**Digital IC Frontend Design Methodology**

**(ARTOSYN)**

|  |  |  |  |
| --- | --- | --- | --- |
| Version | Writer | Date | Description |
| V0.1 | Sha Shen | 2017/10/9 | Initial |
| V0.2 | Sha Shen | 2017/11/15 | Add chapter 5-7 |
|  |  |  |  |

## Table of contents

[Table of contents 1](#_Toc499804828)

[Chapter 1. Design Flow Overview 2](#_Toc499804829)

[Chapter 2. Design Specification 3](#_Toc499804830)

[Chapter 3. RTL Design guideline 4](#_Toc499804831)

[Chapter 4. Function Verification 5](#_Toc499804832)

[Chapter 5. FPGA Porotype Building 6](#_Toc499804833)

[Chapter 6. ASIC synthesis 7](#_Toc499804834)

[Chapter 7. Power Estimation 8](#_Toc499804835)

## Chapter 1. Design Flow Overview

This document is intended as a guideline for digital IC frontend design. The frontend design flow can be summarized in Fig. 1.1.



Fig.1 Digital frontend design flow

Each design step is described below:

* 1. **Design Specification**

**Output of this stage:**

* A PDF document which describe the overall system function and architecture. A feature list should also be included in this document.

**Tools to be used:**

* Text editor like word for documents
* excel for tables
* visio for diagrams
  1. **Algorithm modeling**

**Output of this stage:**

* Documents describe the algorithms which will be used in the system. Any related materials such as any book, paper or web link should also be added as reference.
* C or matlab code to build an algorithm model for the whole system

**Tools to be used:**

GCC(preferred) or Visual Stuido for C model

Matlab

* 1. **RTL Design**

**Output of this stage:**

Verilog HDL code of the system described in Design Specification

Preliminary FPGA Timing

Preliminary ASIC Timing

No critical warning in DC/Synplify report

Lint/CDC

**Tools to be used:**

Synplify for FPGA timing

DC for ASIC timing

Spyglass for Lint/CDC

* 1. **Function Verification**

**Output of this stage:**

RTL Test case list and the status of each test case

Code coverage generated by VCS

Code coverage generated by Veloce

**Tools to be used:**

VCS

Veloce

* 1. **FPGA Prototyping**

**Output of this stage:**

FPGA test case list and the status of each test case

**Tools to be used:**

Synplify

Vivado for Xilinx FPGA

QuartusII for Altera FPGA

S2C PlayerPro for S2C FPGA board

* 1. **ASIC synthesis and formal verification**

**Output of this stage:**

Netlist file ready for backend design

Pass Formal verification

Timing/area statistics

Lint/CDC

**Tools to be used:**

DC, FM, Conformal, Spyglass

* 1. **Power Estimation**

**Output of this stage:**

Average and peak power of typical application

**Tools to be used:**

VCS, PrimeTimePX

Table 1.1 Overview of design stage, used tools and output

|  |  |  |
| --- | --- | --- |
| Design stage | Tools | Output |
| Design Specification | Word  excel  visio | PDF file describes the system function and architecture |
| Algorithm modeling | Matlab  gcc/VS | Documents describe the algorithms.  C or matlab code |
| RTL design | Synplify  DC  Spyglass | Verilog HDL code |
| Function verification | VCS  Veloce | RTL Test case list and the status of each test case  Code coverage generated by VCS  Code coverage generated by Veloce |
| FPGA prototype | Synplify  Vivado  QuartusII  PlayerPro | FPGA test case list and the status of each test case |
| ASIC synthesis | DC  FM  Conformal Spyglass | Netlist file ready for backend design  Pass Formal verification  Timing/area statistics  Lint/CDC clear |
| Power  estimation | VCS  PrimeTimePX | Average and peak power of typical application |

* 1. **others**

## Chapter 2. Design Specification

The design specification should include descriptions of both hardware and software. The outcome of this stage will be summarized in two documents: A hardware design specification intended for digital IC design team and a software programming guide intended for software team and verification team.

A typical hardware design specification consists of the following chapters:

1. Overview. The basic functionality of the system is descripted here. A typical application scenario should also be presented in a diagram. The internal architecture of the hardware design is not necessary here and the related contents can be included in Chapter 3.
2. Feature list. The detailed features should be listed in this chapter which defines the exact system functionality.
3. Input/output ports description.
4. Hardware architecture. A detailed internal architecture is described by a diagram here. The diagram of system controller or FSM (Finite State Machine) should also be listed in this chapter. Key performances listed below should also be specified in this chapter
   1. ASIC working frequency (signoff corner, signoff frequency and typical working frequency)
   2. FPGA working frequency
   3. Estimated DDR bandwidth
   4. Estimated ASIC gate count or FPGA resource
   5. Othters

The whole system should be partitioned into multiple sub-blocks. The silicon area of each sub-block should also be estimated before RTL design. The FSM of each subblock should also be presented if necessary.

