VLSI 设计

Norm 指令的低功耗全定制设计

Triloo

国防科技大学计算机学院

July 16, 2017

Contents

报告说明

本次实验主要完成 tms320 DSP 的 NORM 指令的全定制设计，设计流程为：电路图设计、功能验证、时序分析与电路优化、版图设计与验证等步骤，主要包括 31 位反向器、2 路 31 位选择器、以及前导零检测电路等子模块。其中，前导零模块的设计参考"Low-power leading zero counting and anticipation"[1] 一文, 即通过逻辑优化，尽可能的降低需要用到的器件的数量来降低电路的功耗，具体内容参考第一章节中设计思路一节。最后，基于以上子模块，完成 NORM 指令的全定制设计，并通过 DRC、LVS 检查。

Chapter 1

功能与结构设计

本章节将给出 Norm 命令的基本功能以及全定制实现的总体结构图等。

1.1 指令功能

首先 Norm 指令具有如下形式：

NORM (.unit)　　*src2, dst*

其过程涉及两个操作数，*src2* 为输入操作数，*dst* 为目的操作数，位宽均为 32 位。需要实现的功能如下：

> The number of bits of the first nonredundant sign bit from the MSB of the src2 operand is placed in dst.

该指令的功能即根据符号位的数值统计前导零或一的数量，若符号位为 1，则统计符号位后前导一的位数，否则统计符号位后前导零的位数。例子如图1.1所示：

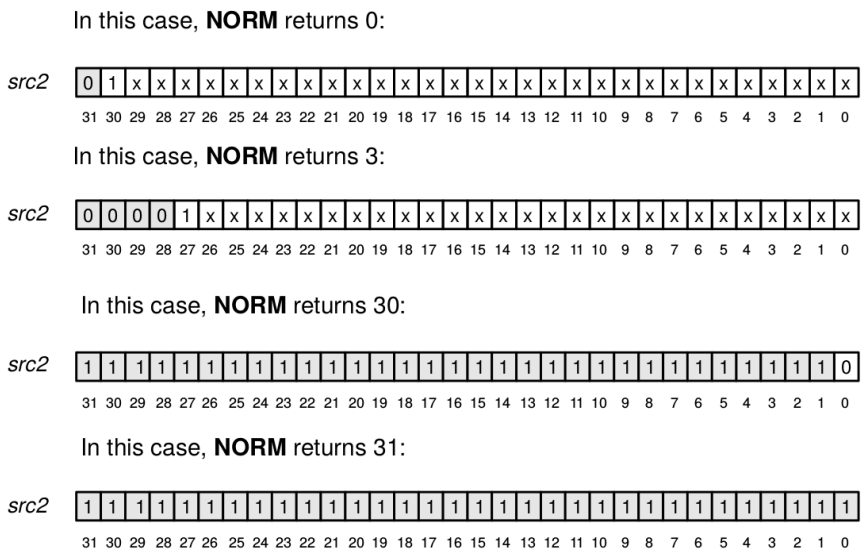


Figure 1.1: NORM 指令功能示例

并要求该指令需要在一个时钟周期以内完成运算。存在多种电路可以完成该指令的功能，但本次实验仅选择基于" 前导零计数" 的方式对其进行实现。

1.2 设计思路及总体结构图

本小节将给出 NORM 指令的设计思路，即各个子模块的原理，然后给出 NORM 指令的总体结构图。

1.2.1 设计思路

通过对上述指令的说明进行分析可以发现，该指令的功能为计算符号位后连续 1 或 0 的个数，连续零的个数可以通过前导零计算的模块进行，而前导 1 的个数可以通过先将输入操作数进行按位取反后再进行前导零的统计。

前导零计算 (LZC) 模块

通过以上讨论，可以明确前导零计数模块 (LZC, Leading Zero Count Unit) 的设计对 NORM 指令的实现至关重要。目前同样存在多种计算前导零的实现方法，文章 [1] 中给出了三种已有的计算方法，下面对其进行简单说明。在本章节的最后一部分中，给出该篇文章实现的低功耗改进方法，这也是本次实验采用的方案。

- **Encoder-Based LZC Unit**
  第一种方法是基于两步编码过程的实现电路。首先检测 Leading Digit(即前导零统计过程中的第一位不为零的位，前导 1 的统计过程中类似。)，并将检测结果编码成 One-Hot 编码形式，如输入为 00110100, 则第一步的编码结果输入为 00100000。为了得到 One-Hot 码，引入中间变量 S，与 One-Hot 码不同的是，该中间变量在 leading digit 位之后的所有位都为 1，例如同样输入 00110100 时，中间变量 S 为 00111111。设 S 的第 i 位表示为 $S_i, 0 \leq i \leq n-1$，定义如下：

$$S_i = A_{n-1} + A_{n-1} + \cdots + A_{i+1} + A_i \tag{1.1}$$

  其中 $+$ 表示逻辑或运算。从公式1.1可以看出 S_i 的含义为最高位与 i_th 位之间是否存在为 1 的位。在得到中间变量 S 后，通过将 S 的连续两位的数值来得到 One-Hot 码 (用 L 表示)：

$$L_i = \overline{S}_{i+1} \cdot S_i = \overline{S}_{i+1} \cdot A_i \quad (0 \leq i \leq n-2) \tag{1.2}$$

  其中 $L_{n-1} = S_{n-1}$。得到 One-Hot 码 L 的电路被称为优先编码器 (Priority Encoder)。

  在本方法中用到的第二个编码器是将得到的 One-Hot 码转换为类似" 8421" 码的有权码。例如，对于输入为 8 位的数值的转换过程可以表示为：

$$\begin{aligned} Z_2 &= L_3 + L_2 + L_1 + L_0 \\ Z_1 &= L_5 + L_4 + L_1 + L_0 \\ Z_0 &= L_6 + L_4 + L_2 + L_0 \end{aligned} \tag{1.3}$$

  经过上述两次编码后，即完成前导零统计的过程。

- **Algorithm-Based LZC Unit**
  已有的第二种计算前导零的电路被称为基于算法的统计。在该方法中，首先将输入 n 位数值分成 $n/2$ 个连续两位构成的 Groups. 对于每一 Group 实行前导零计算。接下来，相邻的两个 Group 的计算结果被组合，且基于左 Group 的结果对两个结果进行选择。然后经过 $\log_2 n$ 次迭代后即可实现前导零检测的功能。这种方法可以通过模块化的方式进行全定制实现，同时也是目前已知的最快的基于静态 CMOS 的前导零实现方法。其示意图如图1.2所示：

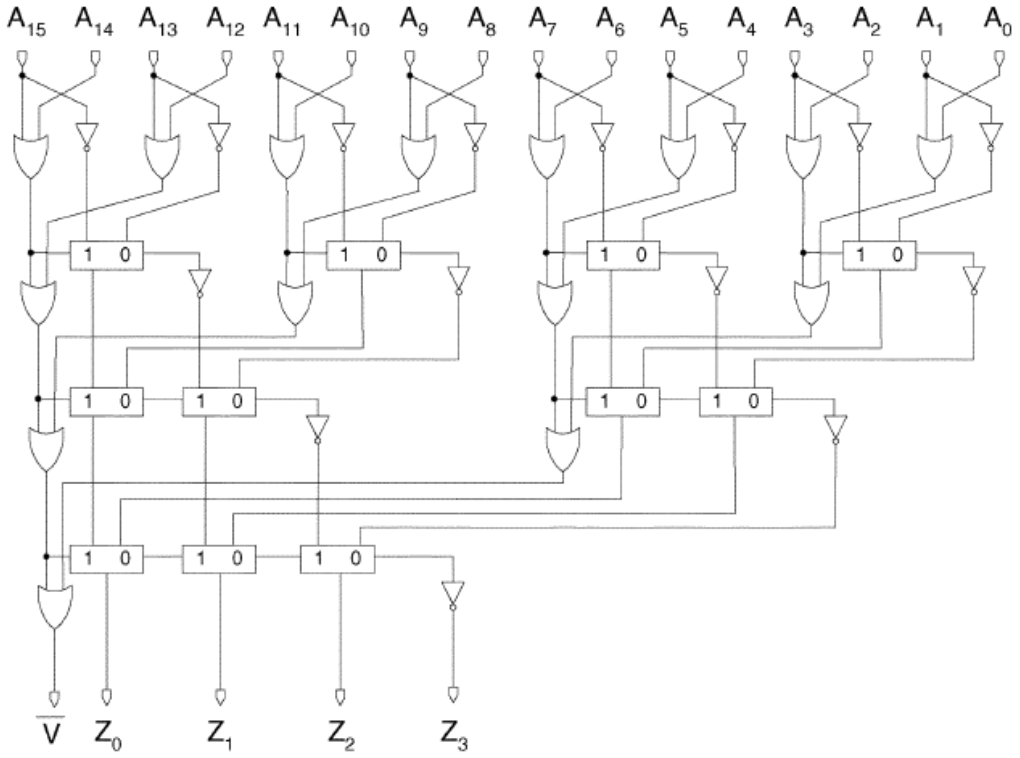


Figure 1.2: Algorithm-based LZC 的 16-bit 实现电路图

- **Recursively-Based LZC Unit**
  这种方法通过迭代的方式计算 LZC 结果 (用 Z 表示) 的第 i 位。下面以 8-bit 的 LZC 结算过程为例进行说明。首先对最高的 4 位进行 $A_7A_6A_5A_4$ 检测，若此四位全为零，则说明输入的数值 A 中至少存在 4 个 0，因此将 Z_2 置位，即得到 $Z_2 = 1$，这意味着我们还需要对 $A_3A_2A_1A_0$ 进行检测并统计。接下来则需要对 A_3, Z_2 以及 A_7, A_6 进行检测，并基于检测结果来决定 Z_1 的具体数值，并根据 Z_2 的数值来决定具体是根据 A_3, Z_2 还是 A_7, A_6 来确定 Z_1 的数值，然后重复上述过程，直至完成 LZC 统计。

