ASIC Design Guidelines

V1.0

Table of Content
1 Introduction ............................................................................................................................... 1
2 In the beginning… .................................................................................................................... 1
3 Design practice ......................................................................................................................... 2
4 Verilog RTL Coding style .......................................................................................................... 5
5 Template for Verilog RTL module ............................................................................................ 7

1 Introduction
This document outlines a design guideline that can help designers to plan and to execute a successful ASIC project. It gives an overview of design practices that lend itself to a well-structured and synthesizable RTL code. This overview does not replace, by any stretch, any good references on EDA and HDL, but rather it is a compilation of information gathered from previous design experience and should be used as a checklist.

2 In the beginning…
In real life, it is often hard to allow sufficient time to plan, specify, and analyze the requirement for an ASIC development. Generally, the project planner or the lead designer has to make some judgment call and decide how to compress a project schedule without significantly risking the success. Important architectural and definition works may need to be reprioritized for variety of reasons (availability of resource, aggressive target date). The list below shows tasks that should be done during the project-planning phase to minimize risk and hopefully can compress the schedule through parallel development effort.

1. A design specification, which defines the functions and the characteristics of an ASIC/SOC, serves as a contract between all interested parties, including customer, marketing, engineering, and production team. In addition to details on design related functionality, the specification should include the production related criteria, such as are fault coverage (or DPS), test time, tester type and capacity, packaging. Teams with diverse discipline can then (almost) independently execute their portion of work.

2. Partition the design using top-down strategy and take into account any useable IPs. A well-defined block interface specification, which include a list of ports and description of the communication protocol, allow parallel development with less miscommunication. A well-partitioned chip with good signal flow between the major blocks lends itself to faster turn-around in the physical design.

3. Define the design methodology early in the project and list all the necessary point tools, which ideally are already “pre-approved” (revision that has been used successfully before). In the course of the project, some point tool may require upgrade or hot-fixes. Generally, changing the RTL code to conform to the “problem” tool or creating an external band-aid solution, is faster than waiting for the vendor to “fix” the tool.

---

Example: High-density packaging and its tooling can have a long lead-time, or special simulation tool and resources may also be required.
4. Define and distribute a design guideline to each team member to establish the common practice for the project. The next sections elaborate the recommended practice in more detail.

5. Create a complete list of technology files, special IPs, I/O cells and memory cells that are required for the project. Any missing library files or technology files for any IP can have a very long lead-time. It is recommended to control the revision of these files tightly. Instead of linking to common areas, these files should be copied to project directory for better control and reproducibility, but at the expense of the storage space.

6. Create an aggressive but doable project plan. Define each milestone clearly, each with a set of deliverables and its completion criteria. In a typical situation, the completion date is unchangeable; therefore the planner needs to adjust the size of the resources / manpower and / or arrange for additional IP to be developed in parallel by external entity. Good design partitioning and block definition allow the design tasks to be done in parallel successfully, possible in multi site arrangement. Verification of large design typically requires two or three times the effort of the design itself, and should be coordinated between the system, verification and design teams. A comprehensive simulation plan helps in estimating this effort.

3 Design practice

Common design practice is changing with the advancing technology and the EDA tools that support it. The complexity of chip design grows exponentially as ASICs become larger (die size and number of I/O pads), have staggering target performance and allocate stingy power budget. It is nowadays harder to establish a common design practice for such a wide range of challenges. This section attempts to list some of the key design practices.

1. Apply the golden rule “correct by design” to every task in the project and resist the temptation to quickly design modules and rely on simulations to find problems. It really improves the likelihood of achieving “first time right” design. A good simulation plan is an important document, but simulations usually uncover design issues that the designer wants to find, not the ones he/she is not aware of.

2. As the EDA tools become more sophisticated, designers tend to rely on iterative approach in the design process. Predictability and minimizing the number of iterations are important to time critical projects. It can be achieved by providing a good seed into the process and by anticipating issues at the architectural stage. The iteration loop for timing closure starts at RTL design instead of after netlist release point. The designer should have an estimate on the complexity of each main module, and a rough estimate on the module size. This estimate, combined with other physical attributes (die size, packaging, pinout sequence), can be used to get some idea on the physical structure of the major busses. Heavy loaded busses with long wire traces may require pipeline to help meet the timing. Large busses can be split into hierarchies of busses. Finding the right configuration that meets the bandwidth requirements is the challenge for chip architects, since current EDA tools do not provide a lot of support in solving this problem. It becomes an art, instead of science. This usually leads to an over-designed system, but it

---

2 The project leader should have the opportunity to decide if new update is safe and that the benefit outweighs the concern on incompatibility with the tools and the current design state.

