**HScaler設計報告書**

**Rivision: 0.01**

**作成者:王瑶**

目录

[1. 設計概要 3](#_Toc459931994)

[2. I/F仕様 4](#_Toc459931995)

[3. デザイン仕様 6](#_Toc459931996)

[4. 検証結果 9](#_Toc459931997)

[5. 論理合成結果 13](#_Toc459931998)

[6. 設計環境 14](#_Toc459931999)

[7. 設計履歴 15](#_Toc459932000)

[8. 更新履歴 17](#_Toc459932001)

[9.残件 18](#_Toc459932002)

# 設計概要

**1.1要求仕様**

**表1.1要求仕様**

|  |  |
| --- | --- |
| **項目** | **詳細** |
| スケーラー仕様 | 水平1/N間引き(N=1, 2, 4) |
| フィルター | lanczosフィルター(4tap) |
| 入力フォーマット | bmp (R: 8bit G: 8bit B: 8bit) |
| 出力フォーマット | bmp (R: 8bit G: 8bit B: 8bit) |
|  | 1pixel/clock |

**1.2実現した機能**

用Verilog语言设计实现了对标准的BMP格式的图片的水平1/N（N=1,2,4）缩放。

该设计的输入是bmp格式图片RGB(R: 8bit G: 8bit B: 8bit)的值，输出的也是bmp格式的图片(R: 8bit G: 8bit B: 8bit)，每个时钟上升沿输入/输出一个像素。

处理像素的算法是lanczos算法。缩放的实现是通过计数器来实现。

其中可以选择的功能有：模式选择、缩放倍数选择、滤波器系数的选择等功能。

**1.3参考資料**

《图像缩放》 360百科

《BMP图片文件格式和位图资料》 博客

《几种插值算法的对比研究》 Trent1985的专栏

《verilog实现bmp图片的旋转》 天津大学，王玉斌

《Lanczos resampling》 Wikipedia,the free encyclopedia

《verilog数字系统设计教程》 夏宇闻

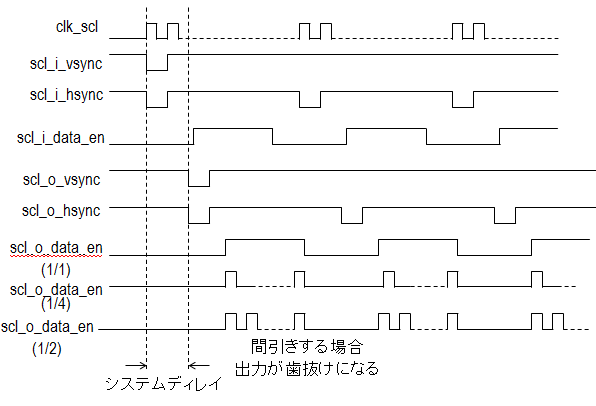
# I/F仕様

**2.1端口列表**

**表2.1端口列表**

|  |  |  |  |
| --- | --- | --- | --- |
| **Port name** | **I/O** | **wide** | **role** |
| rst\_n\_scl | I | U1 | 非同期リセット(負論理) |
| clk\_scl | I | U1 | クロック |
| scl\_i\_vsync | I | U1 | 垂直同期信号入力、フレーム開始(負論理) |
| scl\_i\_hsync | I | U1 | 水平同期信号入力、ライン開始(負論理) |
| scl\_i\_data\_en | I | U1 | 入力データvalid信号 |
| scl\_i\_data\_r | I | U8 | 入力データR |
| scl\_i\_data\_g | I | U8 | 入力データG |
| scl\_i\_data\_b | I | U8 | 入力データB |
| scl\_o\_vsync | O | U1 | 垂直同期信号出力 |
| scl\_o\_hsync | O | U1 | 水平同期信号出力 |
| scl\_o\_data\_en | O | U1 | 出力データvalid信号 |
| scl\_o\_data\_r | O | U8 | 出力データR |
| scl\_o\_data\_g | O | U8 | 出力データG |
| scl\_o\_data\_b | O | U8 | 出力データB |
| scl\_cfg\_mode | I | U1 | 動作モード  0: スルー　1:間引き |
| scl\_cfg\_rsz | I | U1 | 間引き倍数  0: ½　1: ¼ |
| scl\_cfg\_flt | I | U2 | フィルター係数選択 |