低功耗前导零计算模块原理

在这一部分，将给出低功耗前导零计算模块的具体设计思路，即文章 [1] 中的"PROPOSED LZC ALGORITHM" 一节。首先，将公式1.2带入公式1.3中可以得到

如公式1.4所示的结果：

$$\begin{aligned} Z_2 &= \overline{S}_4 \cdot S_3 + \overline{S}_3 \cdot S_3 + \overline{S}_2 \cdot S_1 + \overline{S}_1 \cdot S_0 \\ Z_2 &= \overline{S}_6 \cdot S_5 + \overline{S}_5 \cdot S_4 + \overline{S}_2 \cdot S_1 + \overline{S}_1 \cdot S_0 \\ Z_2 &= \overline{S}_7 \cdot S_6 + \overline{S}_5 \cdot S_4 + \overline{S}_3 \cdot S_2 + \overline{S}_1 \cdot S_0 \end{aligned} \tag{1.4}$$

通过对 S_i 的计算过程分析可以发现，$(S_i, S_j)\ with\ i > j$ 不可能取 $(1,0)$ 的结果，基于这一性质可以对 Z 的计算过程进行简化，最终可以得到如下结果：

$$\begin{aligned} Z_2 &= \overline{S}_4 \\ Z_2 &= \overline{S}_6 \cdot S_4 + \overline{S}_2 \\ Z_2 &= \overline{S}_7 \cdot S_6 + \overline{S}_5 \cdot S_4 + \overline{S}_3 \cdot S_2 + \overline{S}_1 \end{aligned} \tag{1.5}$$

然后利用公式1.2的关系将公式1.5中的变量 S_i 替换为输入 A，即得到公式所示的输入与输出的关系：

$$\begin{aligned} Z_2 &= A_7 + A_6 + A_5 + A_4 \\ Z_2 &= A_7 + A_6 + \overline{A}_5 \cdot \overline{A}_4 \cdot (A_3 + A_2) \\ Z_2 &= A_7 + \overline{A}_6 \cdot A_5 + \overline{A}_6 \cdot \overline{A}_4 \cdot A_3 + \overline{A}_6 \cdot \overline{A}_4 \cdot \overline{A}_2 \cdot A_1 \end{aligned} \tag{1.6}$$

最后，文章 [1] 给出了统一的计算 LZC 结果每一位 Z_i 的公式：

$$\begin{aligned} F(X_{n-1}, X_{n-2}, \ldots, X_1, X_0) &= X_{n-1} + \overline{X}_{n-2} \cdot X_{n-3} + \\ &+ \overline{X}_{n-2} \cdot \overline{X}_{n-4} \cdot \overline{X}_{n-5} + \cdots + \overline{X}_{n-2} \cdot \overline{X}_{n-4} \cdot \overline{X}_{n-6} \cdots \overline{X}_2 \cdot X_1 \end{aligned} \tag{1.7}$$

基于公式1.7的 8-bit LZC 计算过程如图所示：

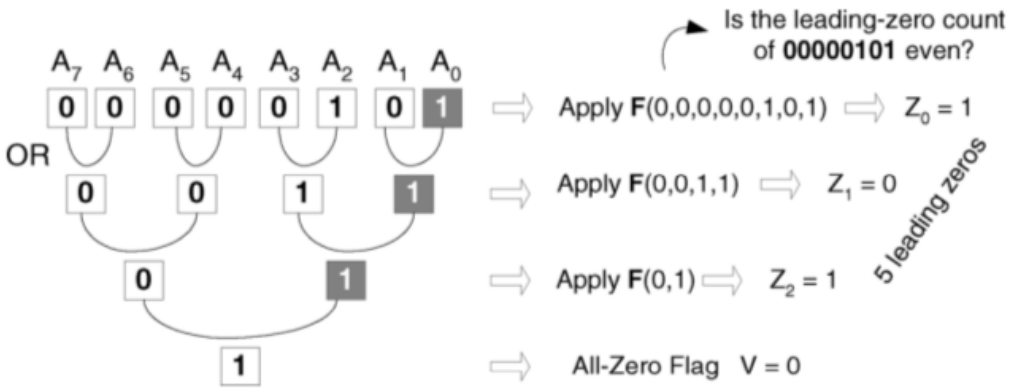
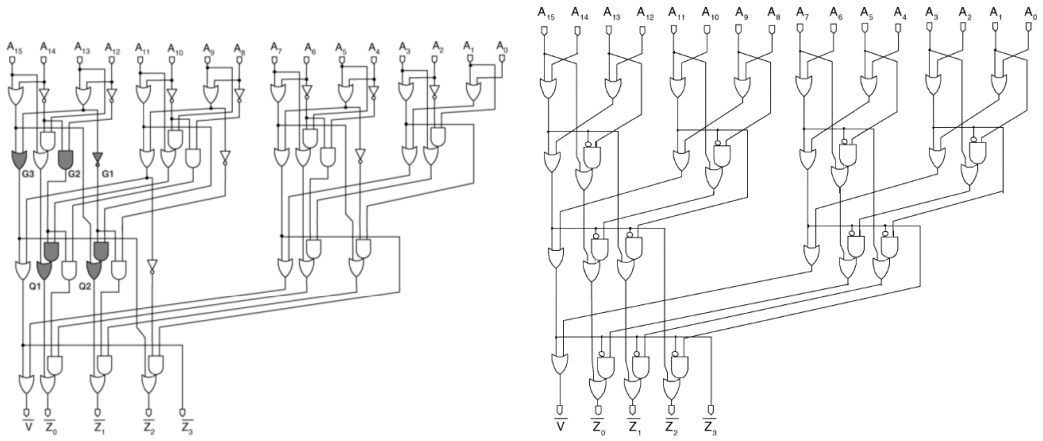


Figure 1.3: 8-bit 输入 LZC 计算过程示意图

低功耗 16-bit LZC 电路图

基于公式1.7的关系，可以得到 16-bit 输入时的 LZC 电路图，如图1.4(a)所示，同时文章中还给出了一种功耗优化的改进电路，如图1.4(b)所示。可以看出，功耗优化后的电路图用到的器件数量更少，连线更加简单，且可以模块化实现，因此易于全定制设计，本次实验将采用图1.4(b)的电路完成 NORM 指令的 LZC 子模块的设计。

关于低功耗 LZC 的设计更详细的信息可参考文章 [1]。



(a) 16-bit LZC 电路 (b) 功耗优化的 16-bit LZC 电路

Figure 1.4: 16-bit 的 LZC 电路及其功耗优化电路

2 路 31 位选择器的设计

由于 NORM 指令除了需要检测前导零，还需要完成前导一的检测，因此，在设计过程中将输入的 32 位数据分成两路，一路将 0 ～ 31 位进行取反，另一路不做处理，并根据符号位是 0 还是 1 进行选择，即若符号位为 1，则选择取反后的数据送入 LZC 电路，若符号位为 0，则将不做处理的数据送入 LZC 电路。2 路 31 位选择器的每一位基于图1.5所示的 1 位选择器实现：

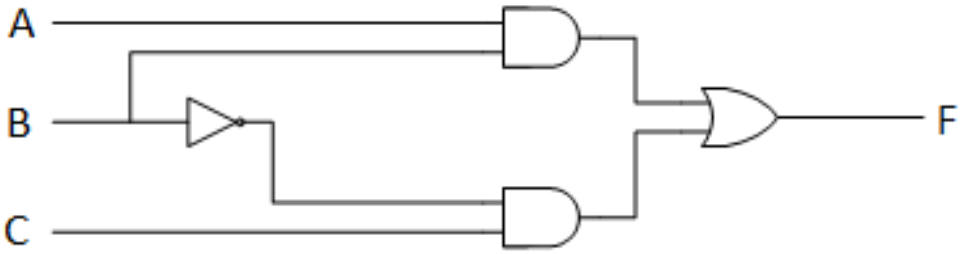


Figure 1.5: 2 路 1 位选择器单元

其中, 若 B 为 0，则 F 为 C 端输入的数据，若 B 为 1，则 F 为 A 端输入的数据。

1.2.2 总体结构图

基于以上讨论，可以得到 NORM 指令的总体结构图，如图1.6所示。

1.3 算法流程

本小节将结合图1.6对 NORM 指令的实现过程进行详细说明。

首先，输入数据的位宽为 32 位，由于 NORM 指令不仅需要计算符号位后连续 0 的个数，而且还要计算符号位后连续 1 的个数，因此为了实现电路复用，，选择将输入数据的 30 ～ 0 位数据分成两路，一路按位取反，一路不做任何处理，然后

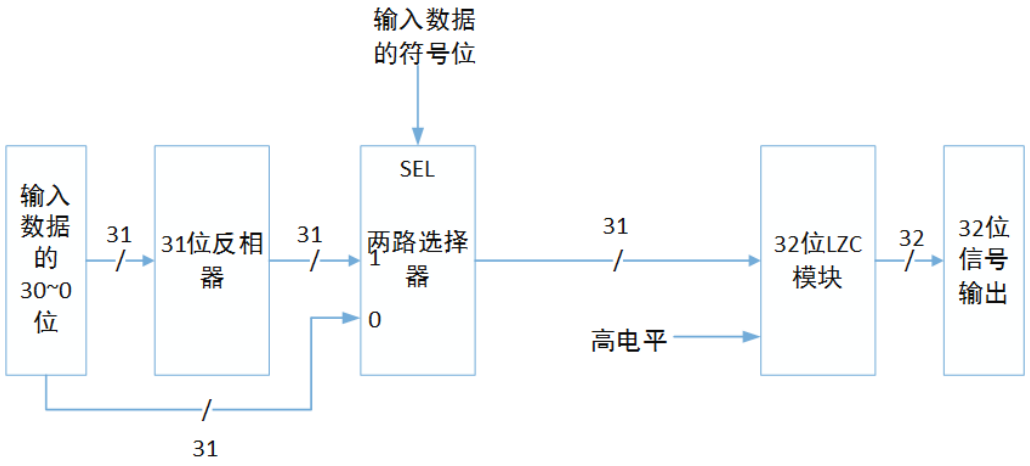


Figure 1.6: NORM 指令结构框图

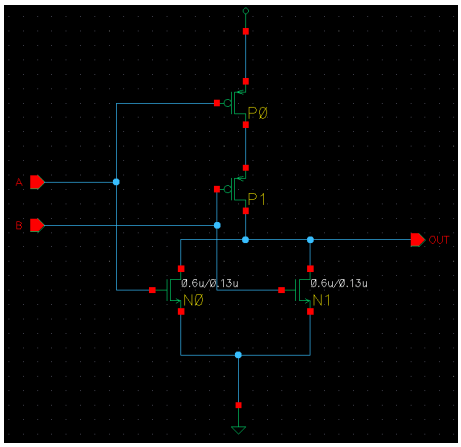
将 31 位反向器的结果与原始数据送入两路 31 位选择器，选择信号为输入数据的符号位，其工作原理为：若符号位为 1 则选择反向器的结果，否则选择原始数据。两路选择器的结果接下来被送往 32 位 LZC 模块，该模块实现前导零计算的功能。由于 LZC 是 32 位而输入数据是 31 位，因此还需要额外的一位数据输入到 32 位 LZC 中，且输入的数据不能影响 NORM 指令的正确性，因此在这里 LZC 的最低位的输入被设置为恒 1。最后，将 32 位 LZC 的结果输出，此即为 NORM 指令的结果。