1. Register description
2. Interrupt description
3. Power estimation. PrimetimePX should be used along with the simulation waveform in order for more accurate power estimation.

## Chapter 3. RTL Design guideline

2.1 RTL style guideline

Some general guidelines of RTL style are listed below:

1. Avoid using timescale in synthesizable RTL. Use the timescale directive only in testbench files.
2. Setting the timescale precision with caution. Higher time precision will lead to significant lower simulation speed.
3. Avoid using delay in an assign statement of synthesizable RTL. Delay statement is not synthesizable and prone to cause simulation/synthesis mismatch.
4. Avoid using synthesis directives in Synplify or DC. Synthesis directives such as full case, parallel case and so on are dangerous for they are also prone to cause simulation/synthesis mismatch.
5. Implement the chip select function of a synthesizable SRAM model, even for FPGA SRAM models. All ASIC SRAM will include a chip select port. Avoid function mismatch between FPGA SRAM model and ASIC SRAM model.
6. The port names of SRAM for FPGA should be the same as those of an ASIC SRAM model
7. The power related SRAM ports such as LS/DS/SD(light-sleep, deep-sleep and shutdown) should be tied to fix value in test mode.
8. The power related SRAM ports such as LS/DS/SD(light-sleep, deep-sleep and shutdown) should be used together with chip select port to enable or disable SRAM access.
9. Check the width of a sum or product in order to avoid bit truncation

A Bug example is shown in Fig. 2.1. Expanding the width of register “pack\_cont” to 7bit will fix the bug. Please note that the statement in Fig. 2.1 won’t trigger any lint warning even if checked by DC/Lint/Spyglass.

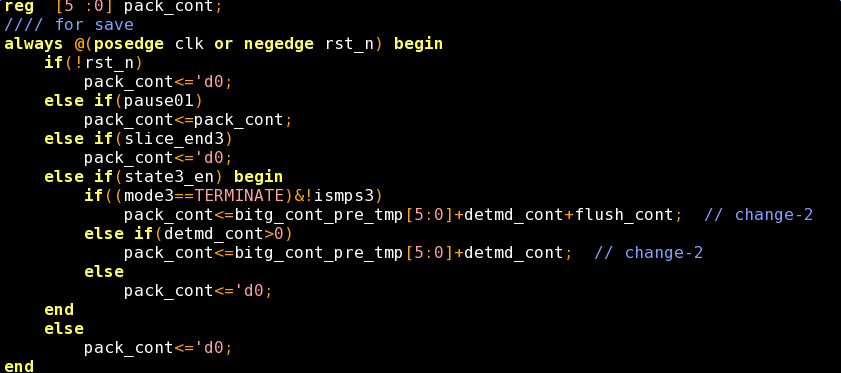


Fig. 2.1 Register overflow bug example

1. Others

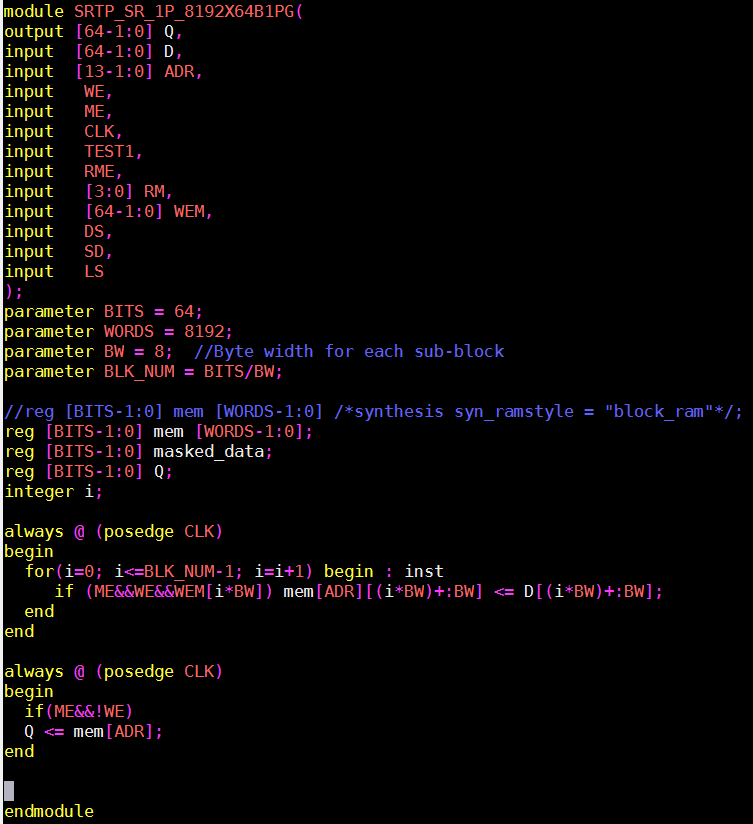


Fig. 2.2 single port SRAM port example

Various tools should be used to check the RTL style.