**2.2入出力タイミングチャート**



**图2.1 タイミングチャート**

如图2.1所示，在第一个时钟上升沿所有信号复位，在第二个时钟上升沿首先使水平同步信号和垂直同步信号有效，开始读入图片数据，因为只处理水平方向，而且像素是按照一行一行的方法读取的，因此垂直同步信号一直有效，水平同步信号每隔固定时间有效，输入数据使能信号来控制有效数据的读入，即在数据使能有效期间读入的像素是要处理的像素。输出数据使能来控制缩放的倍数，若是原图输出，则将所有有效的像素输出；若是1/2缩放，则将有效像素的一半输出；若是1/4缩放，则将有效像素的1/4像素输出，最后得到的输出波形如图所示。

# デザイン仕様

**3.1　設計概要**

图片缩放器处理图片的过程是：先将图片的RGB数据存入寄存器中，然后判断端点做端处理乘法，加法操作，最后对RGB值做平均操作，输出处理过后的RGB值；有效缩放使能scl\_i\_data\_en的输出，是先判断scl\_cfg\_mode和scl\_cfg\_rsz的值，决定缩放倍数，然后用计数器计数分频，最后输出锯齿形的scl\_o\_data\_en。无论是否进行滤波压缩处理，数据从进入到输出共需要6个clk，故将scl\_i\_data\_en延时6个clk后再进行输出。

该图像缩放器，实现办法分为两个模块进行。第一个模块是像素处理模块，第二个是缩放模块。

**3.1.1ピクセルを処理するモジュールが実現した機能**

（1）像素点RGB值的读入：定义六个八位寄存器，每个时钟上升沿读入一个像素点

（2）定义四种滤波器，根据输入信号scl\_cfg\_flt来选择不同的滤波器。

（3）端处理：使能信号的输入和RGB信号的输入定义六个寄存器，分别为data\_in\_x\_0, data\_in\_x\_1, data\_in\_x\_2,data\_in\_x\_3,data\_in\_x\_4,data\_in\_x\_5和en\_ff0,en\_ff1,en\_ff2,en\_ff3,

en\_ff4,en\_ff5，非端点像素输入时直接将寄存器data\_in\_x\_0, data\_in\_x\_1, data\_in\_x\_2,

data\_in\_x\_3和滤波器模板的值对应相乘，即

itmp\_r\_0\*coef\_3，itmp\_r\_1\*coef\_2，itmp\_r\_2\*coef\_1，itmp\_r\_3\*coef\_0

若为端点像素则需要根据en\_ffx的值判断端点类型再做处理。若（en\_ff2==1 && en\_ff3==0）

则是首端有效像素点，此时寄存器data\_in\_x\_0, data\_in\_x\_1, data\_in\_x\_2中值有效,

data\_in\_x\_3中的值无效，因此不能正常加权平均，处理的办法是，将第一个像素点即data\_in\_x\_2中的值输入两次补齐空缺相乘；若（en\_ff0==0 && en\_ff1==1）则为末端倒数第二个个点，同理，寄存器data\_in\_x\_1中的值输入两次分别相乘；若（en\_ff1==0 && en\_ff2==1）则为最末端点，将寄存器data\_in\_x\_2中的数输入三次分别和滤波器相乘。

**3.1.2間引くモジュールが実現した機能**

使用计数器控制输出使能，若为1/2缩放则输出使能每隔一个时钟输出一次，实现在两个像素点中选一个；若为1/4缩放则输出使能每隔3个时钟周期输出一次，实现在4个像素中输出一个像素。

