IP Qualification Guidelines (I)
Design Guidelines & 應用實例

Speaker: 陳佩蕙

STC/ITRI

Mar. - Apr., 2004

Outline

- **Motivation**
- **IP Generic Design Flow**
- **Severity Level Definition**
- **The Structure of Design Guidelines**
  - Soft IP
    - Verilog HDL Coding Guidelines
    - Design Style Guidelines
    - Synthesis Script Design Guidelines
  - Hard IP
    - Design Style Guidelines
    - Physical Design Consideration
- **Lint Tools Certification**
- **Case Study**
- **Conclusion**

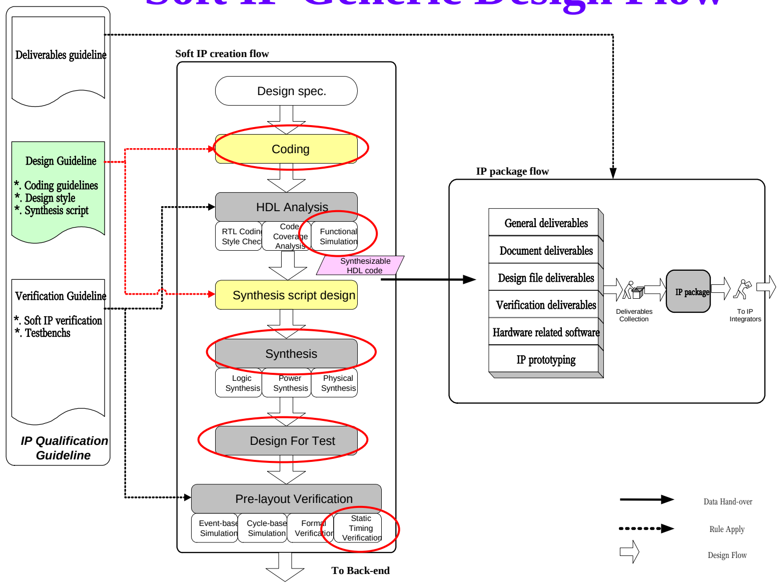
Motivation

- **Challenges for SoC era**
  - Design is more and more complicated
  - Device geometries is more and more smaller
  - Time to market
  - Need off-the-shelf IP solutions for integration
- **Use of various IPs leads to many problems**
- **IP Providers 和 IP Users 皆需要有一套共通標準來確保 IP 的Quality，以降低 IP reuse 時的風險，提高 SoC 成功的機會。**

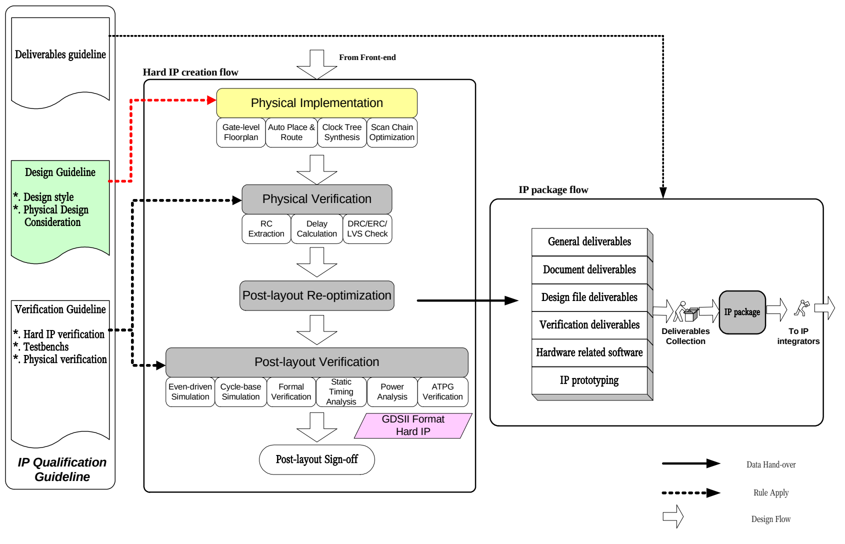
IP Quality 是 SoC 成敗關鍵

Outline

Soft IP Generic Design Flow



Hard IP Generic Design Flow



Outline

- **Motivation**
- **IP Generic Design Flow**
- ***Severity Level Definition***
- **The Structure of Design Guidelines**
  - Soft IP
    - Verilog HDL Coding Guidelines
    - Design Style Guidelines
    - Synthesis Script Design Guidelines
  - Hard IP
    - Design Style Guidelines
    - Physical Design Consideration
- **Lint Tools Certification**
- **Case Study**
- **Conclusion**

Severity Levels Definition

- **Mandatory 1 (M1) –**
  Rules must be followed. If not, design must be fixed.
- **Mandatory 2 (M2) –**
  Rules should be followed. If not, documentation must be provided.
- **Recommended (R) –**
  Rules are recommended to be followed in the design.

Outline

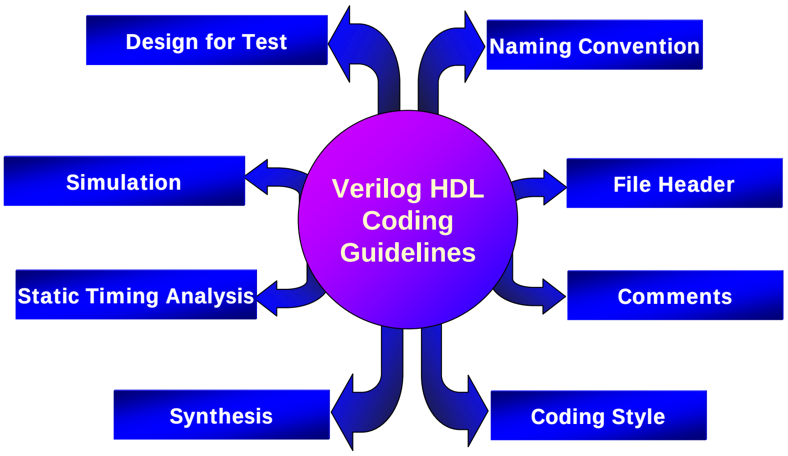
- **Motivation**
- **IP Generic Design Flow**
- **Severity Level Definition**
- ***The Structure of Design Guidelines***
  - *Soft IP*
    - *Verilog HDL Coding Guidelines*
    - Design Style Guidelines
    - Synthesis Script Design Guidelines
  - Hard IP
    - Design Style Guidelines
    - Physical Design Consideration
- **Lint Tools Certification**
- **Case Study**
- **Conclusion**

Why Coding Guidelines?

- CAD tools is essentiality to be involved in each stage of the ASIC design flow
- However, every CAD tool has its own capability and limitations
- Designer must be aware of the capability and limitations of each CAD tools he used
- Coding Guidelines are required for specific tools to obtain best performance and avoid exhausted iteration

Tool is nothing!
but without tool, you can't do anything!

Verilog HDL Coding Guidelines



Naming Convention Rule Lists

	Naming Convention	Level
[S.NC.1]	A file must contain at most one module unit	M2
[S.NC.2]	File name should be identical to design unit name	R
[S.NC.3]	Only alphanumeric and underscore are allowed in naming	M1
[S.NC.4]	Names starting with a letter	M2
[S.NC.5]	Names can not be distinguishable in letter case only	M1
[S.NC.6]	Use underscores to separate names composed of several words	R
[S.NC.7]	Use consistent signal names throughout the hierarchy	M2
[S.NC.8]	Parameters and text macros for constant names are uppercase letters	M2
[S.NC.9]	Signals, constructs and instance names are lowercase letters	M2
[S.NC.10]	Use meaningful names	R
[S.NC.11]	Consistent clock signal name	M2
[S.NC.12]	Consistent reset signal name	M2
[S.NC.13]	Consistent active-low signal name	M2
[S.NC.14]	For FSM variables, naming in <fsm_cs>, <fsm_ns>	R
[S.NC.15]	For latch variables, end in _lat	R
[S.NC.16]	For tri-state signals, end in _z	R
[S.NC.17]	For asynchronous signals, end in _a	R
[S.NC.18]	Verilog and VHDL keywords are prohibited	M1
[S.NC.19]	The length of signal names does not exceed 32 characters	M2
[S.NC.20]	Descriptive and consistent abbreviation	R
[S.NC.21]	Document abbreviations and additional naming conventions	R
[S.NC.22]	Instance names must be identical to the module name	M2
[S.NC.23]	Consistent ordering of bits for describing multibit buses	M2

Naming Convention

[S.NC.1]	A file must contain at most one module unit	M2

- Design structure is more distinct

[S.NC.2]	File name should be identical to design unit name	R

- <*design_unit_name*>.v, where, <*design_unit_name*> is the name of the design unit (i.e., module name).
- `arbiter.v`: Verilog design file for module arbiter.