Chapter 2

电路图设计

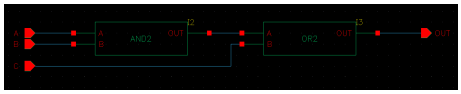
本章节将给出 NORM 指令的几个主要子模块的电路图设计，包括基本的与门、或门、反向器、多位选择器、LZC 模块等。

图2.1所示为主要单元的电路图设计。

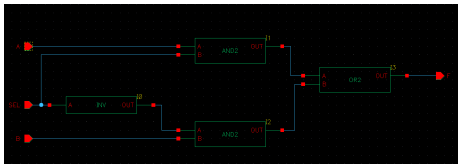


(a) 基本单元或非门

(b) 31 位反向器



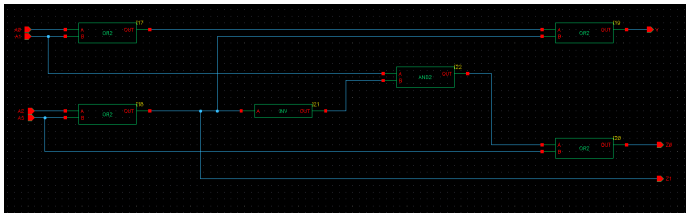
(c) 与或模块



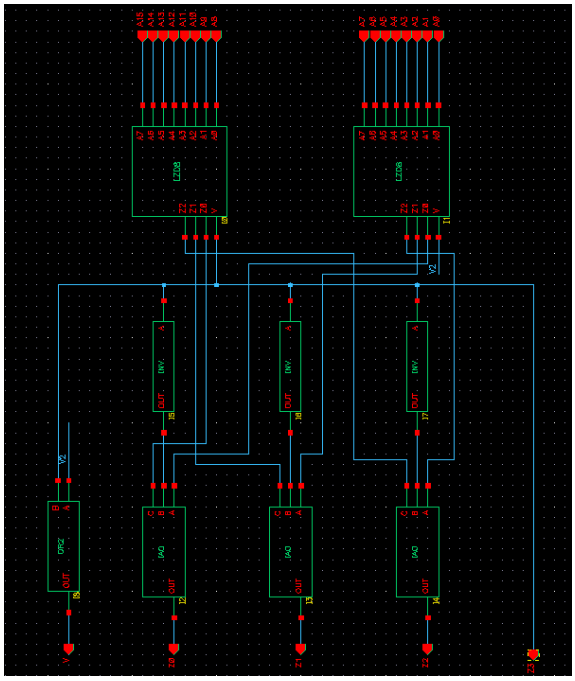
(d) 两路选择器

Figure 2.1: 主要基本单元的电路图设计

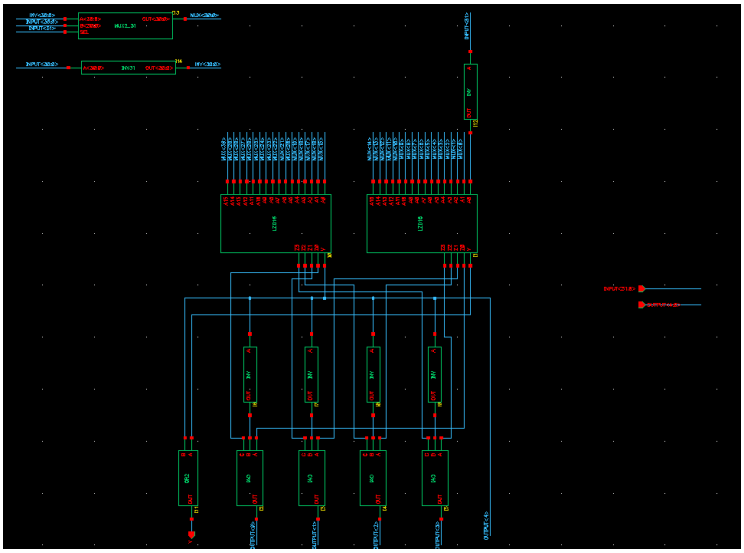
图2.2为 LZC 模块的电路图设计，包括 4-bit、16-bit、32-bit 等不同层次。



(a) 4-bit LZC 模块



(b) 16-bit LZC 模块



(c) 32-bit LZC 模块

Figure 2.2: 主要基本单元的电路图设计

从图2.2可以看出，32 位 LZC 模块可由两个 16-bit LZC 子模块构成，此外还包括一个两路 31 位选择器，选择信号为输入操作数的符号位。需要注意的是，图2.2(c)所示的模块的输出为 6 为，其中包括 5 位的 LZC 计算结果，还有 1 位信号 V 用于支持更宽的操作数。图2.3为基于 32 位 LZC 模块以及 NORM 指令的功能要求实现的最终的电路图。此时，输出信号 V 被接地，且 LZC 计算结果被扩展为 32 位，高位 ($Lzc < 31:5 >$) 通过 27 个反向器置零。

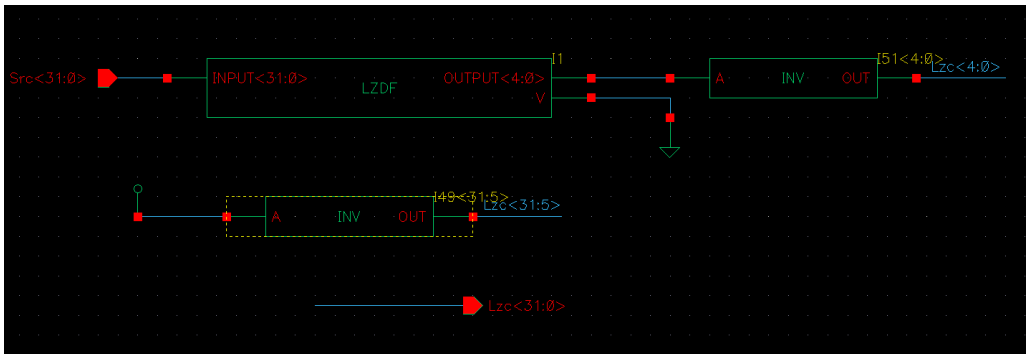


Figure 2.3: NORM 指令的实现电路

电路图中所有单元用到的 NMOS、MPOS 器件的长度均设为 0.13um，NMOS 的宽度设为 0.6um,PMOS 的宽度设为 0.8um。

Chapter 3

功能验证

本章节主要基于上一章的电路图设计进行功能验证，用到的主要工具软件为NC。

3.1 NC 简介

NC-Verilog 是 Candence 公司的 Verilog 数字逻辑模拟器，具有运行快、精度高、调试功能强大、使用灵活等优点，是业界公认的黄金模拟器。NC-Verilog 的运行分为两个步骤，首先是编译，检查语法错误等，然后模拟执行。NC-Verilog 使用 NCC 技术，提高了模拟的性能减少了内存的使用。采用 INCA 架构，具备了支持多种 HDL 语言、多设计层次、数模混合设计的能力。

运行流程包括：建立列表文件，列表文件包含激励信号；建立配置文件，以一个脚本的形式对 NC 进行配置，包括启动 GUI 界面、开启覆盖率统计、访问权限等设置；运行 NC-Verilog，以上一步的配置启动 NC；波形查看与调试，可以通过“shm_open”和“shm_probe”两个命令来保存波形，并且可以查看信号的波形以及逻辑结构图等，这一步主要通过查看波形以确定模块的正确性；查看覆盖率，这一步同样需要一个 TCL 脚本文件进行配置，并借助 ICCR 软件进行分析；导出 VCD 文件，可以通过 tcl 脚本或 NC 软件或“dumpfile”“dumpvars”来完成。以上就是功能验证部分的工作流程。

3.2 NC 验证结果

由于存在多个模块需要验证，为方便操作，在完成“或”模块的功能验证后，借助如下所示脚本来准备下一模块功能验证所需的文件以及部分内容等。

Listing 3.1: 准备 NC 所需的文件脚本

```
#!/bin/sh

function copyfile(){
        echo "Prepare␣files␣for␣NC"
        touch $1/$1_tb.v
        head -n 5 ./OR2/OR2_tb.v > $1/$1_tb.v
        mv $1.v $1
        cp ./OR2/filelist.v $1
        cp ./OR2/ncverilog.options $1
```

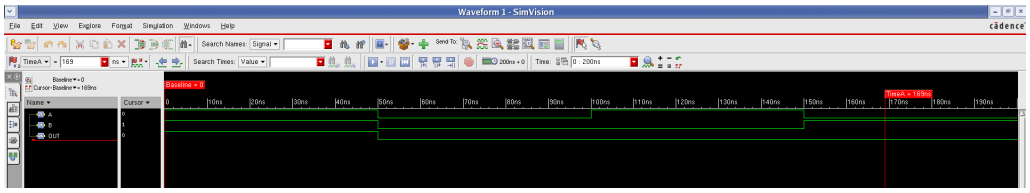
```
        cp ./OR2/run* $1
}
if [ ! -d "$1" ]; then
        mkdir "$1"
        echo "$1 is created"
        copyfile "$1"
else
        echo "$1 already existed"
        copyfile "$1"
fi
```