**3.2構造図**



**图3.1 構造図**

图3.1为设计的顶层模块的输入输出结构图。

**3.3ブロック図**



**图3.2 ブロック図**

图3.2为两个模块的输入输出结构图。

**3.4ブロックの中に詳細な回路図**



**图3.3 詳細な回路図**

图3.3为总设计详细的实现电路图。

# 検証結果

**4.1検証の环境**



**图4.1検証の环境**

**4.2検証の方法**

Testbench中同时读入两幅图片，分别为期待值图片exp.bmp和待处理图片input.bmp。将读取到的input.bmp的RGB数据发送给hscaler，由hscaler做处理之后，再将结果返回到testbench，由testbench重新写入生成图片。在重新生成图片的过程中，同时和期待值数据作比较，若两点像素值相同，则不输出；若不相同，则会输出“ERRO”。

**4.3検証の项目**

**表4.1検証の项目**

|  |  |
| --- | --- |
| **機能検証** | **異常検証** |
| スルー動作 | mode是否受rsz和flt的异常影响 |
| 水平1/2間引き | 图片大小验证 |
| 水平1/4間引き | reset |
| フィルター |  |

**4.4検証リスト**

**表4.2検証リスト**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 序号 | 文件名 | 图像尺寸 | scl\_cfg\_mode | scl\_cfg\_rsz | scl\_cfg\_flt | 验证目的 |
| 1 | Input1 | 1280\*1024 | 0 | 0 | 0 | 不压缩验证 |
| 2 | Input1 | 1280\*1024 | 0 | 0 | 1 | scl\_cfg\_rsz和scl\_cfg\_flt  对scl\_cfg\_mode的影  响验证 |
| 3 | Input1 | 1280\*1024 | 0 | 1 | 0 |
| 4 | Input1 | 1280\*1024 | 0 | 1 | 1 |
| 5 | Input1 | 1280\*1024 | 1 | 0 | 0 | 1/2缩小验证 |
| 6 | Input1 | 1280\*1024 | 1 | 1 | 0 | 1/4缩小验证 |
| 7 | Input1 | 1024\*768 | 0 | 0 | 0 | 图片大小验证（原图） |
| 8 | Input1 | 1024\*768 | 1 | 0 | 0 | 图片大小验证（1/2压缩） |
| 9 | Input1 | 1024\*768 | 1 | 1 | 0 | 图片大小验证（1/4压缩） |
| 10 | Input2 | 640\*480 | 1 | 0 | 0 | 滤波器模块1验证 |
| 11 | Input2 | 640\*480 | 1 | 0 | 1 | 滤波器模块2验证 |
| 12 | Input2 | 640\*480 | 1 | 0 | 2 | 滤波器模块3验证 |
| 13 | Input2 | 640\*480 | 1 | 0 | 3 | 滤波器模块4验证 |