[S.NC.3]	Only alphanumeric and underscore are allowed in naming	M1

- [A-Z, a-z, 0-9, _]. However, it is prohibited for using consecutive underscores.

Naming Convention

[S.NC.4]	Names starting with a letter	M2

- Names starting with an underscore may cause syntax error and is not allowed with some EDA tools.

[S.NC.5]	Names can not be distinguishable in letter case only	M1

- Do not use `ABC`, `abc`, and `aBc` to represent three different names.

[S.NC.6]	Use underscores to separate names composed of several words	M2

- RAM address bus can use `ram_addr` for naming.

Naming Convention

[S.NC.7]	Use consistent signal names throughout the hierarchy	M2

```
block1 u_block1 (
        .rst_n          (rst_n),
        .clk            (clk),
        .addr           (addr),
        .wr_data        (wr_data),
        . . .,
        );
```

```
block1 u_block1 (
        .reset_n        (rst_n),
        .clock          (clk),
        .address        (addr),
        .WR_data        (wr_data),
        . . .,
        );
```

[S.NC.8]	Parameters and text macros for constant names are uppercase letters	M2

- `parameter   DATA_WIDTH = 32;`
- `` `define     AHB_TRANS_SEQ    2'b11 ``

Naming Convention

[S.NC.9]	Ports, signals, constructs and instance names are lowercase letters	M2

- Ports

```
input       rst_n;
output      wr_data;
```

- Signals

```
reg [7:0]   a_reg;
wire        we_n;
```

- Constructs

```
task        busy;
function    call_addr;
```

- Instance names

```
block1  u_block1 (…
```

Naming Convention

[S.NC.10]	Use meaningful names	R

- Naming should represent the purpose of the item.

[S.NC.11]	Consistent clock signal name	M2

- Recommend using *clk* as a substring for the clock signal naming.

[S.NC.12]	Consistent reset signal name	M2

- Recommend using *rst* as a substring for the reset signal naming.

[S.NC.13]	Consistent active-low signal name	M2

- Recommend using _*n* as the suffix for the active-low signal naming.

Naming Convention

[S.NC.14]	For FSM variables, naming in <fsm_cs>, <fsm_ns>	R
[S.NC.15]	For latch variables, end in _lat	R
[S.NC.16]	For tri-state signals, end in _z	R
[S.NC.17]	For asynchronous signals, end in _a	R
[S.NC.18]	Verilog and VHDL keywords are prohibited	M1
[S.NC.19]	The length of signal names does not exceed 32 characters	M2

[S.NC.20]	Descriptive and consistent abbreviation	R
[S.NC.21]	Document abbreviations and additional naming conventions	R

- Exp. recommend using `ram_addr` instead of `ra` to represent ram address

Naming Convention

[S.NC.22]	Instance names must be identical to the module name	M2

Module instantiated once:
```
decoder u_decoder (…);
```
Module with multiple instantiated:
```
mux4 u_mux4_1 (…);
mux4 u_mux4_2 (…);
```

[S.NC.23]	Consistent ordering of bits for describing multibit buses	M2

- Recommend using `[x:0]` in Verilog.

File Header Rule Lists

	File Header	Level
[S.FH.1]	**Each file must include a file header**	**M1**
[S.FH.2]	**File header must start and end in file header boundary tags**	**M1**
[S.FH.3]	**Necessary fields for a file header**	
[S.FH.3.1]	Include file name	**M2**
[S.FH.3.2]	Include author and contact information	**M2**
[S.FH.3.3]	Include a release version and version description	**M2**
[S.FH.3.4]	Include a release date	**M2**
[S.FH.3.5]	Include a purpose section	**M2**
[S.FH.3.6]	Include a parameter description	**M2**
[S.FH.4]	**Other header reuse issues**	
[S.FH.4.1]	Include the **reset strategy** description	**R**
[S.FH.4.2]	Include the **clock strategy** description	**R**
[S.FH.4.3]	Include the **critical timing** description	**R**
[S.FH.4.4]	Include the **test feature** description	**R**
[S.FH.4.5]	Include the **asynchronous interface** description **if existed**	**R**
[S.FH.4.6]	Include the **scan methodology** description	**R**
[S.FH.5]	**Additional construct should include a header**	**M2**
[S.FH.6]	**Additional construct header must start and end in construct header boundary tags**	**M2**
[S.FH.7]	**Necessary fields for additional construct header**	
[S.FH.7.1]	Include construct name and data type of return values	**M2**
[S.FH.7.2]	Include construct type	**M2**
[S.FH.7.3]	Include a purpose section	**M2**
[S.FH.7.4]	Include a parameter description	**M2**

File Header

```
//+FHDR------------------------------------------------------------------------
// (C) Copyright Company Industrial technology Research Institute (ITRI)
// All Right Reserved
//-----------------------------------------------------------------------------
// FILE NAME: decoder.v
// AUTHOR: phchen
// CONTACT INFORMATION: phchen@itri.org.tw
//-----------------------------------------------------------------------------
// RELEASE VERSION: V1.0
// VERSION DESCRIPTION: First Edition no errata
//-----------------------------------------------------------------------------
// RELEASE DATE: 10-31-2002
//-----------------------------------------------------------------------------
// PURPOSE:  HADDR is used to decoded for generating HSELx signals to
//           select outputs to the AHB system slaves, and controls the read
//           data multiplexer.
//-----------------------------------------------------------------------------
// PARAMETERS:
//PARAMETER NAME        RANGE         DESCRIPTION           DEFAULT VALUE
//ADDR_WIDTH            [32,16]       width of the address   32
//-FHDR------------------------------------------------------------------------
```

Additional Construct Header

```
//+CHDR--------------------------------------------------------------
// CONSTRUCT NAME & RETURN VALUES:
// task_write(haddr, hwrite, hwdata)
//-------------------------------------------------------------------
// TYPE: task
//-------------------------------------------------------------------
// PURPOSE:  input address, write, data signal to execute system write
//-------------------------------------------------------------------
// PARAMETERS:
//PARAMETER NAME        RANGE          DESCRIPTION        DEFAULT VALUE
//DATA_WIDTH           [32,16]         width of the data  32
//-CHDR--------------------------------------------------------------
```

Comments Rule Lists

	Comments	Level
[S.COM.1]	Use comments for port declarations	M1
[S.COM.2]	Use comments for internal signal declarations	R
[S.COM.3]	Use comments for functional sections	R
[S.COM.4]	Use one-line comments instead of multiple-line comments	R
[S.COM.5]	Use comments for cell instantiations	M2
[S.COM.6]	Delete unnecessary codes	R
[S.COM.7]	Use comments for synthesis directives	R
[S.COM.8]	Use comments for compiler directives	R

- Aid IP users to understand the design concept of original designer.
- Improve the readability and portability of the code.
- Help for IP maintenance and management.

Comments

[S.COM.1]	Use comments for port declarations	M1

```
input          rst_n; // system reset
input          clk; // system clock
output [31:0]  wr_data; // system write data bus
```

[S.COM.2]	Use comments for internal signal declarations	R

- Internal signal, ex. `wire` and `reg` should have descriptive comments.

[S.COM.3]	Use comments for functional sections	R

- Functional `always` blocks or `assign` statements should have descriptive comments.

Comments

[S.COM.4]	Use one-line comments instead of multiple-line comments	M2

- Use one-line comments // , instead of multiple-line comments /*…..*/.

[S.COM.5]	Use comments for cell instantiations	M2

```
and u_and1(A_out, SEL_b, A); //explain the reason
```

[S.COM.6]	Delete unnecessary codes	R

- Unnecessary codes represents old code, or any unused code.
- Keep the HDL code as clean as possible.