其中`head -n 5 ./OR2/OR2_tb.v > $1/$1_tb.v`为准备各个子模块公用的 Testbench 信息，共 5 行，具体内容如下：

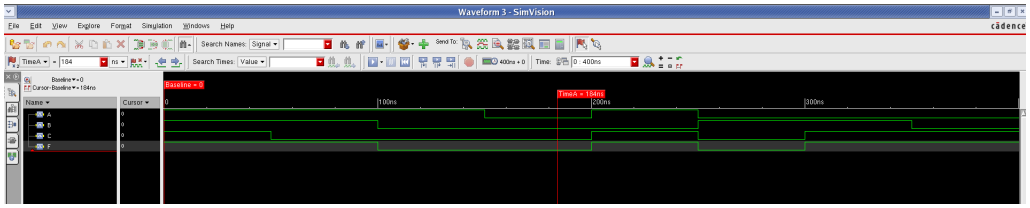
```
`timescale 10ns/1ns
module cds_globals;
supply1 VDD_;
supply0 GND_;
endmodule
```

Testbench 剩下的内容即为根据具体模块的实例化以及激励的产生等。

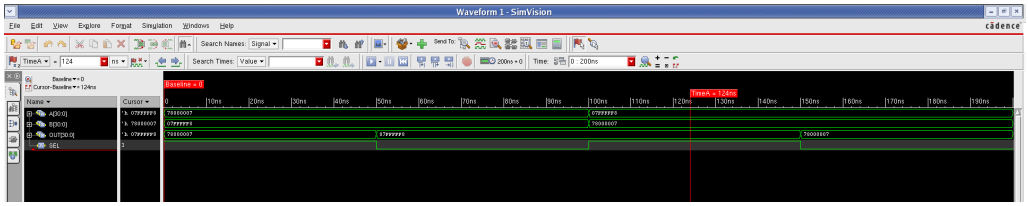
图3.1所示为基本单元的功能验证波形。



(a) 二输入与的验证结果



(b) 三输入与或的验证结果

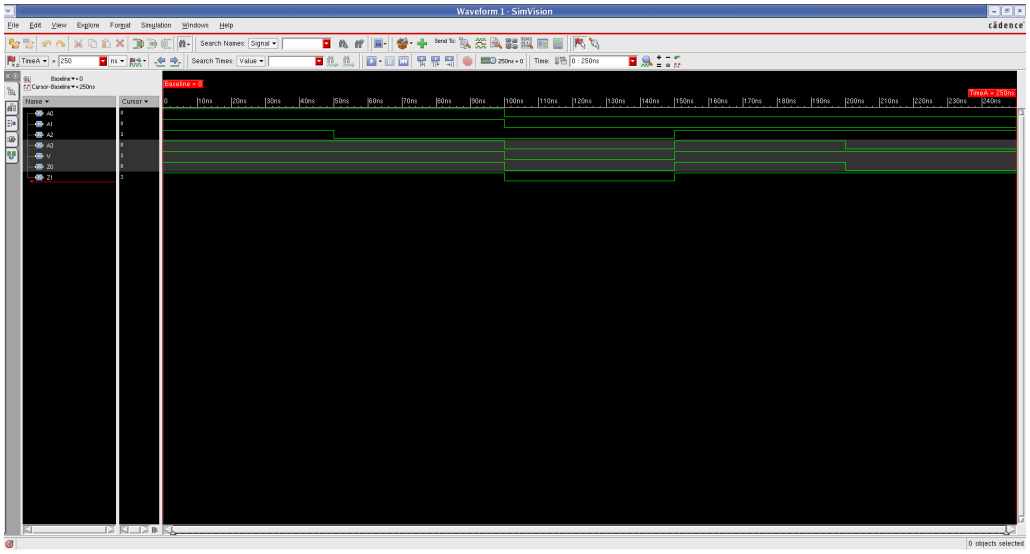


(c) 两路 31 位选择器

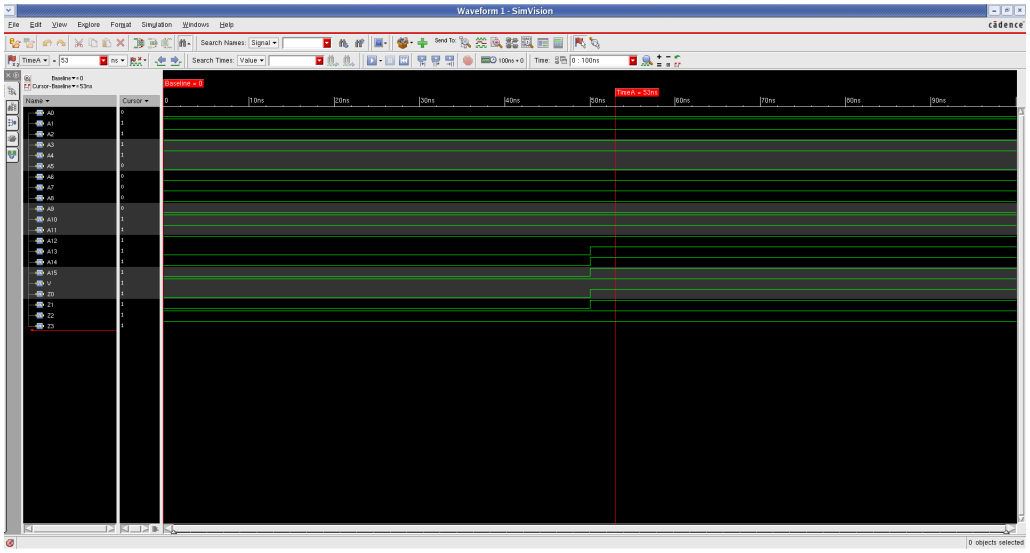
Figure 3.1: 基本单元的 NC 验证结果

图3.2所示为各级 LZC 模块的功能验证波形。

最后，基于图2.3所示的 NORM 指令的 NC 验证结果如图3.3所示。表3.1 给出了 LZC 模块以及 NORM 指令模块的输入与输出的关系，其中 LZC 模块的输出为真实值的相反数，且在输入数的所有位全为 0 的时候，V 也为 0(实际值为 1)。从



(a) 4-bit LZC 验证结果



(b) 16-bit LZC 验证结果

Figure 3.2: LZC 单元的 NC 验证结果

表3.1可以看出，图2.3所示的 **NORM** 电路图可以正确的实现 **NORM** 指令的功能，即统计符号位后与符号位相同值的连续位数。

Table 3.1: LZC 模块以及 NORM 模块的 NC 验证结果

模块	输入	输出
4-bit LZC	0100	10
	1011	11
	0000	00($V=0$)
16-bit LZC	16'h1C1E	1100
	16'hFC00	1111
NORM 模块	32'h0000_FFFF	32'h0000_000F
	32'hFFFF_0000	32'h0000_000F
	32'h0F00_0000	32'h0000_0003
	32'h00F0_0000	32'h0000_0007
	32'hFF0F_0000	32'h0000_0007

对于 NORM 指令模块，输入 (Src) 为 $32'hFFFF\_0000$ 和 $32'h0000\_FFFF$ 时，输出 (Lzc) 是相同的，均为 $32'h0000\_000F$，由于 NORM 指令的统计过程并不包括符号位在内，因此最终的结果为 15 而不是 16，同理，输入为其它数值时也有类似结果。

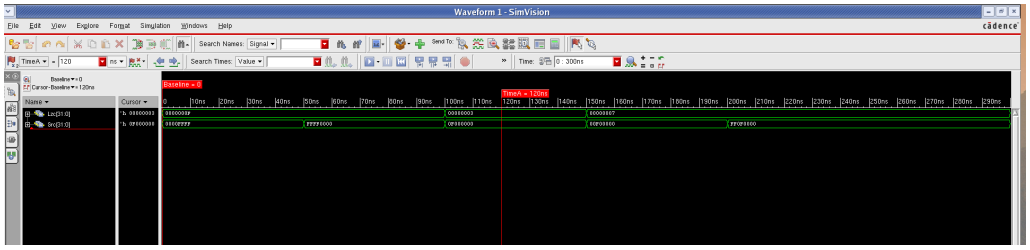


Figure 3.3: NORM 模块 NC 验证结果

Chapter 4

时序分析与电路优化

本章节主要对第二章设计的电路图导出.cdl 网表借助 HSpice 工具软件进行时序分析，即输入到输出延时的大小的检测。

4.1 工具简介

HSPICE 是 Meta-Software 公司为集成电路设计中的稳态分析，瞬态分析和频域分析等电路性能的模拟分析而开发的一个商业化通用电路模拟程序，它在伯克利的 SPICE（1972 年推出），MicroSim 公司的 PSPICE（1984 年推出）以及其它电路分析软件的基础上，又加入了一些新的功能，经过不断的改进，目前已被许多公司、大学和研究开发机构广泛应用。

HSPICE 可与许多主要的 EDA 设计工具，诸如 Cadence,Workview 等兼容，能提供许多重要的针对集成电路性能的电路仿真和设计结果。采用 HSPICE 软件可以在直流到高于 100GHz 的微波频率范围内对电路作精确的仿真、分析和优化。在实际应用中，HSPICE 能提供关键性的电路模拟和设计方案，并且应用 HSPICE 进行电路模拟时，其电路规模仅取决于用户计算机的实际存储器容量。

本次实验中需要对网表进行修改，相关原理主要参考文献 [2], 电子书版权为复旦大学所有。

4.2 基于 HSpice 的时序结果

在由 Composer 导出 NORM 模块的网表后，需要在网表中添加工艺库以及激励等信息，这两部分信息如下所示，首先确定.13 的工艺库以及温度等参数。

```
0.13um INV Characteristics
.options LIST NODE POST
.lib 'l013_v2p5.lib' TT
.TEMP 25
.GLOBAL VDD, GND
OP
```