**4.5検証結果**

1.功能验证：输入1280\*1024大小的图片，输出如下验证结果

**图4.2 原图输出**

**1280\*1024**

**图4.4 1/4缩放**

**3320\*1024**

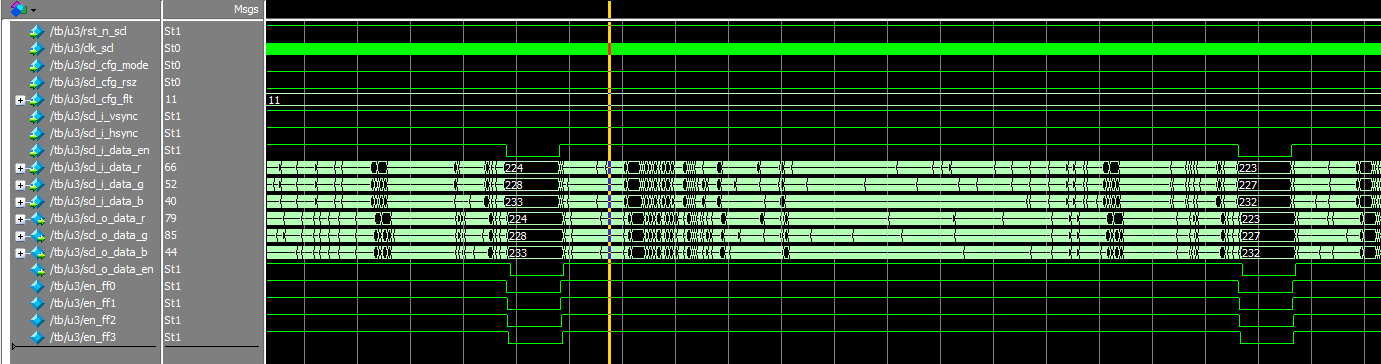
**图4.3 1/2缩放**

**6640\*1024**

2.时序验证

２.时序验证

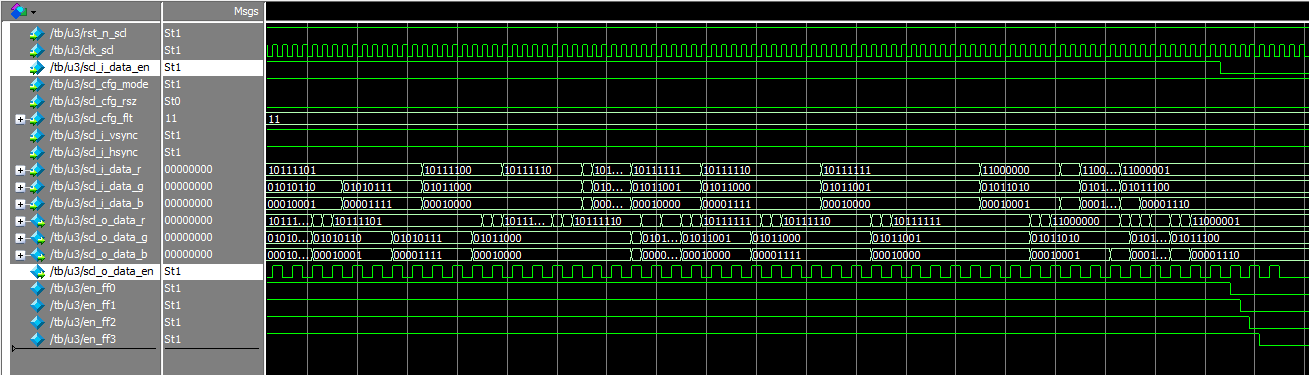
（1）スルー動作



**图 4.5 原图输出**

如图 4.5所示，原图输出的scl\_i\_data\_en和scl\_o\_data\_en的波形具有一致性，说明没有对图像做缩放处理，直接原图输出。

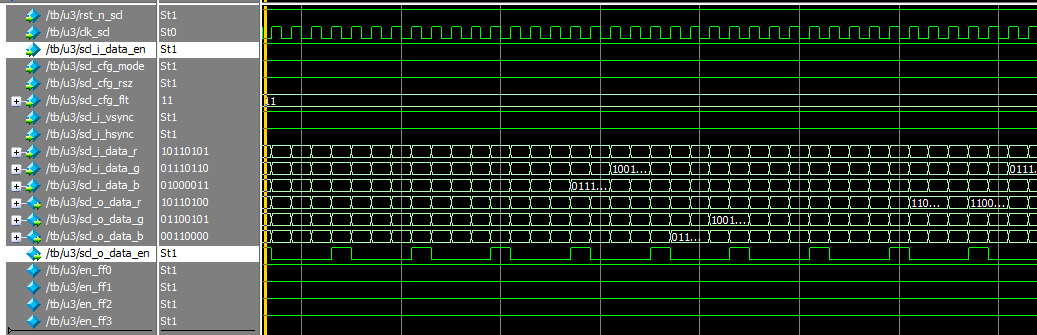
（2）1/2縮小



**图 4.6 1/2缩放输出**

如图 4.6所示，1/2缩放输出的scl\_o\_data\_en每隔一个时钟置1一次，将scl\_i\_data\_en波形分成了锯齿状输出，说明对图像做了1/2缩放。

