
          next;
        }
      }
      if($oneline =~ /((†+){)/ ) {
          print out "VIOL:G5.3.2.1: hardcoding found type6\n";
          print out "$nn: $line1\n\n";
          next;
      }
  } #while ($line1 = <infile0>)
```

file: downto.pl

```
#!/usr/bin/perl
###############################################################################
# To extract WARNINGS/VIOLATIONS in RTL for R5.2.1.11 of Openmore
# To extract [0:x] type structurs in RTL
# ONLY ports are checked. regs, wires etc are NOT checked
# Bceause RMM spread sheet refers to ports ONLY
###############################################################################

if(@ARGV[0] eq "") {
  print "ERROR: insufficient fields\n";
  print "Usage:unix> stuff.pl <input_file> \n";
  exit;
}

open(out,">@ARGV[0].R5_2_1_11") || die "Couldn't open mapfile for @ARGV[0]\n" ;
open(infile0,"<@ARGV[0]") || die "Couldn't open infile0. @ARGV[0]\n ";
#system("awk '/module/,/endmodule/' @ARGV[0]>@ARGV[0].module")
    while ($line1 = <infile0>) {
         if(($line1 =~ /\*/)) {
           $ignore = 1;
         }
         if(($ignore == 1) && ($line1 =~ /\*/)) {
           $ignore = 0;
         }
         $nn++;
        if($ignore == 1) {
         next;
        }
         chomp($line1);
         @lines = split(///, $line1);
         $no_of_ele = $#lines;
        $oneline = $lines[0];
        $oneline =~ s/(\s+)//g;

        if($oneline =~ /input|output|inout/ ) {
          if($oneline =~ /(\[(0):(†+)])/ ) {
            print out "VIOL:R5.2.1.11: [0:x] found type1\n";
            print out "$nn: $line1\n\n";
          }
          if($oneline =~ /(\[(0):(\w+)])/ ) {
            print out "VIOL:R5.2.1.11: [0:x] found type2\n";
            print out "$nn: $line1\n\n";
          }
        }
    } #while ($line1 = <infile0>)
file: clkname.pl
```

```
#!/usr/bin/perl
###############################################################################
###
# To extract WARNINGS/VIOLATIONS in RTL for G5.2.1.6 of Openmore
# To find that a clock singal name is 'clk' or prefixed with 'clk'
# all the signals which are which follow 'posedge' are considered to be clock
# signas
# Assumption: there are no Async Resets in the design
###############################################################################
###
#WARNING:WARNING: This file is not ready to be used

if(@ARGV[0] eq "") {
  print "ERROR: insufficient fields\n";
  print "Usage:unix> stuff.pl <input_file> \n";
  exit;
}

$clkname = "####";
open(out,">@ARGV[0].G_5_2_1_6") || die "Couldn't open mapfile for @ARGV[0]\n" ;
open(infile0,"<@ARGV[0]") || die "Couldn't open infile0. @ARGV[0]\n ";
#system("awk '/module/,/endmodule/' @ARGV[0]>@ARGV[0].module")
    while ($line1 = <infile0>) {
         if($line1 =~ /module/) {
           $ignore = 1;
         }
         if(($line1 =~ /;/) && ($ignore == 1)) {
           $ignore = 0;
         }
         $nn++;
         if($ignore == 1) {
           next;
         }
         chomp($line1);
         @lines = split(///, $line1);
         $no_of_ele = $#lines;
         $origline = $lines[0];
         $lines[0] = " "."$lines[0]";
         $lines[0] =~ s/›/ /g ;
        if($lines[0] =~ /(posedge\s+(\w+)\s+)/) {
            $clk = $2;
          if(!($clk =~ /^(clk)/)) {
             if($clk eq $clkname) {
             }
             else {
             $clkname = $clk;
```

```
            print out "VIOL:G.5.2.1.6 clk like signal not prefixed with ’clk’\n";
            print out "$nn: $origline\n";
            }

          }
        }
    } #while ($line1 = <infile0>)
```

filename: chars132.pl