下面是网表中输入的激励:

```
X Lzc<31> Lzc<30> Lzc<29> Lzc<28> Lzc<27> Lzc<26> Lzc<25>
+Lzc<24> Lzc<23> Lzc<22> Lzc<21> Lzc<20> Lzc<19> Lzc<18> Lzc<17> Lzc<16>
```

```
+Lzc<15> Lzc<14> Lzc<13> Lzc<12> Lzc<11> Lzc<10> Lzc<9> Lzc<8> Lzc<7> Lzc<6>
+Lzc<5> Lzc<4> Lzc<3> Lzc<2> Lzc<1> Lzc<0> Src<31> src1<30> src1<29>
+Src<28> Src<27> Src<26> Src<25> Src<24> Src<23> Src<22> Src<21>
+Src<20> Src<19> Src<18> Src<17> Src<16> Src<15> Src<14> Src<13>
+Src<12> Src<11> Src<10> Src<9> Src<8> Src<7> Src<6> Src<5> Src<4>
+Src<3> Src<2> Src<1> Src<0>
+NORM
V1 VDD 0 1.2V
V4 Src<31> 0 0
V5 Src<30> 0 0
V6 Src<29> 0 0
V7 Src<28> 0 pulse(0 1.2V 0.5N 0N 0N 2N 4N)
V8 Src<27> 0 1.2V
V9 Src<26> 0 1.2V
V10 Src<25> 0 1.2V
V11 Src<24> 0 1.2V
V12 Src<23> 0 0
V13 Src<22> 0 0
V14 Src<21> 0 0
V15 Src<20> 0 0
V16 Src<19> 0 0
V17 Src<18> 0 0
V18 Src<17> 0 0
V19 Src<16> 0 0
V20 Src<15> 0 0
V21 Src<14> 0 0
V22 Src<13> 0 0
V23 Src<12> 0 0
V24 Src<11> 0 1.2V
V25 Src<10> 0 0
V26 Src<9> 0 1.2V
V27 Src<8> 0 1.2V
V28 Src<7> 0 1.2V
V29 Src<6> 0 1.2V
V30 Src<5> 0 1.2V
V31 Src<4> 0 1.2V
V32 Src<3> 0 1.2V
V33 Src<2> 0 1.2V
V34 Src<1> 0 1.2V
V35 Src<0> 0 1.2V
.tran 0.2n 10n
.end
```

根据 NORM 指令的具体功能，在输入端口 Src<28> 输入周期为 4ns 的脉冲瞬态电压源，脉冲宽度为 2ns。定义电压脉冲源的具体语法如下所示：

```
Vxxx n+ n- PU<LSE> <(>v1 v2 <td <tr <tf <pw <per>>>>>  <)>
```

其中< >为可省略部分。.trans 0.2n 10n 表示工具共需要分析 10ns 的瞬态分析，步长为 0.2ns。

最后，HSpice 的时序分析结果如图4.1所示。从图中可以看出，当输入为最高 5 位 (包括符号位) 由 00001 变为 00011 时，输出的最低两位由 11 变为 10，其输出延时为 395ps(如图所示)，同时，输出随输入的变化情况与 NC 的仿真波形都说明设计的 NORM 指令模块功能的正确性！

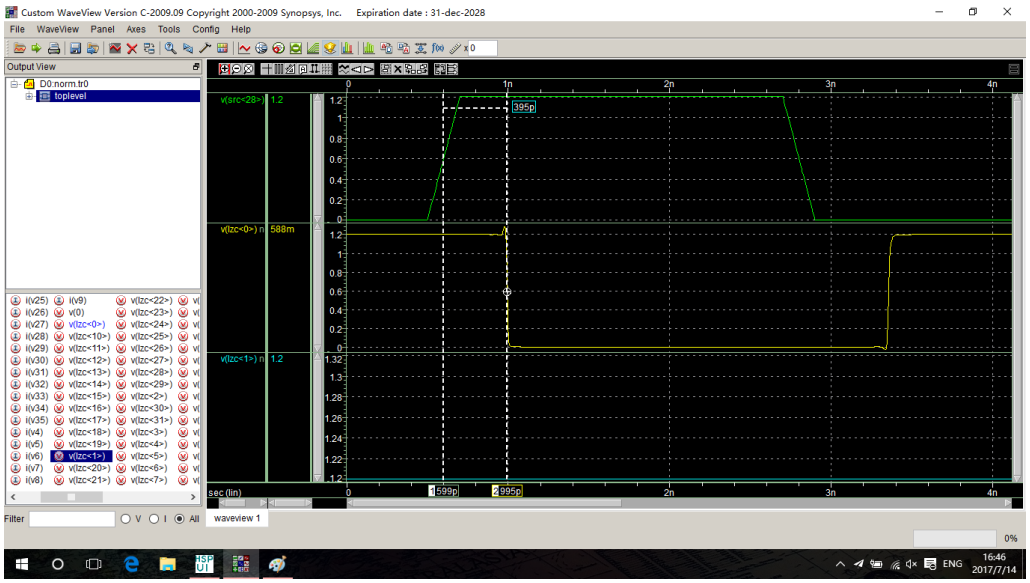


Figure 4.1: NORM 指令的 HSpice 分析结果

Chapter 5

版图设计与验证

本章节主要给出 NORM 指令基于第二章电路图的版图设计，包括电路图中的所有模块的版图设计，并对最终的设计进行 DRC、LVS 检查。为了控制篇幅，这里仅给出所有子模块中几个主要模块的版图、DRC 验证结果、LVS 验证结果的截图。

需要指出的是，在完成版图设计过程中，为了方便电源、地的走线，基本单元的高度被设置为同样的高度；为了减少需要的金属层，在连线过程中采取的基本原则是：横向采用 *M1* 层金属，纵向连线采用 *M2* 层金属，这样有利于减少不必要的绕线，因此，本次 *NORM* 的设计所有的连线都在 *M1*、*M2* 层内完成，没用到高层金属，但这样也导致设计中需要的 *M1*、*M2* 层间的通孔数量较多；最后，为了实现 *NORM* 指令最终的版图尽可能更规则，*2* 路 *31* 位选择器采用双列的形式实现，这样就可以保证 *NORM* 模块最终可实现近似正方形的版图。

下面对几个主要模块进行说明，包括版图、DRC 验证结果以及 LVS 验证结果。

AND2 模块

本模块主要完成量输入与门的设计。其版图、DRC 检查结果、LVS 检查结果如图5.1所示。

IAO 模块

本模块主要完成三输入与或逻辑，即输入 A 与 B，然后其结果再与 C 进行或操作。模块的版图等结果如图5.2所示。

MUX2_1 模块

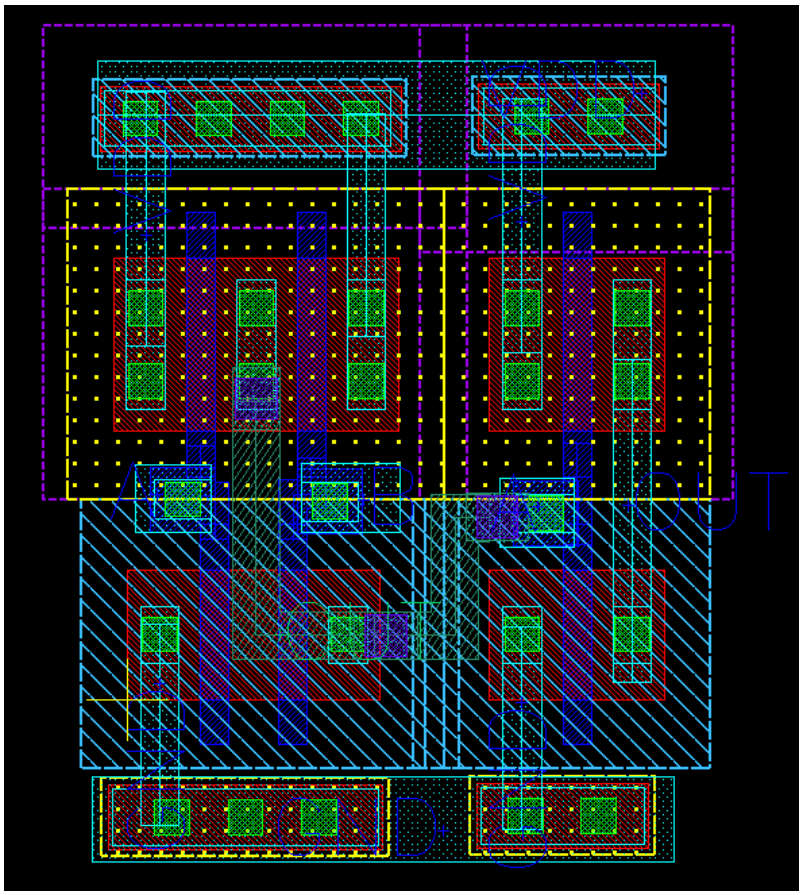
本模块主要完成两路一位选择器的功能，其结果如图5.3所示，为防止出现有源区距离过小的 DRC 错误，其各个顶层单元之间的有源区彼此重合。

LZD4 模块

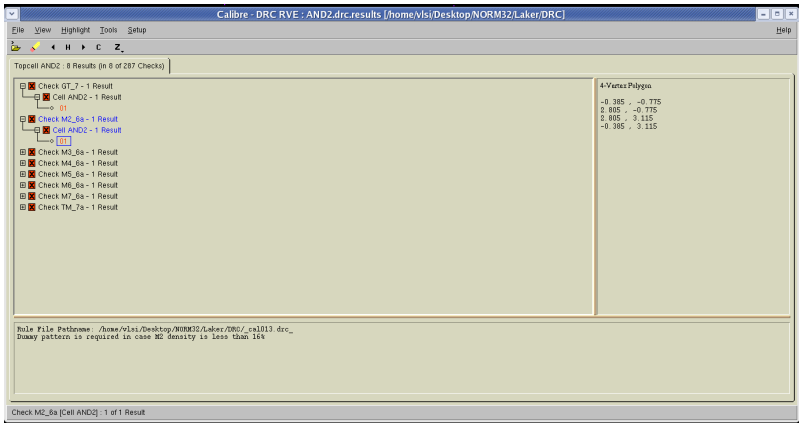
本模块完成 4-bit 输入数据的前导零检测功能，输出为 2 位，且真实结果为输出的反向。其结果如图5.4所示。