Comments

[S.COM.7]	Use comments for synthesis directives	M2

```
case (select) // synopsys full_case
  3'b000: y = 1'b1;
  3'b001: y = 1'b0;
  3'b101: y = 1'b0;
  3'b111: y = 1'b1;
endcase
```

[S.COM.8]	Use comments for compiler directives	M2

```
`ifdef FOR_SIMULATION // show some test messages for debug
…
`else // NORNAL CONFIGURATION, i.e., for synthesis
   …
`endif // end FOR_SIMULATION
```

Coding Style Rule Lists

	Coding Style	Level
[S.CS.1]	Code should be aligned in a tabular format	M2
[S.CS.2]	Use space instead of tab stops for code indentation	M2
[S.CS.3]	Use a separate line for each HDL statement	M1
[S.CS.4]	Use a separate line for each port declaration	M1
[S.CS.5]	Preserve port order	M2
[S.CS.6]]	Declare all internal nets in one section	R
[S.CS.7]	Keep line length within 72 characters	R
[S.CS.8]	Use parameters over the 'define directives to specify configurable constants	R
[S.CS.9]	Preserve relationships between constants	R
[S.CS.10]	Use explicit port mapping instead of positional association in module instantiation	M1
[S.CS.11]	Avoid expression in port connections	M1
[S.CS.12]	Use parameters for FSM state encoding	M1
[S.CS.13]	Use parentheses in complex equations	M2
[S.CS.14]	Wire must be explicitly declared	M1
[S.CS.15]	Operand size must match	M1
[S.CS.16]	Expression in condition should be 1- bit value	R
[S.CS.17]	Finite State Machine (FSM) coding style	R

Coding Style

[S.CS.1]	Code should be aligned in a tabular format	M2

- Aligns the HDL keyword items of the same kind (`module/endmodule`, `always` block `begin/end`, `if/else` statement, `case` statement, etc.)

[S.CS.2]	Use space instead of tab stops for code indentation	M2

- Tabs stops must not be used because they represent different number of spaces from one system to another.

[S.CS.3]	Use a separate line for each HDL statement	M1

Use:
```
out1 = a1 & b1; // AND operation
out2 = a2 | b2; // OR operation
```
Do not use:
```
out1 = a1 & b1; out2 = a2 | b2;
```

Coding Style

[S.CS.4]	Use a separate line for each port declaration	M1

Use:
```
  input A; // port A description
  input B; // port B description
```
Do not use:
```
  input A, B;
```

[S.CS.5]	Preserve port order	M2

- The order of port declaration keep the same as the port order listed in the module declaration

[S.CS.6]	Declare all internal nets in one section	R

- The declaration of all internal nets should follow the port I/O declaration

Coding Style

[S.CS.7]	Keep line length within 72 characters	R

- Line length should be limited within 72 characters to reserves space to put the additional line number.

[S.CS.8]	Use parameters over the 'define directives to specify configurable constants	R

- If a `` `define `` is used, it should be undefined using `` `undef ``. Otherwise, it may cause re-definition error message if other module also exists the same `` `define `` constant.

[S.CS.9]	Preserve relationships between constants	R

```
parameter      HALFWORD = 16;
parameter      WORD = (2 * HALFWORD);
```

Coding Style

[S.CS.10]	Use explicit port mapping instead of positional association in module instantiation	M1

Use:
```
adder u_adder(
      .a   (A),
      .b   (B),
      .sum (SUM)
        );
```

Do not use:
```
adder u_adder(
        A,
        B,
        SUM
          );
```

[S.CS.11]	Avoid expression in port connections	M1

```
foo u_foo (
     .bad         ((m & n) ^ ((p | q) > 4'hc)),
     .good        (good)); // expressions in port connections
```

Coding Style

[S.CS.12]	Use parameters for FSM state encoding	M1

```
parameter [1:0] // symbolic state_vector
  ST_IDLE = 2'b00,
  ST_READ = 2'b01,
  ST_WRITE = 2'b10,
  ST_WAIT = 2'b11;
```

[S.CS.13]	Use parentheses in complex equations	M2

```
assign req_out = ((req_ext & en) | req_int);
```

Coding Style

[S.CS.14]	Wire must be explicitly declared	M1

```
wire int_signal_a; // internal signal for block1 signal_a
wire int_signal_b; // internal signal for block1 signal_b
block u_block(
     .signal_a (int_signal_a),
     .signal_b (int_signal_b
              );
```

- Do not count on the language-specific implication.
- Follow the strong type-checking principles that are enforced by most modern programming languages.

Coding Style

[S.CS.15]	Operand size must match	M1

```
  wire [32:0] signal_a; // signal_a with 33 bit width
  wire [7:0]  signal_b; // signal_b with 8 bit width
```

Do not use:

```
 signal_x <= signal_a & signal_b;
```

- Potential error of bit zero-extended

```
module test_width(in, enable, out);
  parameter WIDTH = 33;
  input  [WIDTH-1:0] in;
  input  enable;
  output [WIDTH-1:0] out;

  assign out = (enable) ? in : 'bz;
endmodule
```

Coding Style

[S.CS.16]	Expression in condition should be 1- bit value	R

Do not use:
```
  if (data)
   data_enable = 1;
```
Use:
```
  if (data > 0)
   data_enable = 1;
```

[S.CS.17]	Finite State Machine (FSM) coding style	R

- Using two-process coding style for Mealy state machine and three-process style for Moore state machine.

Synthesis Rule Lists

	Synthesis	Level
[S.SYN.1]	**Use only synthesizable statement**	**M1**
[S.SYN.1.1]	Waveform statement are prohibited	**M1**
[S.SYN.1.2]	System task and function for simulation (e.g. $display, $monitor, $printf, …) are prohibited	**M1**
[S.SYN.1.3]	Wait statement and # delay statement are prohibited	**M1**
[S.SYN.1.4]	Data type of real and event are prohibited	**M1**
[S.SYN.1.5]	Only one clock per always sensitivity list	**M1**
[S.SYN.1.6]	Loop must be in a static range	**M1**
[S.SYN.2]	**Avoid any embedded synthesis scripts**	**M2**
[S.SYN.3]	**Avoid full_case and parallel_case synthesis directive statement**	**M2**
[S.SYN.4]	**Specify code fragment for combinational logic completely**	**M2**
[S.SYN.5]	**Verilog primitives are prohibited**	**M1**
[S.SYN.6]	**All unused module inputs must be driven**	**M1**
[S.SYN.7]	**All unused module outputs should be connected**	**R**
[S.SYN.8]	**Avoid top level glue logic**	**M2**
[S.SYN.9]	**Incomplete case statements should have default case assignments**	**M2**
[S.SYN.10]	**Assign a default state to the state machines**	**R**

Synthesis

[S.SYN.1]	Use only synthesizable statement	M1
[S.SYN.2]	Avoid any embedded synthesis scripts	M2

[S.SYN.3]	Avoid full_case and parallel_case synthesis directive statement	M2

```
case (select) // synopsys full_case
  3'b000: y = 1'b1;
  3'b001: y = 1'b0;
  3'b101: y = 1'b0;
  3'b111: y = 1'b1;
endcase
```

```
always @(signal_1 or signal_2
          or signal_3)
begin
  case (1) //synopsys parallel_case
    signal_1: value = 2'b01;
    signal_2: value = 2'b10;
    signal_3: value = 2'b11;
  endcase
```

Synthesis

[S.SYN.4]	Specify code fragment for combinational logic completely	M2

Missing condition:

```
module SYN3_1 (a, data_out);
input a;
output data_out;
always @ (a)
  begin
    if (a == 1'b1)
      data_out = 1'b0;
  end
endmodule
```

Missing assignment:

```
always @(condition)
begin
  case (condition)
    2'b00: data_out = 4'b0000;
    2'b01:
    2'b10: data_out = 4'b0011;
    2'b11:
  endcase
end
```

Synthesis

[S.SYN.5]	Verilog primitives are prohibited	M1

- Verilog `user-Defined-Primitive` (UDP)
- Verilog built-in primitive

[S.SYN.6]	All unused module inputs must be driven	M1

- Avoid any floated input ports. Floating input ports may result in fatal error of chip.

Synthesis

[S.SYN.7]	All unused module outputs should be connected	R

- Unconnected output ports may not affect the functionality but make users confuse.