```
#!/usr/bin/perl
################################################################################
###
# To extract WARNINGS/VIOLATIONS in RTL for R5.2.7.1 of Openmore
# To check if the no of chars in each lines are less than  or equal to 132
# Split the line using 'no' delimeter, and then have the no of elements in
# resulting array.
################################################################################
###

if(@ARGV[0] eq "") {
  print "ERROR: insufficient fields\n";
  print "Usage:unix> stuff.pl <input_file> \n";
  exit;
}

open(out,">@ARGV[0].R5_2_7_1") || die "Couldn't open mapfile for @ARGV[0]\n" ;
open(infile0,"<@ARGV[0]") || die "Couldn't open infile0. @ARGV[0]\n ";
#system("awk '/module/,/endmodule/' @ARGV[0]>@ARGV[0].module")
    while ($line1 = <infile0>) {
         if(($line1 =~ /\*/)) {
           $ignore = 1;
         }
         if(($ignore == 1) && ($line1 =~ /\*/)) {
           $ignore = 0;
         }
         $nn++;
        if($ignore == 1) {
         next;
        }
         chomp($line1);
         @lines = split(//, $line1);
         $no_of_ele = $#lines;
         if($no_of_ele>132) {
            print out "VIOL:R5.2.7.1: $no_of_ele chars in a line found.Which are grater than 132\n";
            print out "$nn: $line1\n\n";
         }
    } #while ($line1 = <infile0>)
```

D Log file generated by scripts evaluating IP

filename: ahb_external_memory_registers.v_rtl.G5_3_2_1
VIOL:G5.3.2.1: hardcoding found type1
137: input [1:0] htrans;

VIOL:G5.3.2.1: hardcoding found type1
138: input [2:0] hsize;

VIOL:G5.3.2.1: hardcoding found type1
140: input [31:0] haddr; // AHB address bus bits.

VIOL:G5.3.2.1: hardcoding found type1
141: input [31:0] hwdata;

VIOL:G5.3.2.1: hardcoding found type1
150: output [1:0] hresp; // AHB response. This module only provides.

VIOL:G5.3.2.1: hardcoding found type1
151: reg [1:0] hresp; // two types of response:

VIOL:G5.3.2.1: hardcoding found type1
157: output [31:0] hrdata;

VIOL:G5.3.2.1: hardcoding found type1
158: reg [31:0] hrdata;

VIOL:G5.3.2.1: hardcoding found type1
168: output [3:0] enable; // Active high level outputs. These are used

VIOL:G5.3.2.1: hardcoding found type1
171: output [3:0] read_only; // Active high level outputs. These are used

VIOL:G5.3.2.1: hardcoding found type1
177: output [3:0] read_wait_state0;

VIOL:G5.3.2.1: hardcoding found type1
178: output [3:0] read_wait_state1;

VIOL:G5.3.2.1: hardcoding found type1
179: output [3:0] read_wait_state2;

VIOL:G5.3.2.1: hardcoding found type1
180: output [3:0] read_wait_state3;

VIOL:G5.3.2.1: hardcoding found type1
184: output [3:0] write_wait_state0;

VIOL:G5.3.2.1: hardcoding found type1
185: output [3:0] write_wait_state1;

VIOL:G5.3.2.1: hardcoding found type1
186: output [3:0] write_wait_state2;

VIOL:G5.3.2.1: hardcoding found type1
187: output [3:0] write_wait_state3;

VIOL:G5.3.2.1: hardcoding found type1
193: reg [2:0] slave_state;

VIOL:G5.3.2.1: hardcoding found type1
197: reg [3:0] reg_addr;

VIOL:G5.3.2.1: hardcoding found type1
209: reg [9:0] mem0_control_reg,

VIOL:G5.3.2.1: hardcoding found type1
215: reg [9:0] mem_control_reg [3:0];

VIOL:G5.3.2.1: hardcoding found type1
248: hresp[1:0] <= 2'b00;

VIOL:G5.3.2.1: hardcoding found type2
251: reg_addr <= 4'b0000;

VIOL:G5.3.2.1: hardcoding found type1
277: hresp[1:0] <= 2'b01;