LZD16 模块

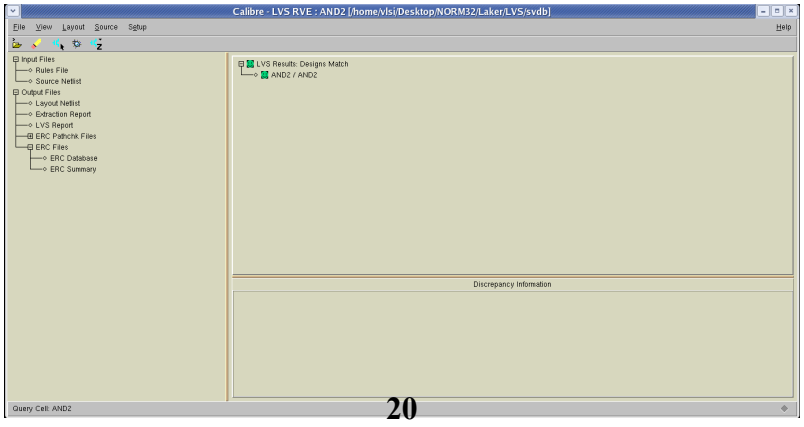
本模块在 LZD8(省略) 的基础上完成 16-bit 输入数据的前导零计算功能，其输出同样为真实结果的反向，结果如图5.5所示。