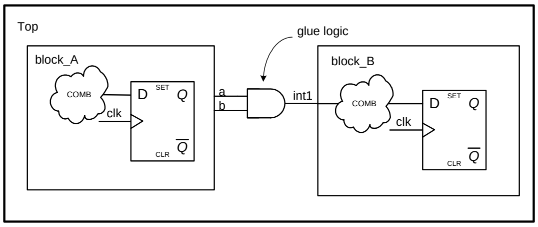
```
Module sub_block
(rst_n, clk, multiplier_out, divider_out);

  input  rst_n, clk;
  output [15:0] multiplier_out;
  output [15:0] divider_out;
```

```
module top (rst_n, clk, control,
                top_out);
    input  rst_n;
    input  clk;
    output [7:0] top_out;
    wire   [7:0]  multiplier_out_nc;
    wire   [15:0] divider_out_nc;

sub_block u_sub_block(
       .rst_n              (rst_n),
       .clk                (clk),
       .multiplier_out_nc  ({top_out,
                          multiplier_out_nc}),
       .divider_out_nc     (divider_out_nc)
       );
```

```
module top (rst_n, clk, control,
                top_out);
    input  rst_n;
    input  clk;
    output [7:0] top_out;
    wire   [7:0]  temp;

sub_block u_sub_block(
       .rst_n              (rst_n),
       .clk                (clk),
       .multiplier_out_nc  ({top_out, temp})
       );
```

Synthesis

[S.SYN.8]	Avoid top level glue logic	M2



- The top level of the design should contain only at leaf levels of the hierarchy tree.
- Synthesis optimization is limited because the top level glue logic can not be merged with the combinational logic inside any modules.

Synthesis

[S.SYN.9]	Incomplete case statements should have default case assignments	M2

- If case statements have no default clause values and not all the cases are covered, unexpected latches may be inferred.

```
always @(condition)
  begin
    case (condition)
      2'b00：data_out = 4'b0000；
      2'b10：data_out = 4'b0010；
      2'b11：data_out = 4'b0011；
      default：data_out = 4'b0000；
    endcase
  end
```

[S.SYN.10]	Assign a default state to the state machines	R

- Prevent state machines from falling into disorder.

工業技術研究院
系統晶片技術中心
Industrial Technology Research Institute
SoC Technology Center

Static Timing Analysis Rule Lists

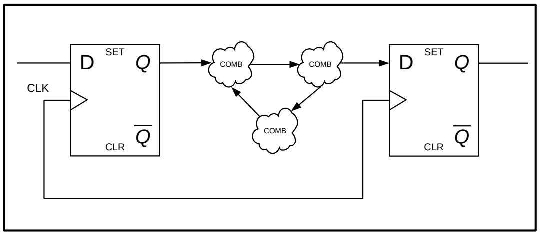
	Static Timing Analysis	Level
[S.STA.1]	Avoid Combinational feedback loop	M2
[S.STA.2]	Using synchronous design practices	M2
[S.STA.3]	Simplify register clock origin	M2
[S.STA.4]	Avoid multicycle path and false path	M2
[S.STA.5]	Avoid clock as data	M2
[S.STA.6]	Avoid using latches	M2

工業技術研究院
系統晶片技術中心
Industrial Technology Research Institute
SoC Technology Center

Static Timing Analysis

[S.STA.1]	Avoid Combinational feedback loop	M2

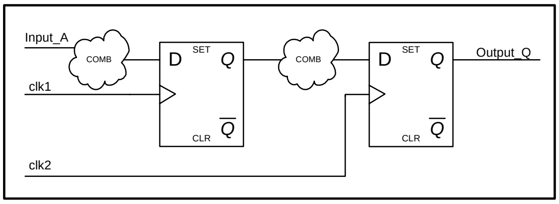
- Combinational feedback loop will cause the problems of unpredictable timing on static timing analysis.
- Two ways to handle combinational feedback loop
  - You can break loops by disabling timing arcs `set_disable_timing` command
  - STA tool will automatically breaks them.



Static Timing Analysis

[S.STA.2]	Using synchronous design practices	M2

- Asynchronous design should be used only when unavoidable, and should be separated from the synchronous design.
- Some static timing analysis tools do not handle the asynchronous design adequately.
- Asynchronous design needs to be verified carefully for its functionality and timing.

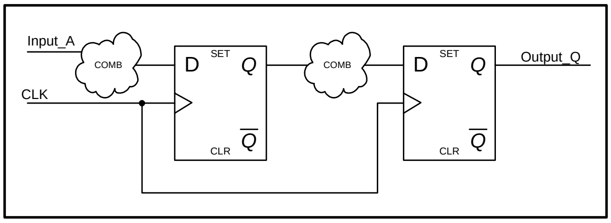


```
sta_shell> set_false_path –from clk1 –to clk2
```

Static Timing Analysis

[S.STA.3]	Simplify register clock origin	M2

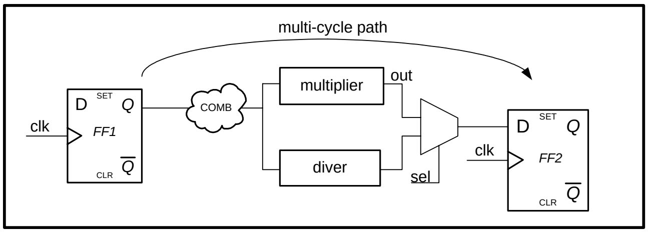
- Avoid connecting internally generated signals or gated clocks to the register clock.
- If the internally generated clock or the gated clock is unavoidable in a design, e.g., a low-power design, separate this portion of circuitry in an individual module at the top level of the design.
- The register clock origin should be simplified to prevent metastability problems on the data being registered



Static Timing Analysis

[S.STA.4]	Avoid multicycle path and false path	M2

- Timing exceptions should be marked by specifying the start and end points of these paths.
- Timing exceptions lead to longer compile times, try to minimize the total number of timing exceptions

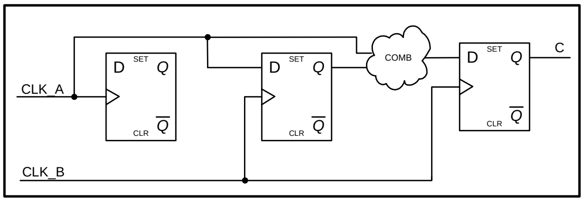


```
sta_shell> set_multicycle_path 3 –from FF1/clk –through multiplier/out –to FF2/D
```

Static Timing Analysis

[S.STA.5]	Avoid clock as data	M2

- Clocks fed to data inputs of registers directly or indirectly could lead to timing violations and race conditions.



[S.STA.6]	Avoid using latches	M2

- Latches are not recommended in your design.
- IF any unavoidable latches are inferred in your design, you must provide documentation to describe the special timing requirements of each latch.

Simulation Rules

	Simulation	Level
[S.SIM.1]	Use non-blocking assignments in sequential always block	M1
[S.SIM.2]	Use blocking assignments in combinational always block	M1
[S.SIM.3]	Avoid missing sensitivity lists in combinational always block	M1
[S.SIM.4]	Avoid redundant sensitivity lists	M1
[S.SIM.5]	Initialize control storage elements	M2
[S.SIM.6]	Do not assign signals don't care (x) value	M1

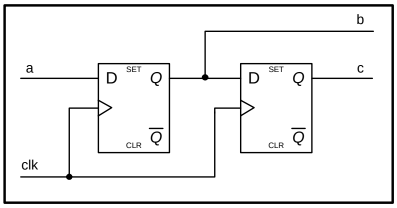
- **Avoid simulation mismatch**
- **Simulator independent**

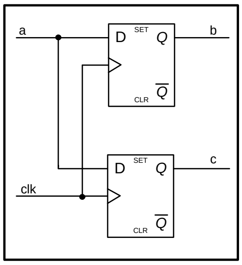
Simulation

[S.SIM.1]	Use non-blocking assignments in sequential always block	M2

```
always @(posedge clk)
  begin
    b <= a;
    c <= b;
  end
```

```
always @(posedge clk)
  begin
    b = a;
    c = b;
  end
```





Simulation

[S.SIM.2]	Use blocking assignments in combinational always block	M1

- Use blocking assignments in always blocks that are intended to generate combinational logic.
- Non-blocking assignments in combinational `always` blocks may cause artificial race conditions which will disappear after synthesis.
- Blocking assignments ensure the circuits will be evaluated correctly and prevent simulation result mismatches between RTL and gate-level circuits.