VIOL:G5.3.2.1: hardcoding found type1
283: hresp[1:0] <= 2'b00;

VIOL:G5.3.2.1: hardcoding found type1
285: reg_addr[3:0] <= haddr[3:0];

VIOL:G5.3.2.1: hardcoding found type1
291: hresp[1:0] <= 2'b00;

VIOL:G5.3.2.1: hardcoding found type1
293: reg_addr[3:0] <= haddr[3:0];

VIOL:G5.3.2.1: hardcoding found type1
308: hresp[1:0] <= 2'b01;

VIOL:G5.3.2.1: hardcoding found type1
338: hresp[1:0] <= 2'b01;

VIOL:G5.3.2.1: hardcoding found type1
344: hresp[1:0] <= 2'b00;

VIOL:G5.3.2.1: hardcoding found type1
346: reg_addr[3:0] <= haddr[3:0];

VIOL:G5.3.2.1: hardcoding found type1
352: hresp[1:0] <= 2'b00;

VIOL:G5.3.2.1: hardcoding found type1
354: reg_addr[3:0] <= haddr[3:0];

VIOL:G5.3.2.1: hardcoding found type1
359: hresp[1:0] <= 2'b00;

VIOL:G5.3.2.1: hardcoding found type1
413: hresp[1:0] <= 2'b01;

VIOL:G5.3.2.1: hardcoding found type1
420: hresp[1:0] <= 2'b00;

VIOL:G5.3.2.1: hardcoding found type1
422: reg_addr[3:0] <= haddr[3:0];

VIOL:G5.3.2.1: hardcoding found type1
428: hresp[1:0] <= 2'b00;

VIOL:G5.3.2.1: hardcoding found type1
431: reg_addr[3:0] <= haddr[3:0];

VIOL:G5.3.2.1: hardcoding found type1
449: hresp[1:0] <= 2'b00;

VIOL:G5.3.2.1: hardcoding found type1
545: hrdata[31:0] <= 32'b0;

VIOL:G5.3.2.1: hardcoding found type1
555: 4'b1000 : hrdata[31:0] <= {22'b0, mem0_control_reg[9:0]};

VIOL:G5.3.2.1: hardcoding found type1
556: 4'b0100 : hrdata[31:0] <= {22'b0, mem1_control_reg[9:0]};

VIOL:G5.3.2.1: hardcoding found type1

557: 4’b0010 : hrdata[31:0] <= {22’b0, mem2_control_reg[9:0]};

VIOL:G5.3.2.1: hardcoding found type1
558: 4’b0001 : hrdata[31:0] <= {22’b0, mem3_control_reg[9:0]};

VIOL:G5.3.2.1: hardcoding found type1
561: default : hrdata[31:0] <= 32’b0;

VIOL:G5.3.2.1: hardcoding found type1
631: mem0_control_reg[9:0] <= {4’b1111, 4’b1111, 2’b11};

VIOL:G5.3.2.1: hardcoding found type1
637: mem0_control_reg[9:0] <= hwdata[9:0];

VIOL:G5.3.2.1: hardcoding found type1
652: mem1_control_reg[9:0] <= {4’b1111, 4’b1111, 2’b00};

VIOL:G5.3.2.1: hardcoding found type1
658: mem1_control_reg[9:0] <= hwdata[9:0];

VIOL:G5.3.2.1: hardcoding found type1
672: mem2_control_reg[9:0] <= {4’b1111, 4’b1111, 2’b00};

VIOL:G5.3.2.1: hardcoding found type1
678: mem2_control_reg[9:0] <= hwdata[9:0];

VIOL:G5.3.2.1: hardcoding found type1
692: mem3_control_reg[9:0] <= {4’b1111, 4’b1111, 2’b00};

VIOL:G5.3.2.1: hardcoding found type1
698: mem3_control_reg[9:0] <= hwdata[9:0];

VIOL:G5.3.2.1: hardcoding found type1
727: assign read_wait_state0 = mem0_control_reg[5:2];

VIOL:G5.3.2.1: hardcoding found type1
728: assign read_wait_state1 = mem1_control_reg[5:2];

VIOL:G5.3.2.1: hardcoding found type1
729: assign read_wait_state2 = mem2_control_reg[5:2];

VIOL:G5.3.2.1: hardcoding found type1
730: assign read_wait_state3 = mem3_control_reg[5:2];

VIOL:G5.3.2.1: hardcoding found type1
731: assign write_wait_state0 = mem0_control_reg[9:6];

VIOL:G5.3.2.1: hardcoding found type1
732: assign write_wait_state1 = mem1_control_reg[9:6];

VIOL:G5.3.2.1: hardcoding found type1
733: assign write_wait_state2 = mem2_control_reg[9:6];

VIOL:G5.3.2.1: hardcoding found type1
734: assign write_wait_state3 = mem3_control_reg[9:6];

filename ahb_external_memory_control.v_rtl.G5_3_2_1
VIOL:G5.3.2.1: hardcoding found type1
189: input [3:0] hsel_mem; // Active high AHB memory bank select.