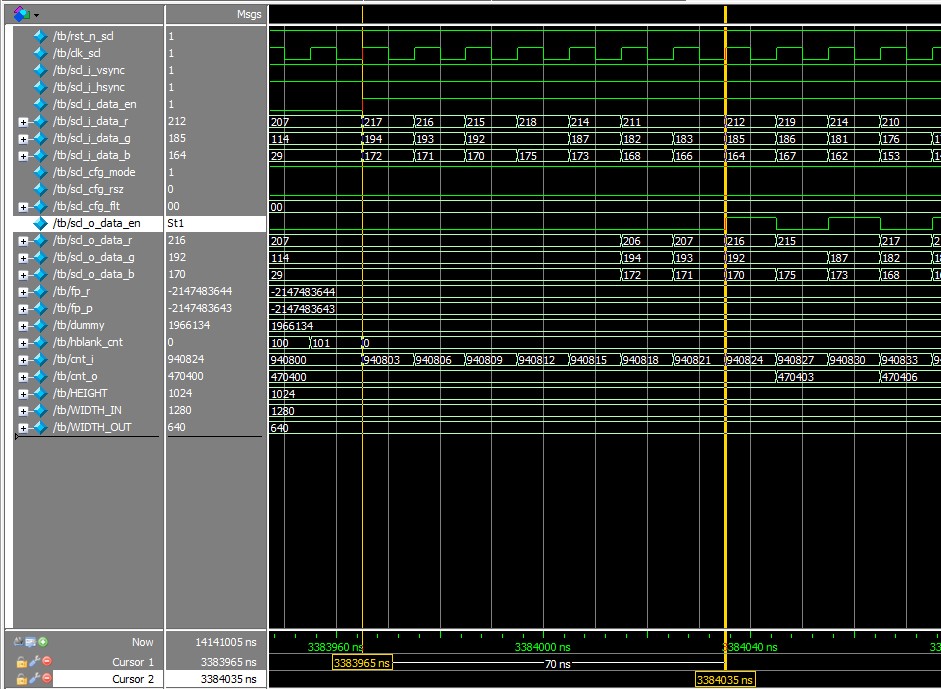
（3）1/4縮小



**图 4.7 1/4缩放输出**

如图 4.7所示，1/2缩放输出的scl\_o\_data\_en每隔3个时钟置1一次，将scl\_i\_data\_en波形分成了锯齿状输出，说明对图像做了1/4缩放。

**（4）システムディレイ**



**图 4.8 システムディレイ**

从图4.8的仿真波形可以看出，两个黄色光标之间的距离是70ns。因为时钟周期为10ns，70ns代表的是输入到输出的系统延迟是7个时钟周期，和设计所预期的系统延迟相同。

# 論理合成結果

**表5.1 Advanced HDL Synthesis Report**

|  |
| --- |
| # RAMs 1 |
| 4x44-bit single-port distributed Read Only RAM 1 |
| # Multipliers 16 |
| 11x9-bit multiplier 16 |
| # Adders/Subtractors 9 |
| 22-bit adder 6 |
| 23-bit adder 3 |
| # Counters 1 |
| 2-bit up counter 1 |
| # Registers 672 |
| Flip-Flops 672 |
| # Comparators 6 |
| 32-bit comparator greater 6 |
| # Multiplexers 19 |
| 1-bit 2-to-1 multiplexer 3 |
| 21-bit 2-to-1 multiplexer 7 |
| 8-bit 2-to-1 multiplexer 9 |