1. Synplify/DC. Immediately after the RTL code is done and before any simulation, Synplify and also DC should be used to check the sanity of the RTL code. At the early stage, the SDC file may be unavailable. The timing information can still be obtained without any SDC file if Synplify is used. A quick estimation of ASIC timing can be performed. As a general rule, the design in ASIC form will normally run by 5-10 times faster than its FPGA counterpart.
2. VCS. VCS also has the ability of lint check by its command line option.
3. DC. After the function verification is done by using VCS, the accurate report of ASIC area and timing can be obtain by DC. Any warning should be cleaned by modifying RTL or suppressed by DC setting. Any inferred latches should also be removed.
4. Spyglass. CDC (clock domain crossings) check should be done at the final stage of RTL design. Lint check can also be done by spyglass.

2.2 SRAM modeling for both ASIC and FPGA

Please note that all SRAM model MUST include the chip select signal which is a common signal presented in almost all ASIC SRAM models. Otherwise the FPGA model will have different behavior with that of an ASIC SRAM model. This difference will cause fatal bugs in ASIC chip which cannot be verified on FPGA.

The FPGA models can be generated by a Python script. The Python script reads in a csv file which contains the width/height and other SRAM model settings. Then it generates the Verilog SRAM models according to the templates.

SRAM models with scalable write mask (bit enable) input ports are show below:

1. single-port SRAM with write mask inputs

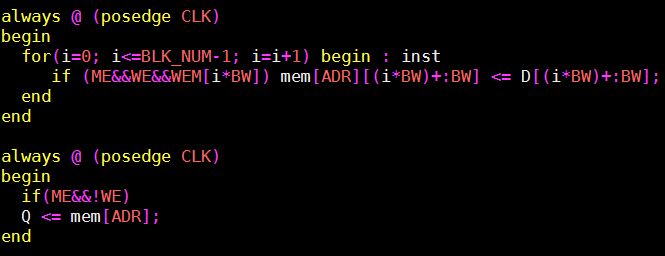


Fig. 2.3 Implement SRAM write mask function in one single statement

BW is the data bit width for a single write strobe signal. WE is read/write control signal and WEM is the write strobe signal. ME is used as the

1. dual-port SRAM with write mask inputs

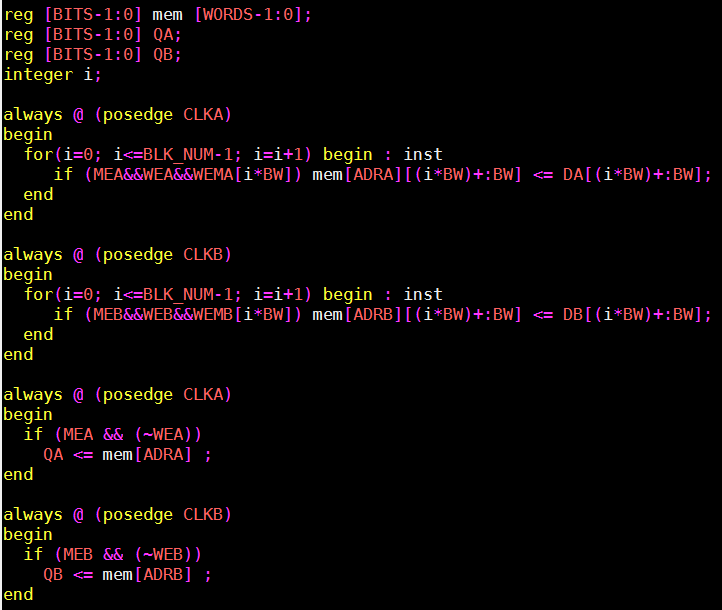


Fig. 2.3 Code example for dual-port SRAM with write mask inputs

2.3

## Chapter 4. Function Verification

3.1 verification plan

A detailed verification plan should be documented to cover each feature on the design specification. Random regression test should also be applied wherever possible. The test case list should be checked against the feature list in the design specification.

3.2 code coverage

Both VCS and Veloce can be used to calculate the code coverage. 100% code coverage must be met before the sub-module RTL is delivered for top level system integration.

3.3 function coverage

A complete regression test should be carried out according to the verification list.

## Chapter 5. FPGA Porotype Building

4.1 Bottom-up compile/synthesis/PR flow

It is a quicker way to synthesis a design over one million gate count by bottom-up method. Synplify should be used as a first choice.

4.2 Synplify for compile/synthesis

Check the clock gating report and make sure all gated clocks or clock mux are removed.

Synplify is prone to crash in case of some ill-written code, especially for SRAM.

4.3 Vivado for P/R (Xilinx FPGA)

4.4 Quartus for Altera FPGA

## Chapter 6. ASIC synthesis

6.1 asic

## Chapter 7. Power Estimation