VIOL:G5.3.2.1: hardcoding found type1
196: input [1:0] htrans; // AHB transfer type indicator.

VIOL:G5.3.2.1: hardcoding found type1
202: input [2:0] hsize; // AHB transfer size indicator.

VIOL:G5.3.2.1: hardcoding found type1
218: input [31:0] haddr; // AHB address bus.

VIOL:G5.3.2.1: hardcoding found type1
220: input [31:0] hwdata; // APB data inout for write cycles.

VIOL:G5.3.2.1: hardcoding found type1
229: output [1:0] hresp; // AHB response. This module only provides.

VIOL:G5.3.2.1: hardcoding found type1
230: reg [1:0] hresp; // two types of response:

VIOL:G5.3.2.1: hardcoding found type1
236: output [31:0] hrdata; // APB data output for read cycles.

VIOL:G5.3.2.1: hardcoding found type1
237: reg [31:0] hrdata;

VIOL:G5.3.2.1: hardcoding found type1
248: input [3:0] enable; // Active high level inputs. These are used

VIOL:G5.3.2.1: hardcoding found type1
251: input [3:0] read_only; // Active high level inputs. These are used

VIOL:G5.3.2.1: hardcoding found type1
274: input [3:0] read_wait_state0;

VIOL:G5.3.2.1: hardcoding found type1
275: input [3:0] read_wait_state1;

VIOL:G5.3.2.1: hardcoding found type1
276: input [3:0] read_wait_state2;

VIOL:G5.3.2.1: hardcoding found type1
277: input [3:0] read_wait_state3;

VIOL:G5.3.2.1: hardcoding found type1
278: input [3:0] write_wait_state0;

VIOL:G5.3.2.1: hardcoding found type1
279: input [3:0] write_wait_state1;

VIOL:G5.3.2.1: hardcoding found type1
280: input [3:0] write_wait_state2;

VIOL:G5.3.2.1: hardcoding found type1
281: input [3:0] write_wait_state3;

VIOL:G5.3.2.1: hardcoding found type1
286: input [31:0] mem_datain_i; // Memory device read databus.

VIOL:G5.3.2.1: hardcoding found type1
288: input [3:0] mem_invertbits_i; // Memory device read invertbits.

VIOL:G5.3.2.1: hardcoding found type1
295: output [3:0] mem_chip_enable_n_o; // Active low chip enable signals for

VIOL:G5.3.2.1: hardcoding found type1
296: reg [3:0] mem_chip_enable_n_o; // external memory devices.

VIOL:G5.3.2.1: hardcoding found type1
304: output [3:0] mem_byte_enable_n_o; // Active low byte strobe signals for

VIOL:G5.3.2.1: hardcoding found type1
305: reg [3:0] mem_byte_enable_n_o; // external memory devices. These are

VIOL:G5.3.2.1: hardcoding found type1
311: output [31:0] mem_address_o; // Output address bus for external

VIOL:G5.3.2.1: hardcoding found type1

312: reg [31:0] mem_address_o; // memory devices.