Simulation

[S.SIM.3]	Avoid missing sensitivity lists in combinational always block	M1

- Missing signals in sensitivity list will cause the mismatch of pre-synthesis simulation and post-synthesis simulation.

```
always @ (a)
  begin
    Z = a or b
  end
```

[S.SIM.4]	Avoid redundant sensitivity lists	M1

- The presence of a signal in the sensitivity list that is not used in the process may cause unneeded evaluations in simulation and result in longer simulation times.

```
always @ (a or b or c)
  begin
    Z = a or b
  end
```

Simulation

[S.SIM.5]	Initialize control storage elements	M2

- All control storage elements, including all latches and registers, in a control path for control circuitry should be set or reset in the beginning.
- Control path represents that signal output from register and pass through the conditional logic for control.
- The initial value (e.g., “0” or “X”) of each uninitialized latch or register is different from one simulator to another. The simulation result may be simulator dependent.

Simulation

[S.SIM.6]	Do not assign signals don't care (x) value	M1

- Prevent *x* propagation through the circuitry, especially for the control path.
- If there exists *x* value in a control path, it may potentially cause the chain of unknown state for all related circuits in simulation.

```
module control_path (rst_n, clk, select, data_out);
input        rst_n;
input        clk;
input  [2:0] select;
output       data_out;
reg    [2:0] Y;
always @ (select)
   begin
      case (select)
         3'b000 : Y = 3'b001;
         3'b001 : Y = 3'b010;
         3'b010 : Y = 3'b100;
         3'b101 : Y = 3'b101;
         default: Y = 3'bxxx;
      endcase
   end
```

```
always @ (posedge clk or negedge
rst_n)
   begin
      if (!rst_n)
          data_out <= 1’b0;
      else if (Y == 3’b001)
          data_out <= 1’b1;
      else
          data_out <= 1’b0;
   end
endmodule
```

Simulation

[S.SIM.7]	Avoid using delay assignment	M2

- Avoid using both inter- and intra-assignment delays.
- Current synthesis tools ignore these delay assignments. Hence, these delays potentially cause the result mismatch between RTL and gate-level simulation.

```
#1 outp = a & b; // inter-assignment delay
always @(posedge clk or negedge rst_n)
  if(rst_n == 1’b0)
    outp_r <= #1 4’b0; // intra-assignment delay
  else
    outp_r <= #1 outp_nxt; // intra-assignment delay
```

Design for Test Rule Lists

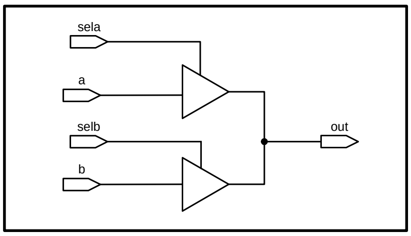
	Design for Test	Level
[S.DFT.1]	Avoid tri-state devices	M2
[S.DFT.2]	Avoid bi-directional nets	M2
[S.DFT.3]	Avoid latches	M2
[S.DFT.4]	Avoid using both edges of clocks	M2
[S.DFT.5]	Avoid gating clocks	M2
[S.DFT.6]	Avoid internal generated clocks	M2
[S.DFT.7]	Avoid internal generated set/reset signals	M2
[S.DFT.8]	Avoid using clocks and set/reset signals as data	M2
[S.DFT.9]	Avoid combinational loop	M2
[S.DFT.10]	Avoid constant inputs or floating outputs	R

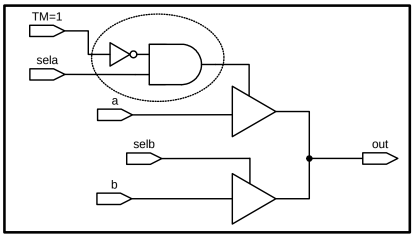
- **Be able to do full-scan insertion**
- **Obtain the maximum fault coverage**

Design for Test

[S.DFT.1]	Avoid tri-state devices	M2

- Multiple drivers will possibly cause bus contention, it is recommended to avoid them.
- It is recommended to be fixed at RTL.

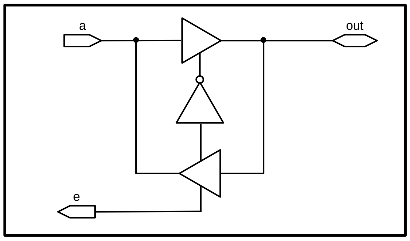




Design for Test

[S.DFT.2]	Avoid bi-directional nets	M2

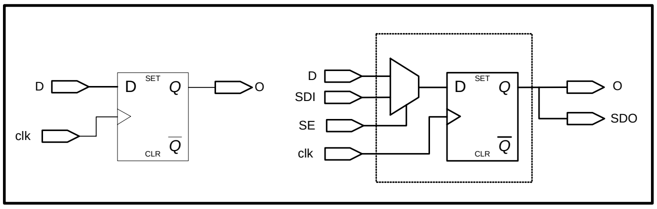
- It is recommended to avoid the bi-directional nets for the digital logic design.
- Bi-direction nets make ATPG tools confuse the signal direction and lose the capability to generate the test patterns for the digital circuitry in which the bi-directional nets involve



Design for Test

[S.DFT.3]	Avoid latches	M2

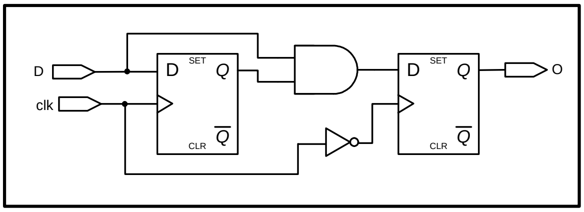
- It is recommended to avoid latches. Scannable registers are recommended.
- Latches are usually non-scannable and non-transparent without any special control circuit.
- Significant fault coverage might be lost if latches are intensively used.



Design for Test

[S.DFT.4]	Avoid using both edges of clocks	M2

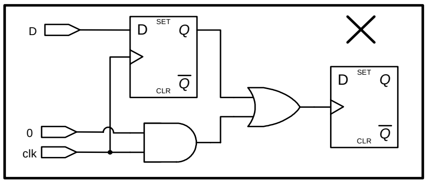
- Use of mixed clock edges increases the difficulty to chain these DFFs.
- There are two solutions proposed up to now.
  - One is to insert the protection circuit between the positive-edge and negative-edge DFF.
  - The other is to separate positive-edge and negative-edge DFFs into different groups for different scan chains.



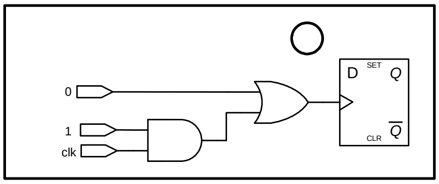
Design for Test

[S.DFT.5]	Avoid gating clocks	M2

- The clock ports of the flip-flops should be controlled directly by one primary clock input.
- The gated clock potentially obstructs the scan shift operations. That is, it's not friendly to DFT.



Serious gated clock can not be controlled by certain pattern

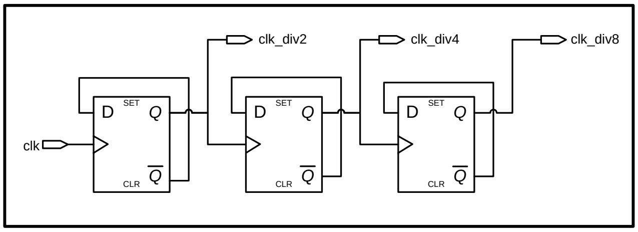


Gated clock but can be controlled by certain pattern

Design for Test

[S.DFT.6]	Avoid internal generated clocks	M2

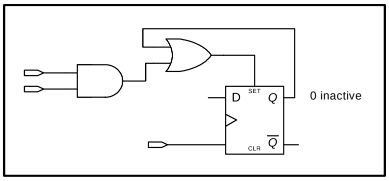
- Internal generated clock makes the sequential element un-synchronous and obstructs the scan operation.
- Hence, this construct significantly reduces the fault coverage.



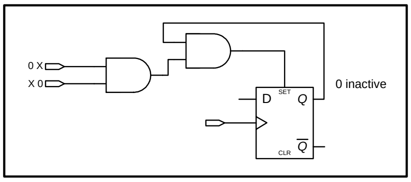
Design for Test

[S.DFT.7]	Avoid internal generated set/reset signals	M2

- The set/reset ports of the flip-flops should be controlled directly by primary inputs to set to the inactive value for the scan operation.
- The internally generated set/reset signal can potentially destroy the integrity of the shift data during the scan shift operation.
- Because the scan operation requires the set/reset ports to be held in inactive states in order to guarantee the shifted data are not destroyed.