3 Tera Systems Inc. developed an optimizer for RTL (TeraFrom), using their TeraGate library, which is derived from foundry’s library.
gives better predictability for the schedule sake. This architectural decision can then be taken into account during the design of the modules and its test benches, minimizing the overall changes to the database.

3. Use a set of command script for every development step to improve team efficiency and to establish reproducibility of the design flow.

4. Clock synthesis constraint must consider PLL jitter, input threshold variation and on-chip variations into the clock uncertainty budget (both setup and hold).

5. Identify various clock domains within the device and the requirement for synchronization for clock domain crossing. Clock domain crossing requires one of the synchronization methods: (1) FIFO’s, (2) simple double FF, (3) clock edge alignment.

6. Synchronizations must either satisfy the Nyquist criteria for all frequency range at the receiving end or must be frequency independent (see the article by Rob Weinstein titled “flankter”).

7. Global synchronous reset is preferred over asynchronous. The synthesis tool can treat synchronous reset just like a normal signal, meeting the required setup and hold time. Global asynchronous reset must be treated as clock and handled via clock tree insertion process to guarantee small skew within the die.

   Synchronous reset
   always @(posedge clk)    Asynchronous reset
   if (~rst_n)             always @(posedge clk or negedge rst_n)
   dff <= 0;
   else if (enable)       else if (enable)
   dff <= input1;

8. Allow provision for soft reset and/or start/stop bits for each main functional module. Your system/software designers will appreciate that during system bring-up.

9. Minimize the number of smaller embedded memories by combining them if possible, or synthesize it into storage elements from the library.

10. Address Design-for-Test early to establish the implications to the design itself (gate count increase, ad-hoc test requirements, maximum production test time, maximum scan length, handling of latches, IDQ, memory BIST, analog BIST, JTAG and boundary scan). Latches must be set into transparent mode during scan. Large counters should be split during test to minimize test time. Production test can cost as much as $0.10 per second.

11. Meeting hold time requirement for large scan chain is difficult and can increase the gate count significantly and adversely impact timing closure. It is more sensible to relax the constraint to the real test environment instead of using the stringent operating condition.

---

4 We assumed that the design partition (hierarchy) of the RTL code, the floorplan and the one used during place and route are the same. In a very big design this may not be the case, due to tool limitation.

5 Clock edge alignment can be implemented only when the frequencies are evenly multiple of each other. The clock tree synthesis guarantees the skew across the chip. Example: large SOC can have several clocks driving the CPUs and theirs peripheries; the periphery clocks are typically divide down from the CPU clock.

6 The pulse width of the input to the “back-back” flip-flop must be longer than the receiving clock period.

7 Embedded memory requires special treatment during physical design and can cause route congestion (blockage).
Depending on the target application, it is often sufficient to meet the hold time requirement under room temperature and narrow supply voltage range.

12. Register all outputs of main modules for better timing control at the top level.

13. Avoid construct that causes conflict with design for test and common design rule. These structures must be eliminated or special test mode must be added to satisfy DFT tools.

- Clock is connected to data input of a flip-flop

- Data is connected to clock input of a flip-flop

- Multiple clocks are combined and connected to clock input of a flip-flop.

- Asynchronous loop.
4 Verilog RTL Coding style

All HDL being used today have rich language elements, especially when support for system modeling is included such as in SystemC and SystemVerilog. A system can be modeled faster and the resulting code is more maintainable thanks to object-oriented concept. On the other hand, the synthesis tools can process only a subset of these constructs. The purposes of recommending a set of coding style are:

- Common style improves readability among codes written by a team, maintainability.
- Synthesizable RTL allows shorter timing closure and produces correct netlist (zero bugs, zero structural problem).
- RTL and netlist are compatible with each point tool specified in the design flow.
- High simulation throughput.

Designers can use a context sensitive editor to help with the RTL coding and debugging. Examples are eclipse, emacs. A template for the RTL code also helps maintaining uniformity in the formatting of header and body areas.

Below is a list of recommendations as an example, and can be modified to one’s own taste. As IP usage becomes more prevalent, these recommendations become less enforceable. Hopefully we have at the very least consistency at the top level and in the internal developed modules.

1. Comments in the header and in the body should describe the major function of the block. Each input and output port declaration should have a brief description.

2. Making module reusable takes some time and effort. At the very least, the RTL code should use parameters or defines to represent constants such as microprocessor register address, bus widths and others. Using parameterized/defined constants improves readability and maintainability.