(a) AND2 版图

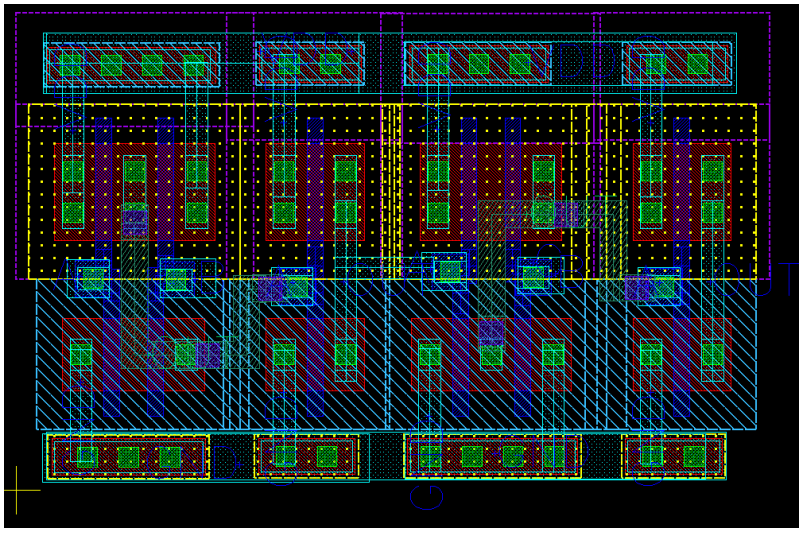


(b) AND2 DRC 验证结果

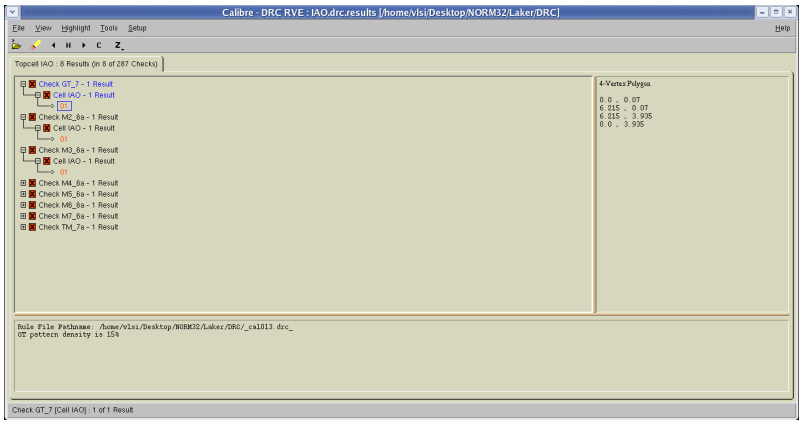


(c) AND2 LVS 验证结果

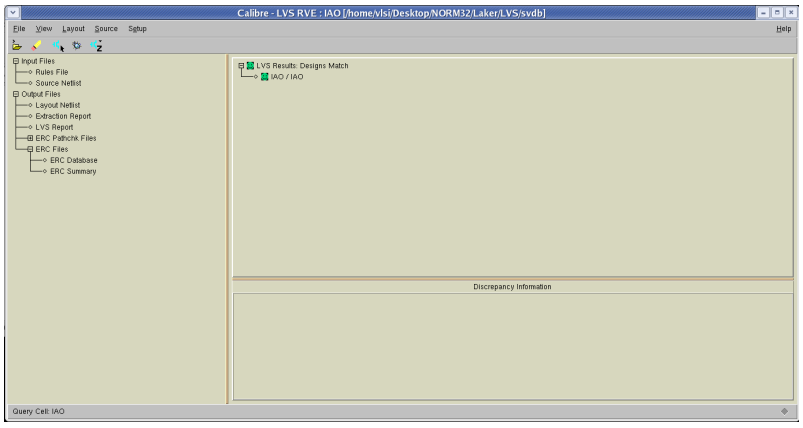
Figure 5.1: AND2 版图设计及其验证结果



(a) IAO 模块版图

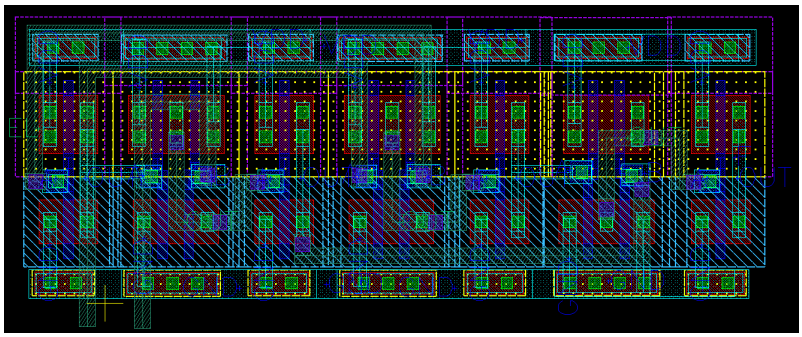


(b) IAO DRC 验证结果

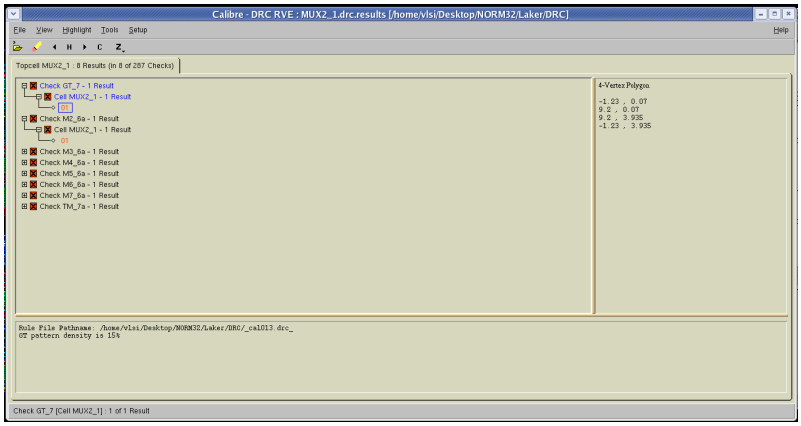


(c) IAO LVS 验证结果

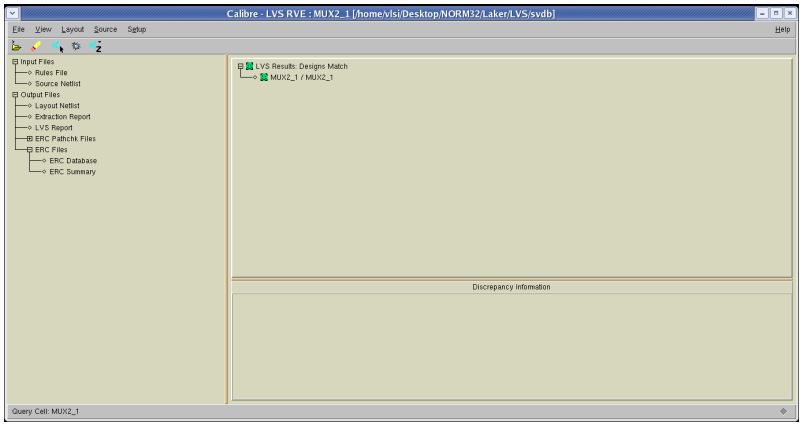
Figure 5.2: IAO 版图设计及其验证结果



(a) MUX2_1 模块版图

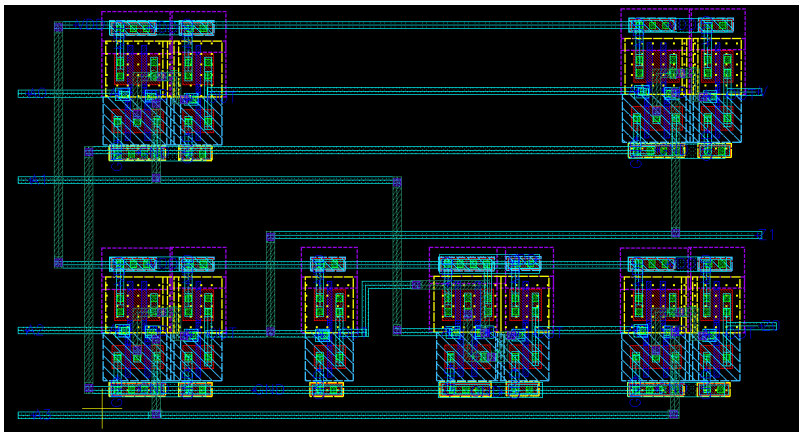


(b) MUX2_1 DRC 验证结果

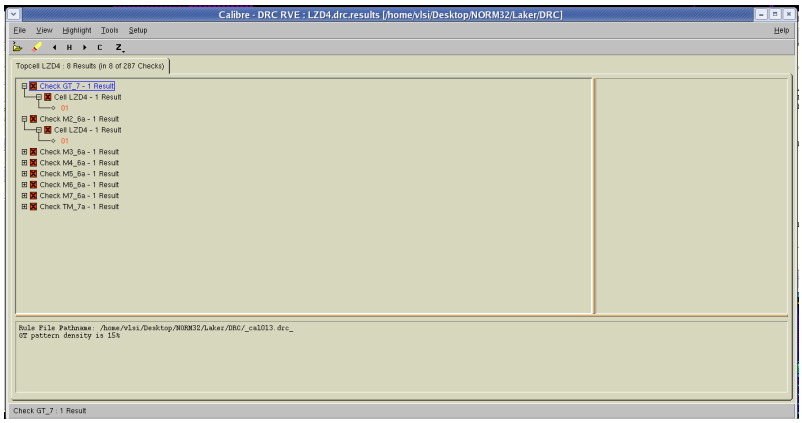


(c) MUX2_1 LVS 验证结果

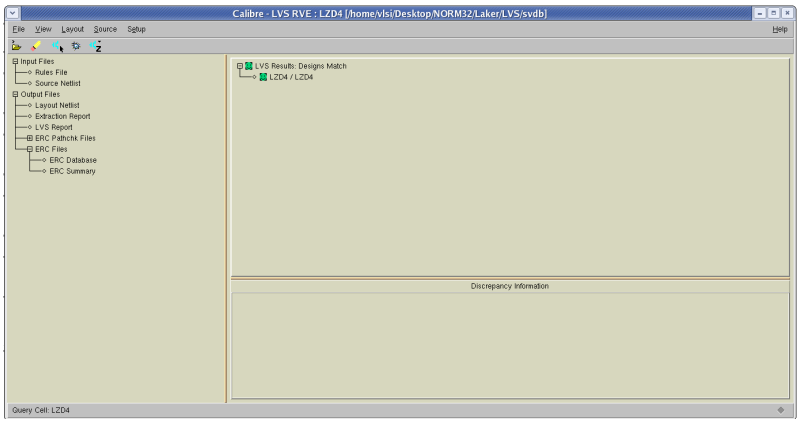
Figure 5.3: MUX2_1 版图设计及其验证结果



(a) LZD4 模块版图

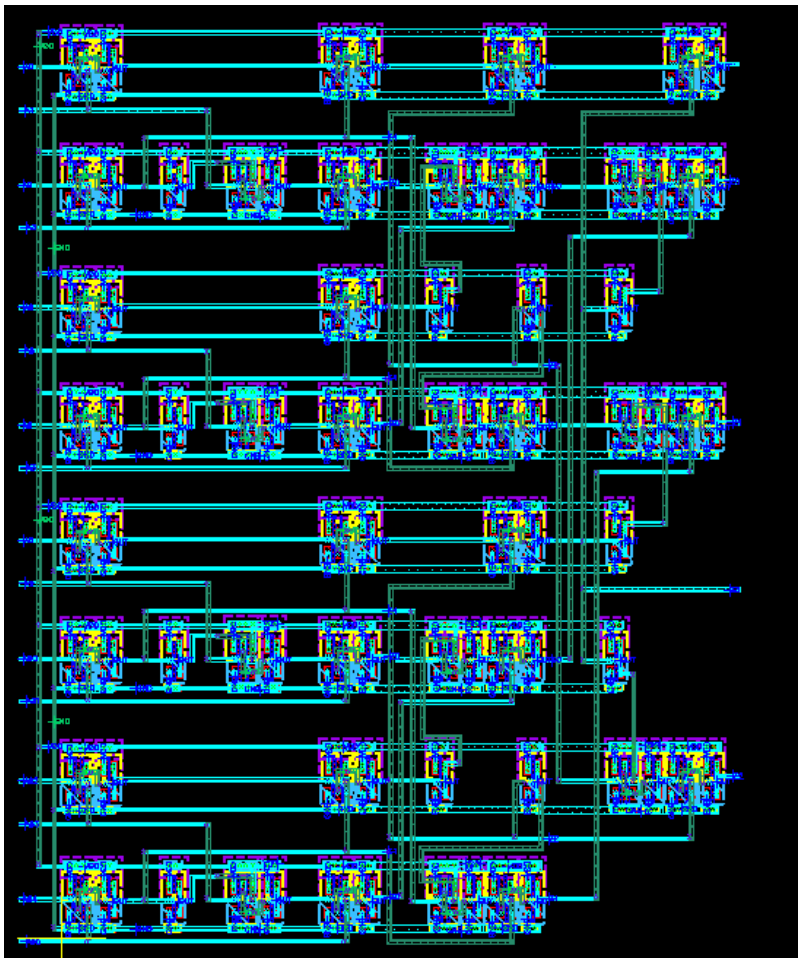


(b) LZD4 DRC 验证结果

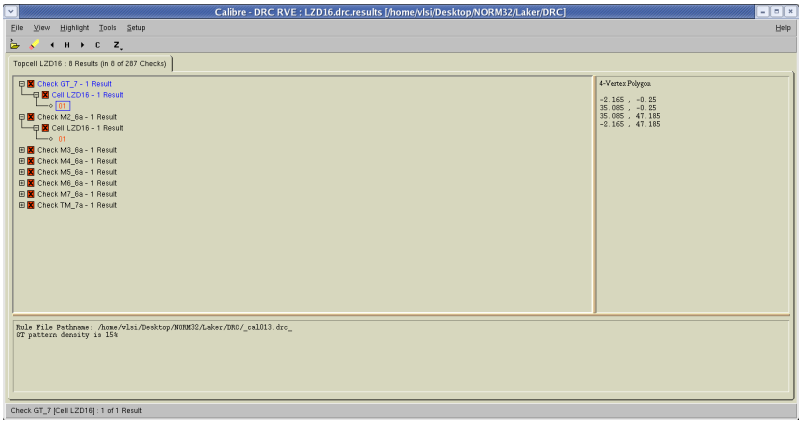


(c) LZD4 LVS 验证结果

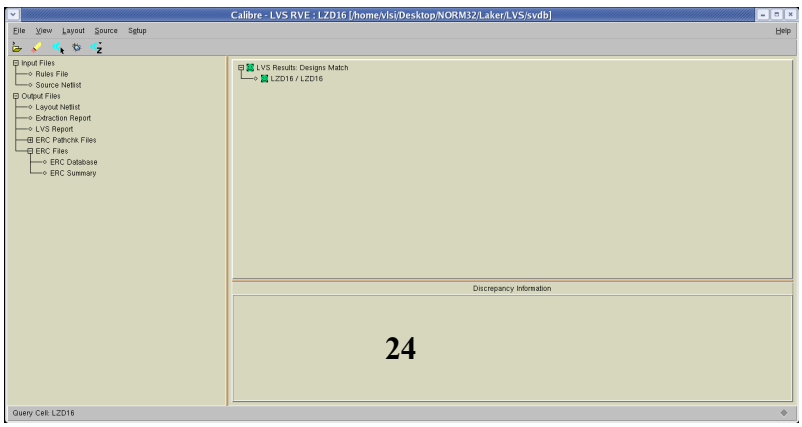
Figure 5.4: LZD4 版图设计及其验证结果



(a) LZD16 模块版图



(b) LZD16 DRC 验证结果



(c) LZD16 LVS 验证结果

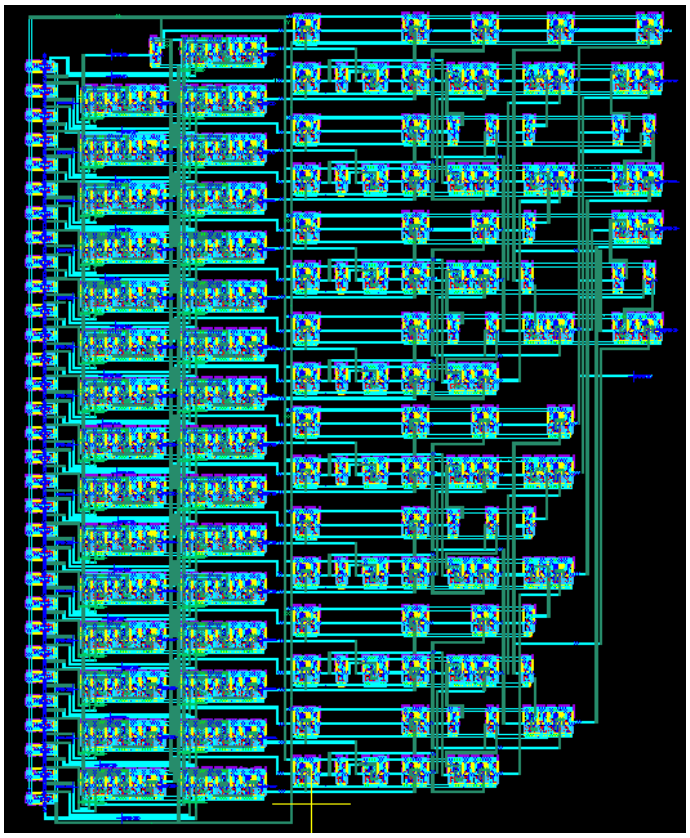
Figure 5.5: LZD16 版图设计及其验证结果

LZDF 模块

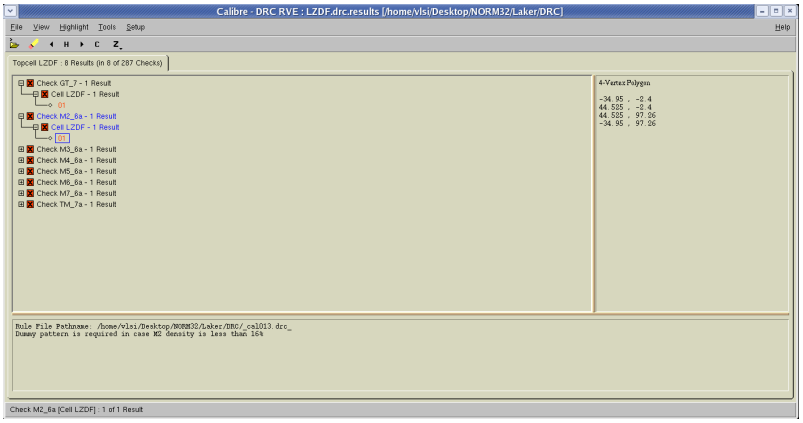
本模块 (LZDF: LZD Final) 在 LZD16 的基础上完成 32-bit 输入数据的前导零计算功能，其输出同样为真实结果的反向，此外，其输入数据也由以前的直接输入变为从选择器输入，而两路选择器的输入，一路为未处理的输入数据，另一路为按位取反输入数据的结果，选择信号为输入数据的符号位。结果如图 5.6所示。