Uncontrollable preset or clear

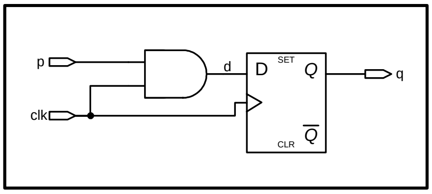


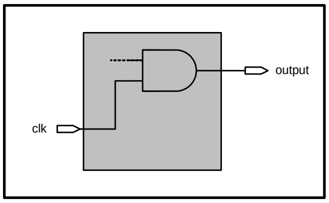
Controllable preset or clear

Design for Test

[S.DFT.8]	Avoid using clocks and set/reset signals as data	M2

- It is recommended that data should not be generated from any clock signal or ser/reset signals.
- Full-scan can always operate abnormally and unsafely in this construct.

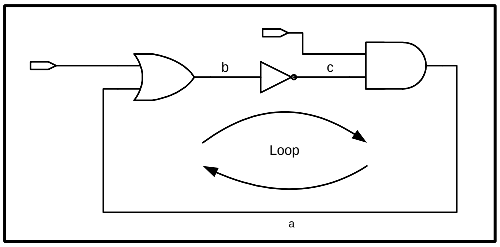




Design for Test

[S.DFT.9]	Avoid combinational loop	M2

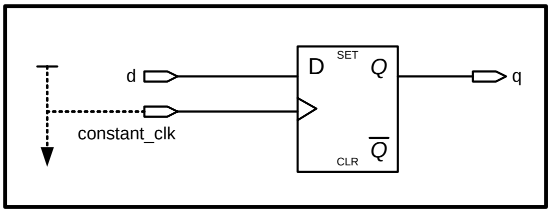
- The combinational loop circuit has the feedback, the ATPG can not determine the logic values in the loop.
- The combinational loop is usually considered to be un-testable if the loop is not broken, so the combinational loop will decrease the fault coverage.



Design for Test

[S.DFT.10]	Avoid constant inputs or floating outputs	R

- It usually occurs when we use the existing functional blocks which are designed and verified, or integrate different cores to synthesize.
- Because we can not control the constant inputs or observe the floating outputs.
- It will reduce the controllability and the observability respectively, the fault coverage is reduced.



Outline

- **Motivation**
- **IP Generic Design Flow**
- **Severity Level Definition**
- **The Structure of Design Guidelines**
  - Soft IP
    - Verilog HDL Coding Guidelines
    - *Design Style Guidelines*
    - Synthesis Script Design Guidelines
  - Hard IP
    - Design Style Guidelines
    - Physical Design Consideration
- **Lint Tools Certification**
- **Case Study**
- **Conclusion**

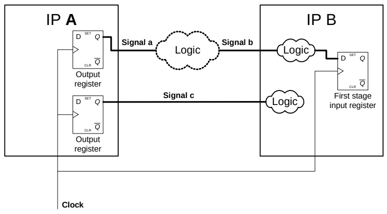
Design Style Guidelines Rule Lists

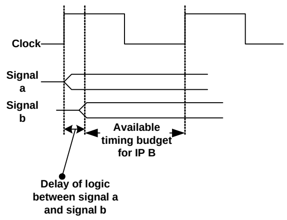
	Design Style Guidelines	**Level**
[S.D.1]	**IP interface design issue**	
[S.D.1.1]	Output signals must be registered	**M2**
[S.D.1.2]	Registers in an IP's interface should be clocked in a single edge of the same clock.	**R**
[S.D.1.3]	The interface of an IP should connect to a bus.	**R**
[S.D.1.4]	The detail information of the input to clock and clock to output signal delay must be planned and provided.	**M1**
[S.D.2]	**IP should be synchronous.**	**M2**
[S.D.3]	**Clock strategy must be documented.**	**M1**
[S.D.4]	**Avoid handcrafted clock gating.**	**M2**
[S.D.5]	**Rules of reset in the IP**	
[S.D.5.1]	IP must be able to reset.	**M2**
[S.D.5.2]	IP reset strategy must be documented.	**M1**
[S.D.6]	**Avoid any technology-dependent element.**	**M2**
[S.D.7]	**Additional functional behavior models should be developed and provided.**	**R**

Design Style Guidelines

[S.D.1.1]	**Output signals must be registered**	**M2**

- Input delays and output drive strengths are more predictable when registering the output signals from an IP.





Clock
Signal a
Signal b
Available timing budget for IP B
Delay of logic between signal a and signal b

Design Style Guidelines

[S.D.1.2]	**Registers in an IP's interface should be clocked in a single edge of the same clock.**	**R**

- It will reduce the design complexity of the other circuitry that interacts with this IP.

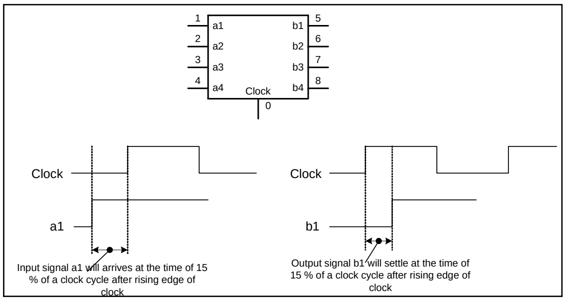
[S.D.1.3]	**The interface of an IP should connect to a bus.**	**R**

- IP should be connected to the rest of system components through MUX-based bus.
- When point-to-point scheme (or connection) is unavoidable, IP creators should use an industry standard

Design Style Guidelines

[S.D.1.4]	**The detail information of the input to clock and clock to output signal delay must be planned and provided.**	**M1**

- The default signal delay from the input signal to clock and clock to output signal, must be hand over to IP integrators in document and script template forms.
- For a soft IP, this information can be represented in terms of the percentage of the input clock period of this IP

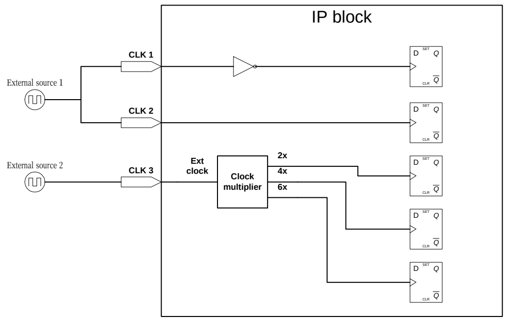

1
2
3
4
5
6
7
8
a1
a2
a3
a4
b1
b2
b3
b4
Clock
0
Clock
a1
Input signal a1 will arrives at the time of 15 % of a clock cycle after rising edge of clock
Clock
b1
Output signal b1 will settle at the time of 15 % of a clock cycle after rising edge of clock

Design Style Guidelines

[S.D.2]	**IP should be synchronous.**	**M2**

- Although asynchronous designs have advantages in power consumption and performance, they could cause some problems, such as metastability.

[S.D.3]	**Clock strategy must be documented.**	**M1**

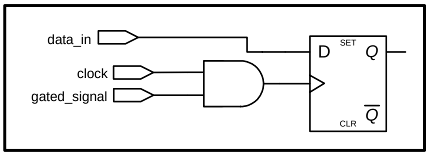


Design Style Guidelines

[S.D.4]	Avoid handcrafted clock gating.	M2

- Designers must avoid inserting any handcrafted clock-gating cells to gate their clock signals.
- If designers need to use clock gating in their IP, they should let power optimization tools to take care this job for them.

```
assign gate_clock= clock & gated_signal
always @(posedge gate_clock)
{
  assign data_reg_1 <=data_in;
}
```



Design Style Guidelines

[S.D.5.1]	IP must be able to reset.	M2

- IP must be designed with reset mechanism to allow IP go back to their initial statuses.
- Control units in IP must be able to reset to ensure the correctness (controllability) of these IP.
- For the reason of saving area, data path can be an exception of this rule.

[S.D.5.2]	IP reset strategy must be documented.	M1

- Necessary information of IP reset strategies should be delivered is specified in soft IP deliverables guidelines, rule **[S.DOC.3].**

Design Style Guidelines

[S.D.6]	**Avoid any technology-dependent element.**	**M2**

- Soft IP should not include any technology-dependent element, such as hard IP or elements that related to any specific technology.
- If there are any technology dependent macros (or IP) in a soft IP, it will have problems when porting this soft IP into different technologies and make this IP difficult to reuse.