3. File name and module name conventions:
   - Verilog file: *.v
   - Verilog header include file: *.vh
   - Script files: *.csh or *.sh or *.tcl
   - Makefiles: *.mk

4. Signal name conventions:
   - "_" can be used as syllable separation for clarity.
   - No mixture between lower and upper case.
   - Upper case is reserved for parameters and pin names.
   - Lower case is for others.

5. Signal name and parameters can use predefined prefix or suffix to enhance readability.

<table>
<thead>
<tr>
<th>Signal</th>
<th>Suffix/Prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Active low signal</td>
<td>xxx_n</td>
</tr>
<tr>
<td>Delayed signal</td>
<td>xxx_1d, xxx_2d, xxx_3d</td>
</tr>
<tr>
<td>Module input and output</td>
<td>xxx_in, xxx_out</td>
</tr>
<tr>
<td>Leading edge pulse, single clock</td>
<td>xxx_le</td>
</tr>
<tr>
<td>Trailing edge pulse, single clock</td>
<td>xxx_te</td>
</tr>
<tr>
<td>Bus width parameter</td>
<td>XXX_W</td>
</tr>
<tr>
<td>Array length parameter</td>
<td>XXX_L</td>
</tr>
</tbody>
</table>

6. List allowable language enhancements in the code. For example, Verilog IEEE 1364-2001 contains some enhancements that are not supported by all simulator, synthesis tools, and editors.
7. The RTL code must not declare any time delay (#), except for I/O pad assignments at the top level. The synthesis tool ignores the delay and can cause simulation mismatch between functional and gate-level.

8. Use liberal sensitivity list in *always* block (@*) to represent combinatorial logic and avoid simulation mismatch between RTL and gate level.

9. Use non-blocking assignment for modeling flip-flop and blocking assignment for modeling combinatorial logic and keep them in separate *always* blocks.

   ```verilog
   always @(posedge clk)
   if (~rst_n)
       q <= 0;
   else
       q <= d | e;

   always @*
   d = a & b & c;
   ```

10. Provide default to incomplete (not full) case statement to avoid unintended latch instantiation. The synthesis directive *full_case* is then not required. It is recommended to inspect the synthesis report to uncover any unintentional latch.

11. Modeling priority encoder should contain unique case items.

    ```verilog
    always @* begin
        {out1, out2, out3} = 3'b0;
        casez (in)
            3'b1??: out1 = 1;
            3'b01?: out2 = 1;
            3'b001: out3 = 1;
        endcase // casez(in)
    end
    ```

    The gate structure is parallel by default (even without synthesis directive *parallel_case*)

12. In general, avoid using *full_case* and *parallel_case* synthesis directive to avoid discrepancy between simulation and synthesis tool.
5 Template for Verilog RTL module

```verilog
/*
 * $Id: $
 * File:           $Source: $
 * Revision:       $Revision: $
 * First created: hsuwito, 9/2003
 * Last updated:   $Date: $
 * By:             $Author: $
 * Description:    Some description
 * More description
 *
 * Copyright (c) 2003-2005 Hauw Suwito
 *
 * This source code is free software; you can redistribute it
 * and/or modify it in source code form under the terms of the GNU
 * General Public License as published by the Free Software
 * Foundation; either version 2 of the License, or (at your option)
 * any later version.
 *
 * This program is distributed in the hope that it will be useful,
 * but WITHOUT ANY WARRANTY; without even the implied warranty of
 * MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
 * GNU General Public License for more details.
 *
 * You should have received a copy of the GNU General Public License
 * along with this program; if not, write to the Free Software
 * Foundation, Inc., 59 Temple Place - Suite 330, Boston, MA 02111-1307, USA
 *
 * See the end of the file for revision history
 *
 */
`include "timescale.v"

module my
 {
  //AUTOARG*
  );

`include "my_param.v"

  // Outputs declaration
  output [1:0]  out1;  // output 1
  output    out2;  // output 2

  // Inputs declaration
  input     clk;    // global clock
  input     rst_n;  // active low reset, synchronous
  input [1:0] inp1;  // input 1
  input     inp2;  // input 2

  // Wires and registers declaration
  //
  reg [2:0]    temp;  // temporary 1
  wire        x;    // one wire

  /*AUTOWIRE*/
  /*AUTOREG*/

  // Module main body
  // *
  **
endmodule // my
```
/*
 * Revision history
 * $Log: $
 */
/*
 *  1   2   3   4   5   6   7   8
 *34567890123456789012345678901234567890123456789012345678901234567890
 * local Variables:
 * verilog-library-directories(".", "../rtl", "../rtl", "..")
 * End:
 */