VIOL:G5.3.2.1: hardcoding found type1
315: output [31:0] mem_dataout_o; // Output databus for external

VIOL:G5.3.2.1: hardcoding found type1
316: reg [31:0] mem_dataout_o; // memory devices.

VIOL:G5.3.2.1: hardcoding found type1
318: output [3:0] mem_invertbits_o; // Output databus for invert bits

VIOL:G5.3.2.1: hardcoding found type1
319: reg [3:0] mem_invertbits_o; // memory devices.

VIOL:G5.3.2.1: hardcoding found type1
321: output [3:0] mem_dataout_en_o; // Output enables

VIOL:G5.3.2.1: hardcoding found type1
322: reg [3:0] mem_dataout_en_o; // Output enables

VIOL:G5.3.2.1: hardcoding found type1
343: reg [3:0] wait_state_cntr;

VIOL:G5.3.2.1: hardcoding found type1
349: reg [1:0] hresp_pr;

VIOL:G5.3.2.1: hardcoding found type1
350: reg [3:0] byte_lane_enable;

VIOL:G5.3.2.1: hardcoding found type1
351: reg [3:0] byte_lane_enable_pr;

VIOL:G5.3.2.1: hardcoding found type1
355: reg [3:0] read_wait_state;

VIOL:G5.3.2.1: hardcoding found type1
356: reg [3:0] write_wait_state;

VIOL:G5.3.2.1: hardcoding found type1
357: reg [3:0] mem_chip_enable_n_pr;

VIOL:G5.3.2.1: hardcoding found type1
363: reg [3:0] hsel_mem_reg; // The captured values of hsel_mem

VIOL:G5.3.2.1: hardcoding found type1

365: reg [32:0] haddr_gray_capture; // Gray format of haddr for checking byte enable in protocol checker

VIOL:G5.3.2.1: hardcoding found type1
368: reg [35:0] hwdata_businvert_capture; // BusInvert coded format of hwdata

VIOL:G5.3.2.1: hardcoding found type1
375: reg [5:0] current_state, next_state;

VIOL:G5.3.2.1: hardcoding found type1
630: always @ (posedge hclk) hresp[1:0] <= #tm_prop hresp_pr[1:0];

VIOL:G5.3.2.1: hardcoding found type2
704: mem_chip_enable_n_pr <= 4'b1111; // Default value

VIOL:G5.3.2.1: hardcoding found type1
721: always @ (posedge hclk) mem_chip_enable_n_o[3:0] <= #tm_prop
mem_chip_enable_n_pr[3:0];

VIOL:G5.3.2.1: hardcoding found type1
807: hsel_mem_reg[3:0] <= #tm_prop 4'b0000;

VIOL:G5.3.2.1: hardcoding found type2
877: byte_lane_enable_pr = 4'b0000;

VIOL:G5.3.2.1: hardcoding found type5
898: 1'b0: byte_lane_enable_pr = 4'b0011;

VIOL:G5.3.2.1: hardcoding found type2
900: default: byte_lane_enable_pr = 4'b1100;

VIOL:G5.3.2.1: hardcoding found type5
906: 1'b0: byte_lane_enable_pr = 4'b1100;

VIOL:G5.3.2.1: hardcoding found type2
908: default: byte_lane_enable_pr = 4'b0011;

VIOL:G5.3.2.1: hardcoding found type1
916: case (haddr_gray_capture[1:0])

VIOL:G5.3.2.1: hardcoding found type2
917: 2'b00: byte_lane_enable_pr = 4'b0001;

VIOL:G5.3.2.1: hardcoding found type2
918: 2'b01: byte_lane_enable_pr = 4'b0010;

VIOL:G5.3.2.1: hardcoding found type2
919: 2’b11: byte_lane_enable_pr = 4’b0100;

VIOL:G5.3.2.1: hardcoding found type2
921: default: byte_lane_enable_pr = 4’b1000;