**表5.2 Final Register Report**

|  |
| --- |
| # Registers 540 |
| # Shift Registers 16 |
| 5-bit shift register 16 |

**表5.3 Timing Summary**

Speed Grade: -3

|  |
| --- |
| Minimum period 3.381ns |
| Maximum Frequency 295.769MHz |
| Minimum input arrival time before clock 1.047ns |
| Maximum output required time after clock 0.511ns |

# 設計環境

**表6.1环境使用介绍表**

|  |  |
| --- | --- |
| **Windows系统下软件名** | **版本** |
| Modesim | 10.1 |
| ISE Design Suite | 14.7 |
|  |  |

# 設計履歴

**7.1アーキ构造阶段**

**问题1**：图片读取问题

对图像缩放没有概念和不理解图像缩放算法的处理过程，导致将图片读入当成一个模块写在了设计中。

**问题2**：输出使能问题

アーキ构造阶段，误以为先产生scl\_o\_data\_en信号，然后用来控制像素的输出。

**解决办法：**

以上两个问题都是由于对图像缩放的处理过程理解不深入导致的。经过查找资料知道了实际上图片的读取和输出都是在tb中实现的，在tb中读入图片信息，依次将每个像素点的RGB值送入设计中做加权平均，输出经过处理的新的RGB值到tb中，最后由tb再生成新的图片。输出使能信号scl\_o\_data\_en的时序和输出的RGB值是一致的，这两个信号同时输入到tb中控制图片输出。实现过程如图所示。



**图7.1 图片的读取方法**

**7.2 coding阶段：**

**问题3**：忽略了滤波器模板存在负值，没有将端口定义为有符号型变量，端口定义错误。

**问题4**：对C++程序中的端处理概念不清，误以为寄存器中输入4个像素之后才开始做乘法。

**问题5**：tb文件中输入使能不知道如何给出。

**解决办法**：

将与滤波器有关的、存在负值的端口定义为有符号型；

因为做加权平均是以原点为基准，左边取一个点右边取两个点，在设计中实现的话，就是每次固定判断存放“原点”像素的寄存器ff2，因此当在第三个时钟周期的时候就开始了像素处理，接着做乘法，加法，加法一共需要6个时钟周期，这也是为何需要6个寄存器存放数据的原因；



**图7.2 像素处理起始点说明**

因为此次处理的图片不存在无效像素点，因此所读入的第一个像素点就是有效像素，所以在这里用两个计数器来实现输入有效波形。第一个计数器计到与行的宽度相同时翻转，作为输入有效，第二个计数器在每读完一行时开始计数，直到计到100时翻转，下一行开始有效。

**7.3验证阶段：**

**问题6:**只验证了正常功能的实现情况，未考虑到异常情况。

**问题7:**不知道用什么样的图片去验证滤波器。

**解决办法**：

**表7.1 验证项目**

|  |  |
| --- | --- |
| **機能検証** | **異常検証** |
| スルー動作 | mode是否受rsz和flt的异常影响 |
| 水平1/2間引き | 图片大小验证 |
| 水平1/4間引き | reset |
| フィルター |  |

如上表所示为最终的验证项目。

验证滤波器的目的是验证乘算和加算是否正常进行，一般用的是渐进图像。

# 更新履歴

**表8.1 履歴表**

|  |  |
| --- | --- |
| **时间** | **完成的内容** |
| 8.1---8.3 | アーキ構造 |
| 8.4---8.9 | RTLコード |
| 8.10---8.13 | 検証仕様 |
| 8.14---8.21 | RTL&報告書 |

# 9.残件

**表9.1残件項目**

|  |  |
| --- | --- |
| **項目** | **詳細** |
| 水平スケーラー拡張 | 水平N/255倍縮小(N=整数, N<=255) |
| 水平スケーラー拡張 | 水平2倍拡大 |
| 垂直スケーラー | 垂直2倍縮小 |