NORM 模块

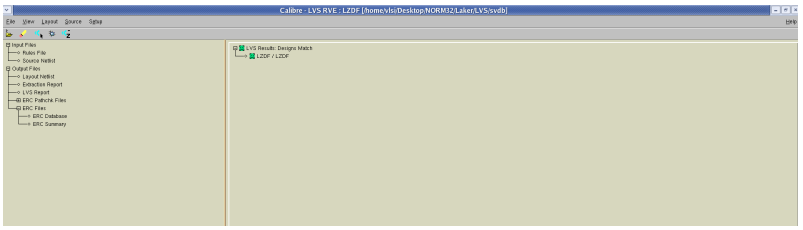
最后，本模块在 LZDF 模块的基础上，完成 NORM 指令模块的版图设计，其输出为 32 位，其中高位 (31∼5 恒为 0)。其结果如图5.7所示。



(a) LZDF 模块版图

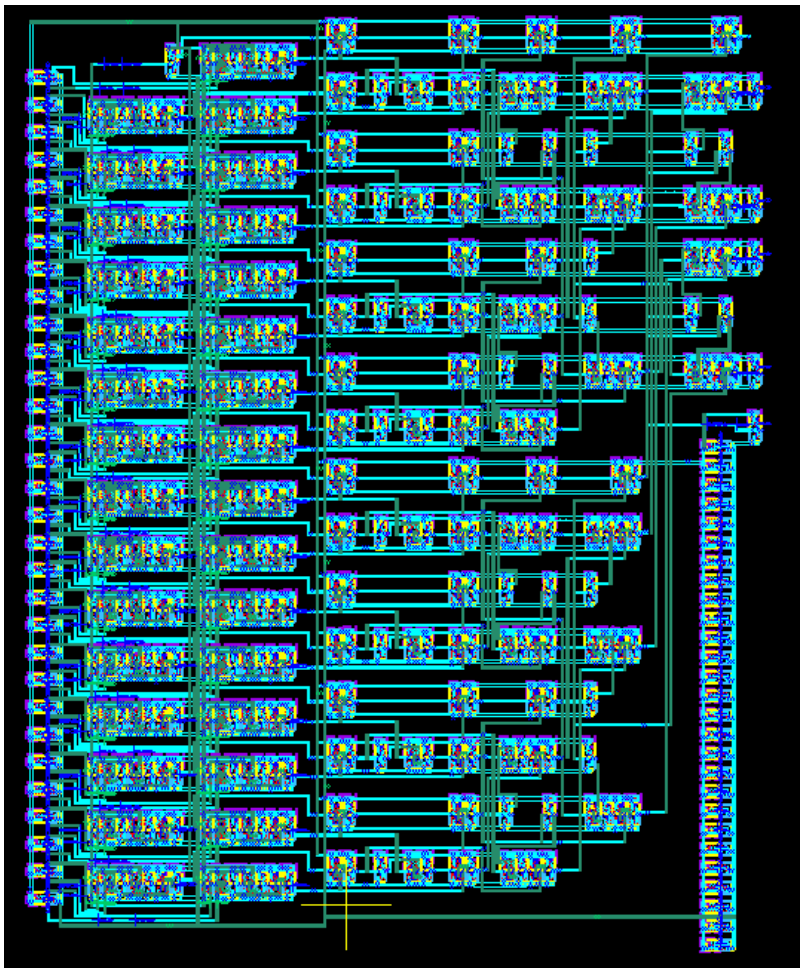


(b) LZDF DRC 验证结果

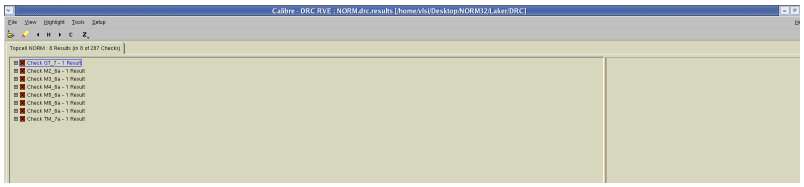


(c) LZDF LVS 验证结果

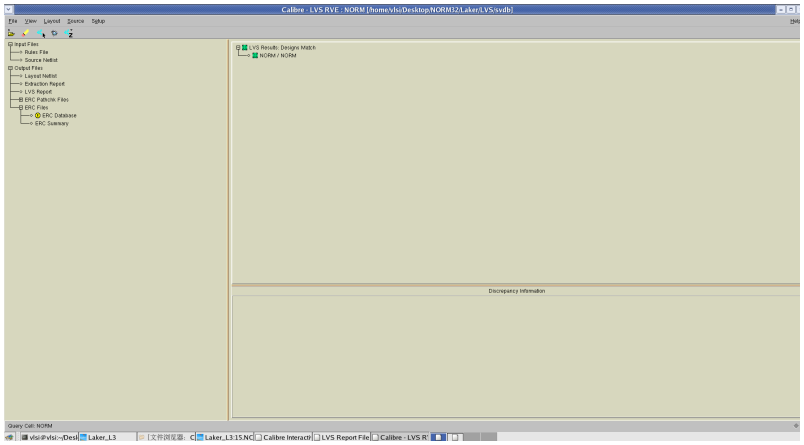
Figure 5.6: LZDF 版图设计及其验证结果



(a) NORM 模块版图



(b) NORM DRC 验证结果



(c) NORM LVS 验证结果

Figure 5.7: NORM 版图设计及其验证结果

Chapter 6

实验总结

通过完成本次实验，熟悉了数字电路全定制流程，包括最初的功能分析、电路图设计、功能验证、时序分析与优化以及后来的版图设计以及 DRC、LVS 验证等。实验结果表明，设计的 NORM 指令模块可以实现正确的功能 (NC、HSpice 结果)，输入至输出的响应延时为 395ps(HSpice 结果)，且版图设计与电路图设计一致、没有 DRC 错误 (通过 DRC、LVS 验证)。下面对本次实验进行总结。

6.1 试验中遇到的问题与解决办法

回顾整个实验流程，遇到的几个主要问题及其解决办法如下：

- **由 Composer 导出电路图的 NC-Verilog 文件以及.cdl 网表过程中，导出失败**
  解决办法：通过对导出文件的过程进行分析，发现在导出顶层模块的上述文件时，需要首先将顶层模块包含的所有底层模块的上述文件导出，然后即可解决错误。
- **在基于设计的.cdl 网表进行时序分析时，输出并不会跟随输入变化**
  解决办法：通过.sp 文件中瞬态源的周期得以解决，即将在 Src<27> 输入端口的脉冲源的周期由最开始的 2ns 增加为 4ns，同时将高电平维持时间由原来的 1ns 改为 2ns 后，输出正常，其结果见图4.1。
- **版图设计中 DRC 验证出现问题**
  解决办法：一般发生 DRC 违反，主要由版图设计中存在电器规则违反所致，如金属层之间的间距过小、金属的面积过小等错误。根据提示修改这些错误后，所有模块均通过 DRC 检查。
- **版图设计中 LVS 验证出现问题**
  解决办法：一般发生 LVS 错误，可以是.cdl 文件与.gds 文件之间存在不一致导致，包括版图中器件的尺寸与电路图中器件的尺寸或名字不一致、连线存在错误等。经过对.cdl 文件以及版图中不一致的地方进行修改后，所有模块的 LVS 检查均通过验证。

6.2 实验收获与不足

从这一次实验中熟悉了各种全定制工具的使用, 对于 Virtuoso 工具的使用更加熟练, 对于版图设计规则有了更全的认识, 能够较好的运用工具来实现设计, 对全定

制设计有了更全面的理解。

通过完成 tms320 DSP 的 NORM 指令的全定制，对实际数字电路功能的实现有了更深的理解，包括功能分析以及具体电路的设计等，为以后的工作打下了坚实的基础。在电路设计过程中，学习了如何通过阅读论文来找出比较有效的解决方案，更重要的，通过完成低功耗的 LZC 模块，了解了一种低功耗设计思路，即通过逻辑优化以及器件共享来减少所需的器件的数量，从而降低电路的整体功耗。

在这段时间里部分完成 NORM 指令的全定制设计, 自己动手实践能力有了很大的提高, 通过跟同学, 老师的交流, 体会到合作的重要性。更重要的，通过解决遇到的问题，锻炼了解决问题的能力。

当然本次实验还存在许多不足，包括最终的版图设计中对电路的面积还存在优化的空间，布局可以更合理一些。在前期功能验证中，下一步的工作可以是引入黄金模型对电路图的正确性进行验证，虽然从已有的几个输入激励中全部得到正确的结果，但还需要提高验证的覆盖率。

References

[1] Giorgos Dimitrakopoulos, Kostas Galanopoulos, Christos Mavrokefalidis, and Dimitris Nikolos. Low-power leading-zero counting and anticipation logic for high-speed floating point units. *IEEE transactions on very large scale integration (VLSI) systems*, 16(7):837--850, 2008.

[2] 宫志超. *HSPICE* 简明教程. 复旦大学, 2007.