[S.D.7]	**Additional functional behavior models should be developed and provided.**	**R**

- IP providers are encouraged to develop and provide more functional models in higher abstraction levels. For example, bus functional models.
- These models can enable IP users to perform more functional verification at higher abstraction levels in a very efficient way.

Outline

- **Motivation**
- **IP Generic Design Flow**
- **Severity Level Definition**
- **The Structure of Design Guidelines**
  - Soft IP
    - Verilog HDL Coding Guidelines
    - Design Style Guidelines
    - *Synthesis Script Design Guidelines*
  - Hard IP
    - Design Style Guidelines
    - Physical Design Consideration
- **Lint Tools Certification**
- **Case Study**
- **Conclusion**

Synthesis Scripts Rule Lists

	Design Style	Level
[S.SCR.1]	Timing budget must be included in synthesis script template.	M2
[S.SCR.2]	Clock and asynchronous reset networks must be set as ideal nets.	M2
[S.SCR.3]	Scripts in synthesis script template must include headers.	M2
[S.SCR.4]	Synthesis scripts must be written with comments.	R
[S.SCR.5]	Keep line length within 72 characters in synthesis script template.	R
[S.SCR.6]	Hard-coded numeric values, filenames and path names are not allowed.	M2
[S.SCR.7]	Keep common constants, parameters, commands and tasks in one or a small number of files, whereas unique ones in individual files.	R

Synthesis Scripts

[S.SCR.1]	Timing budget must be included in synthesis script template.	M2

- IP creators must develop timing budget during soft IP creating (authoring) process.
- IP creators should create a synthesis script template with adjustable timing budget in terms of the percentage of the clock period.

[S.SCR.2]	Clock and asynchronous reset networks must be set as ideal nets.	M2

- Clock and asynchronous reset networks are typically optimized by the back-end tools.
- Instead, they are handled in physical design by means of inserting balanced clock trees with very low skew.

Synthesis Scripts

[S.SCR.3]	**Scripts in synthesis script template must include headers.**	**M2**

```
// +SynHDR+ --------------------------------------------------
// File name: Uart_timing.scr
// Author: Bryan Wang
// Release Version: V.1.0
// Release Date: 2003/06/10/
// Version Description: First version of this script:
// Purpose: Timing budget of UART
// Parameter:
//        Paddr: Address bus width. The default value is 3
//        Prdata: Read data bus width. The default value is 32
// -SynHDR ---------------------------------------------------
```

Synthesis Scripts

[S.SCR.4]	**Synthesis scripts must be written with comments.**	**R**

- Using comments will improve readability of scripts in synthesis script template.

[S.SCR.5]	**Keep line length within 72 characters in synthesis script template.**	**R**

- Line length in synthesis scripts is suggested to be 72 characters or less. Some character space is reserved for the line number.
- Because some output devices like standard terminals or printers have the maximum character limit per line (e.g., 80 characters).

Synthesis Scripts

[S.SCR.6]	**Hard-coded numeric values, filenames and path names are not allowed.**	**M2**

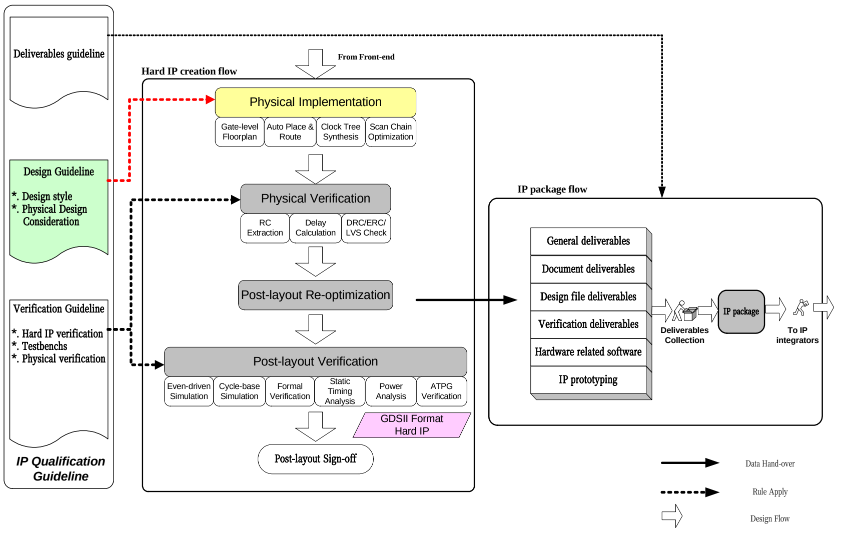
- Any hard-coded numbers, data values, filenames and path names buried in the body of the script are not allowed.

[S.SCR.7]	**Keep common constants, parameters, commands and tasks in one or a small number of files, whereas unique ones in individual files.**	**R**

- Common-used scripts, such as settings of the library and search paths are located in a single setup file.
- Custom-tuned scripts that perform unique tasks like timing constraints are packaged elsewhere.

Outline

Hard IP Generic Design Flow



Hard IP Design Guidelines Rule Lists

	Design Style Guidelines	Level
[H.D.1]	**IP interface design issue**	**M2**
[H.D.1.1]	Output signals must be registered	**R**
[H.D.1.2]	Registers in an IP's interface should be clocked in a single edge of the same clock.	**M1**
[H.D.1.3]	The interface of an IP should connect to a bus.	**M2**
[H.D.1.4]	The detail information of the input to clock and clock to output signal delay must be planned and provided.	**M1**
[H.D.2]	**IPs should be synchronous.**	**R**
[H.D.3]	**Clock strategy must be full documented.**	**M2**
[H.D.4]	**Rules of reset in the IP**	**M2**
[H.D.4.1]	IPs must be able to reset.	**M2**
[H.D.4.2]	IP reset strategies must be documented.	**R**
[H.D.5]	**Additional functional behavior models should be developed and provided.**	**M2**
[H.D.6]	**Synthesis models must be developed and provided.**	**M1**
[H.D.7]	**Power consumption must be analyzed and documented.**	**M2**
[H.D.8]	**Manufacturing test strategy and information must be documented.**	**M1**
	Physical Design Donsideration	**Level**
[H.PD.1]	**Aspect ratio of a hard IP should close to 1:1.**	**R**
[H.PD.2]	**Through-cell blockage map and over-cell blockage map must be included.**	**M1**
[H.PD.3]	**Reserve metal layers for over-IP routing.**	**R**
[H.PD.4]	**Pins of IPs should preserve metal layers with different directions.**	**R**

Hard IP Design Guidelines

[H.D.6]	**Synthesis models must be developed and provided.**	**M1**

- The synthesis models must be developed and provided to enable the synthesis process for the chip that utilizes the given IP.

[H.D.7]	**Power consumption must be analyzed and documented.**	**M2**

- Power consumption including operation condition, operation frequency, and measuring method.

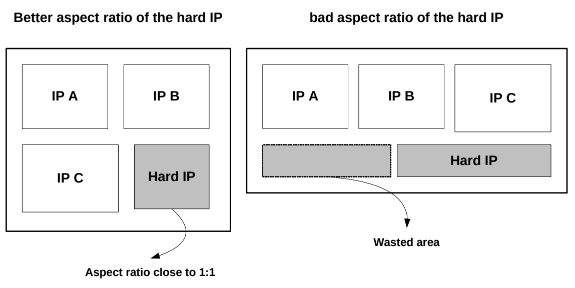
[H.D.8]	**Manufacturing test strategy and information must be fully documented.**	**M1**

- DFT techniques and the related test strategy such as test mode switch, the way to feed the test vectors and observe the results, and the fault coverage must be documented.

Outline

Physical Design Consideration

[H.PD.1]	**Aspect ratio of a hard IP should close to 1:1.**	**R**


Better aspect ratio of the hard IP
bad aspect ratio of the hard IP
IP A
IP B
IP C
Hard IP
IP A
IP B
IP C
Hard IP
Wasted area
Aspect ratio close to 1:1

[H.PD.2]	**Through-cell blockage map and over-cell blockage map must be included.**	**M1**

- Hard IP must include these maps to identify areas where through-cell routing (or over-cell routing) is possible and will not cause timing problems. With this information, IP integrators know which parts of this IP are routable.

Physical Design Consideration

[H.PD.3]	**Reserve metal layers for over-IP routing.**	**R**

- Leave several metal layers for over-IP routing in complex design and big hard macros.
- With reserved routing layers, IP integrators have more flexibility in whole chip routing process, especially when the hard IP occupies large area.