VIOL:G5.3.2.1: hardcoding found type1
926: case (haddr_gray_capture[1:0])

VIOL:G5.3.2.1: hardcoding found type2
927: 2’b00: byte_lane_enable_pr = 4’b1000;

VIOL:G5.3.2.1: hardcoding found type2
928: 2’b01: byte_lane_enable_pr = 4’b0100;

VIOL:G5.3.2.1: hardcoding found type2
929: 2’b11: byte_lane_enable_pr = 4’b0010;

VIOL:G5.3.2.1: hardcoding found type2
931: default: byte_lane_enable_pr = 4’b0001;

VIOL:G5.3.2.1: hardcoding found type2
937: byte_lane_enable_pr = 4’b1111;

VIOL:G5.3.2.1: hardcoding found type5
949: 1’b0: byte_lane_enable_pr = 4’b0011;

VIOL:G5.3.2.1: hardcoding found type2
951: default: byte_lane_enable_pr = 4’b1100;

VIOL:G5.3.2.1: hardcoding found type5
957: 1’b0: byte_lane_enable_pr = 4’b1100;

VIOL:G5.3.2.1: hardcoding found type2
959: default: byte_lane_enable_pr = 4’b0011;

VIOL:G5.3.2.1: hardcoding found type1
968: case (haddr[1:0])

VIOL:G5.3.2.1: hardcoding found type2
969: 2’b00: byte_lane_enable_pr = 4’b0001;

VIOL:G5.3.2.1: hardcoding found type2
970: 2’b01: byte_lane_enable_pr = 4’b0010;

VIOL:G5.3.2.1: hardcoding found type2

971: 2’b11: byte_lane_enable_pr = 4’b1000;

VIOL:G5.3.2.1: hardcoding found type2
973: default: byte_lane_enable_pr = 4’b0100;

VIOL:G5.3.2.1: hardcoding found type1
978: case (haddr[1:0])

VIOL:G5.3.2.1: hardcoding found type2
979: 2’b00: byte_lane_enable_pr = 4’b1000;

VIOL:G5.3.2.1: hardcoding found type2
980: 2’b01: byte_lane_enable_pr = 4’b0100;

VIOL:G5.3.2.1: hardcoding found type2
981: 2’b11: byte_lane_enable_pr = 4’b0001;

VIOL:G5.3.2.1: hardcoding found type2
983: default: byte_lane_enable_pr = 4’b0010;

VIOL:G5.3.2.1: hardcoding found type2
989: byte_lane_enable_pr = 4’b1111;

VIOL:G5.3.2.1: hardcoding found type2
1000: byte_lane_enable <= #tm_prop 4’b0;

VIOL:G5.3.2.1: hardcoding found type2
1017: mem_byte_enable_n_o <= #tm_prop 4’b1;

VIOL:G5.3.2.1: hardcoding found type1
1021: mem_byte_enable_n_o <= #tm_prop ~byte_lane_enable_pr[3:0];

VIOL:G5.3.2.1: hardcoding found type2
1038: mem_dataout_en_o <= #tm_prop 4’b0;

VIOL:G5.3.2.1: hardcoding found type1
1044: mem_dataout_en_o <= #tm_prop (byte_lane_enable[3:0] &
{4{mem_dataout_en_pr}});

VIOL:G5.3.2.1: hardcoding found type2
1226: mem_address_o <= #tm_prop 32’h0;

VIOL:G5.3.2.1: hardcoding found type1
1265: hrdata[31:0] <= #tm_prop 32’h00000000;

VIOL:G5.3.2.1: hardcoding found type1

1279: hrdata[31:24] <= #tm_prop ~mem_datain_i[31:24];

VIOL:G5.3.2.1: hardcoding found type1
1283: hrdata[31:24] <= #tm_prop mem_datain_i[31:24];

VIOL:G5.3.2.1: hardcoding found type1
1291: hrdata[23:16] <= #tm_prop ~mem_datain_i[23:16];

VIOL:G5.3.2.1: hardcoding found type1
1295: hrdata[23:16] <= #tm_prop mem_datain_i[23:16];

VIOL:G5.3.2.1: hardcoding found type1
1303: hrdata[15:8] <= #tm_prop ~mem_datain_i[15:8];