[H.PD.4]	**Pins of IPs should preserve metal layers with different directions.**	**R**

- Every hard IP pin needs to have at least one horizontal layer and one vertical layer.
- The purpose of this rule is to provide more flexibility in whole chip routing process.

Outline

- **Motivation**
- **IP Generic Design Flow**
- **Severity Level Definition**
- **The Structure of Design Guidelines**
  - Soft IP
    - Verilog HDL Coding Guidelines
    - Design Style Guidelines
    - Synthesis Script Design Guidelines
  - Hard IP
    - Design Style Guidelines
    - Physical Design Consideration
- ***Lint Tools Certification***
- **Case Study**
- **Conclusion**

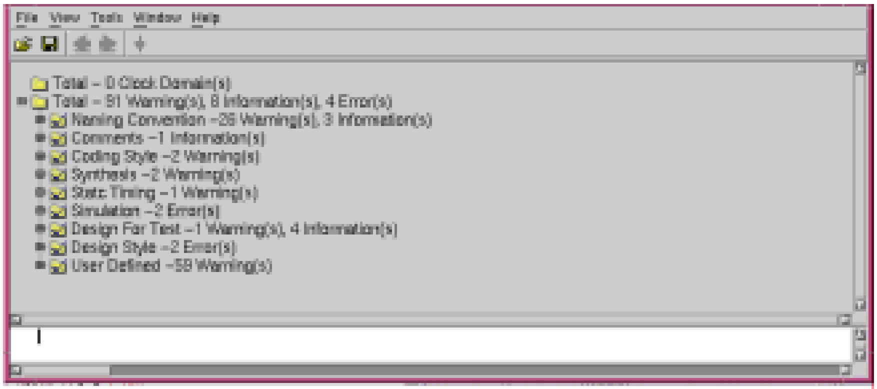
Lint Tool Certification

- **What kind of rules can be checked automatically?**
  - Only rules in Design Guidelines
  - Rules not in conceptual
- **Lint tool limitation**

Checkable	Un-checkable	Total
106	19	125
Target checkable Rate : 84%		

- **Lint tool certification bulletin**

Lint Tool Checking Result



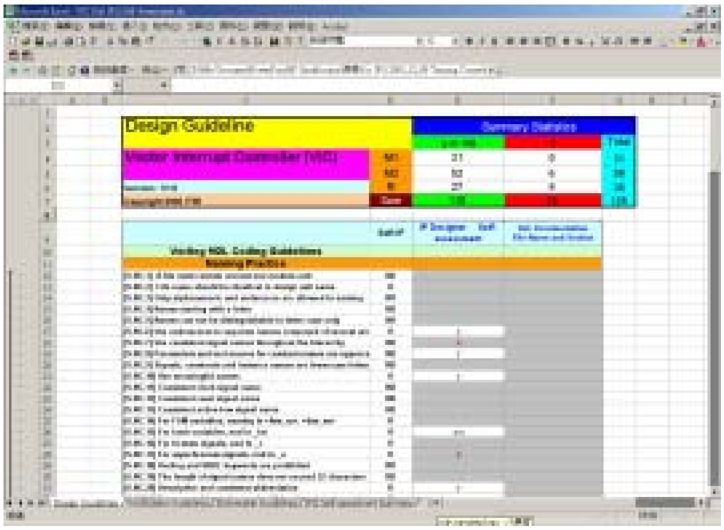
```
Total – 0 Clock Domain(s)
Total – 91 Warning(s), 8 Information(s), 4 Error(s)
  Naming Convention –26 Warning(s), 3 Information(s)
  Comments –1 Information(s)
  Coding Style –2 Warning(s)
  Synthesis –2 Warning(s)
  Static Timing –1 Warning(s)
  Simulation –2 Error(s)
  Design For Test –1 Warning(s), 4 Information(s)
  Design Style –2 Error(s)
  User Defined –59 Warning(s)
```

Outline

- **Motivation**
- **IP Generic Design Flow**
- **Severity Level Definition**
- **The Structure of Design Guidelines**
  - Soft IP
    - Verilog HDL Coding Guidelines
    - Design Style Guidelines
    - Synthesis Script Design Guidelines
  - Hard IP
    - Design Style Guidelines
    - Physical Design Consideration
- **Lint Tools Certification**
- ***Case Study***
- **Conclusion**

Industrial Technology Research Institute
SoC Technology Center

Self-Assessment Spreadsheet


Design Guideline
Summary Statistics
Vector Interrupt Controller (VIC)
M1
M2
R
Total
Verilog HDL Coding Guidelines
Naming Practice

Industrial Technology Research Institute
SoC Technology Center

Violation and Explanation Reports

```
// Lint report ////////////////////////////////////////////////////////
Total - 5 Information(s), 5 Error(s), 153 Warning(s)
  Naming Convention -4 Information(s)
    NC.17 - asynchronous signal name prefix or suffix - 4 Information(s)
      AHB_INTERFACE.v(619) : Information : SUFFIX "_a" should be added to asynchronous signal "hreset_n"
      INTERRUPT_PRIORITY.v(249) : Information : SUFFIX "_a" should be added to asynchronous signal "hreset_n"
      INTERRUPT_REQUEST.v(138) : Information : SUFFIX "_a" should be added to asynchronous signal "hreset_n"
      VECTORED_INTERRUPT.v(103) : Information : SUFFIX "_a" should be added to asynchronous signal "hreset_n"
  Design For Test -1 Information(s)
    DFT.11-2 - constant connected to instance - 1 Information(s)
      VIC.v(525) : Information : port should not be connected to a constant "1'b0"

  Design Style -5 Error(s)
    D.1.1 - outputs leaving partition without been driven by register - 5 Error(s)
      VIC.v(24) : Error : the output "hready_out" is not registered
      VIC.v(24) : Error : the output "hresp" is not registered
      VIC.v(25) : Error : the output "irq_out_n" is not registered
      VIC.v(25) : Error : the output "fiq_out_n" is not registered
      VIC.v(26) : Error : the output "vect_addr_out" is not registered

// Explanation ////////////////////////////////////////////////////////
    The signals "hready_out" and "hresp" must be mux output and
    their sources are registers. The signals "irq_out_n",
    "fiq_out_n", and "vect_addr" are asynchronous signals, they
    don't need to be register output.
///////////////////////////////////////////////////////////////////////
  User Defined -153 Warning(s)
    30003 - Use consistant signal name throughout hierarchy - 133 Warning(s)
```

Violation and Explanation Reports

```
[S.STA.2] Using synchronous design practices
// Explanation ////////////////////////////////////////////////////////////
   VIC is designed to control the interrupt signals which are
   asynchronous. In order to reduced the interrupt latency, VIC uses
   some combinational circuit to manipulate the interrupt signals. These
   combinational circuit must be set as false path under STA. The
   detailed timing exceptions are listed in the VIC data book sec. 9.
/////////////////////////////////////////////////////////////////////////

[S.STA.4] Avoid multicycle path and false path
// Explanation ////////////////////////////////////////////////////////////
   VIC is designed to control the interrupt signals which are
   asynchronous. In order to reduced the interrupt latency, VIC uses
   some combinational circuit to manipulate the interrupt signals. These
   combinational circuit must be set as false path under STA. The
   detailed timing exceptions are listed in the VIC data book sec. 9.
/////////////////////////////////////////////////////////////////////////

[S.D.2] IP should be synchronous.
// Explanation ////////////////////////////////////////////////////////////
   VIC is designed to control the interrupt signals which are
   asynchronous. In order to reduced the interrupt latency, VIC uses
   some combinational circuit to manipulate the interrupt signals. These
   combinational circuit must be set as false path under STA. The
   detailed timing exceptions are listed in the VIC data book sec. 9.
/////////////////////////////////////////////////////////////////////////
```

Outline

- **Motivation**
- **IP Generic Design Flow**
- **Severity Level Definition**
- **The Structure of Design Guidelines**
  - Soft IP
    - Verilog HDL Coding Guidelines
    - Design Style Guidelines
    - Synthesis Script Design Guidelines
  - Hard IP
    - Design Style Guidelines
    - Physical Design Consideration
- **Lint Tools Certification**
- **Case Study**
- ***Conclusion***

Conclusions

- A set of best practice guidelines for creating reusable IP
- Provides the minimum requirements for use in an SoC design methodology
- Complementary to WW standard
- Useful to design high-quality IP and ease IP integration