VIOL:G5.3.2.1: hardcoding found type1
1307: hrdata[15:8] <= #tm_prop mem_datain_i[15:8];

VIOL:G5.3.2.1: hardcoding found type1
1315: hrdata[7:0] <= #tm_prop ~mem_datain_i[7:0];

VIOL:G5.3.2.1: hardcoding found type1
1319: hrdata[7:0] <= #tm_prop mem_datain_i[7:0];

VIOL:G5.3.2.1: hardcoding found type1
1329: hrdata[31:24] <= #tm_prop mem_datain_i[31:24];

VIOL:G5.3.2.1: hardcoding found type1
1334: hrdata[23:16] <= #tm_prop mem_datain_i[23:16];

VIOL:G5.3.2.1: hardcoding found type1
1339: hrdata[15:8] <= #tm_prop mem_datain_i[15:8];

VIOL:G5.3.2.1: hardcoding found type1
1344: hrdata[7:0] <= #tm_prop mem_datain_i[7:0];

VIOL:G5.3.2.1: hardcoding found type1
1370: hwdata_businvert_capture[35:27] <= #tm_prop
bus_invert_coder(hwdata[31:24],mem_dataout_o[31:24],mem_invertbits_o[3]);

VIOL:G5.3.2.1: hardcoding found type1
1371: hwdata_businvert_capture[26:18] <= #tm_prop
bus_invert_coder(hwdata[23:16],mem_dataout_o[23:16],mem_invertbits_o[2]);

VIOL:G5.3.2.1: hardcoding found type1
1372: hwdata_businvert_capture[17:9] <= #tm_prop
bus_invert_coder(hwdata[15:8],mem_dataout_o[15:8],mem_invertbits_o[1]);

VIOL:G5.3.2.1: hardcoding found type1
1373: hwdata_businvert_capture[8:0] <= #tm_prop
bus_invert_coder(hwdata[7:0],mem_dataout_o[7:0],mem_invertbits_o[0]);

VIOL:G5.3.2.1: hardcoding found type1
1379: hwdata_businvert_capture[35:0] <= 36'b0;

VIOL:G5.3.2.1: hardcoding found type1
1407: mem_dataout_o[31:0] <= #tm_prop 32'h00000000;

VIOL:G5.3.2.1: hardcoding found type1
1408: mem_invertbits_o[3:0] <= #tm_prop 4'h0;

VIOL:G5.3.2.1: hardcoding found type1
1421: mem_dataout_o[31:24] <= #tm_prop hwdata_businvert_capture[34:27];

VIOL:G5.3.2.1: hardcoding found type1
1426: mem_dataout_o[23:16] <= #tm_prop hwdata_businvert_capture[25:18];

VIOL:G5.3.2.1: hardcoding found type1
1431: mem_dataout_o[15:8] <= #tm_prop hwdata_businvert_capture[16:9];

VIOL:G5.3.2.1: hardcoding found type1
1436: mem_dataout_o[7:0] <= #tm_prop hwdata_businvert_capture[7:0];

VIOL:G5.3.2.1: hardcoding found type1
1444: mem_dataout_o[31:24] <= #tm_prop hwdata[31:24];

VIOL:G5.3.2.1: hardcoding found type1
1448: mem_dataout_o[23:16] <= #tm_prop hwdata[23:16];

VIOL:G5.3.2.1: hardcoding found type1
1452: mem_dataout_o[15:8] <= #tm_prop hwdata[15:8];

VIOL:G5.3.2.1: hardcoding found type1
1456: mem_dataout_o[7:0] <= #tm_prop hwdata[7:0];

VIOL:G5.3.2.1: hardcoding found type2
1551: wait_state_cntr <= #tm_prop 4'b0000;

VIOL:G5.3.2.1: hardcoding found type2
1558: 3'b100: // Load read wait state value

VIOL:G5.3.2.1: hardcoding found type2
1562: 3'b010: // Load write wait state value

VIOL:G5.3.2.1: hardcoding found type2
1566: 3'b001: // enable counter

VIOL:G5.3.2.1: hardcoding found type2
1573: assign b_wait_states_left = (wait_state_cntr == 4'b0000) ? 0 : 1;