**HScaler設計報告書**

**Rivision: 0.01**

**作成者:雒旭鵬**

目录

[1. 設計概要 3](#_Toc460696954)

[2. I/F仕様 4](#_Toc460696955)

[3. デザイン仕様 7](#_Toc460696956)

[4. 検証結果 10](#_Toc460696957)

[5. 論理合成結果 15](#_Toc460696958)

[6. 設計環境 16](#_Toc460696959)

[7. 設計履歴 17](#_Toc460696960)

[8. 残件 18](#_Toc460696961)

[9. 更新履歴 19](#_Toc460696962)

# 設計概要

* 1. **設計の全体概要**

C++のテンプレートクラスを使って、チャネルを構造します。それから、自分のデザイン(DUT)とテストベンチ(tb)とはチャネルmy\_chanを通して、ハンドシェクで通信できます。「ハンドシェク」というのはinterfaceのきのうを実現できます。

機能概要は表1-1のように：

表1-1　機能概要

|  |  |
| --- | --- |
| プロジェクト | 詳細 |
| 入力メタポート | 1)busy(out)→get()が呼び出されるまでは0(受信不能)  2)ｖａｌｉｄ(in)  3)data(in) |
| 出力メタポート | 1)busy(in)  2)ｖａｌｉｄ(out)→vldが1になったら、busyが0になるまで0に下がらない(送信不能)  3)data(out) |

* 1. **最終実現機能**

実現タイミング図は図1-1のように：

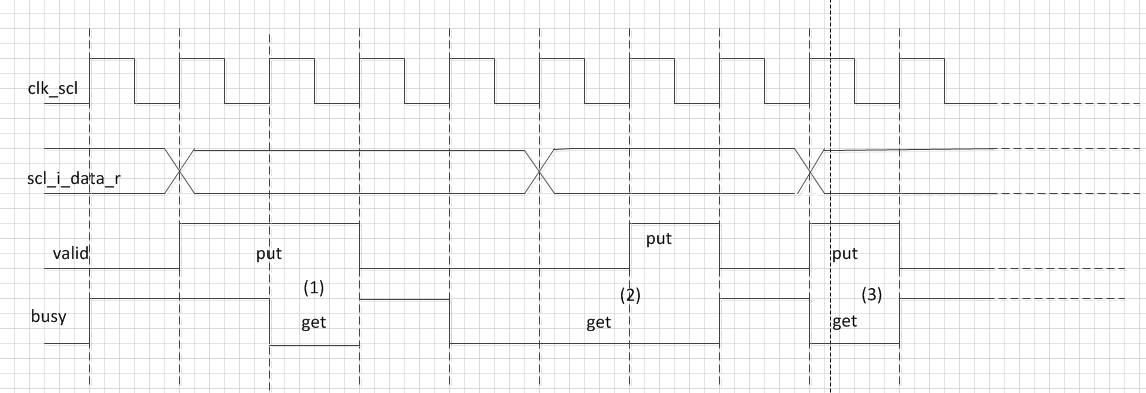


図1-1　タイミング図

　説明：

　このタイミングに場合が二つあります。ｖａｌｉｄ信号が有効の時、ｂｕｓｙ信号の有効を待っています。ほかの場合は、ｂｕｓｙ信号が有効の時、ｖａｌｉｄ信号の有効を待っています。最後、デートの送ると受信するの機能を実現できます。

* 1. **参考資料**

1. 桜井 至 善. SystemCを使ったハードウェア設計. CQ出版社. 2006.11.15
2. J.BHASKER著.夏宇闻 甘伟 译. SystemC入门(第2版). 北京航空航天大学出版社. 2008.09

# I/F仕様

* 1. **端子表**

端子は表2-1のように：

表2-1　my\_chan端子表

|  |  |  |  |
| --- | --- | --- | --- |
| 端子 | I/O | Bit Width | Role |
| In | my\_chan\_in | U8 | 入力データ |
| out | my\_chan\_out | U8 | 出力データ |
| data | siginal | U8 | channel |
| busy | siginal | U1 | channel |
| valid | siginal | U1 | channel |

* 1. **全体の入力出力モジュール図**

my\_chanは図2-1のように：

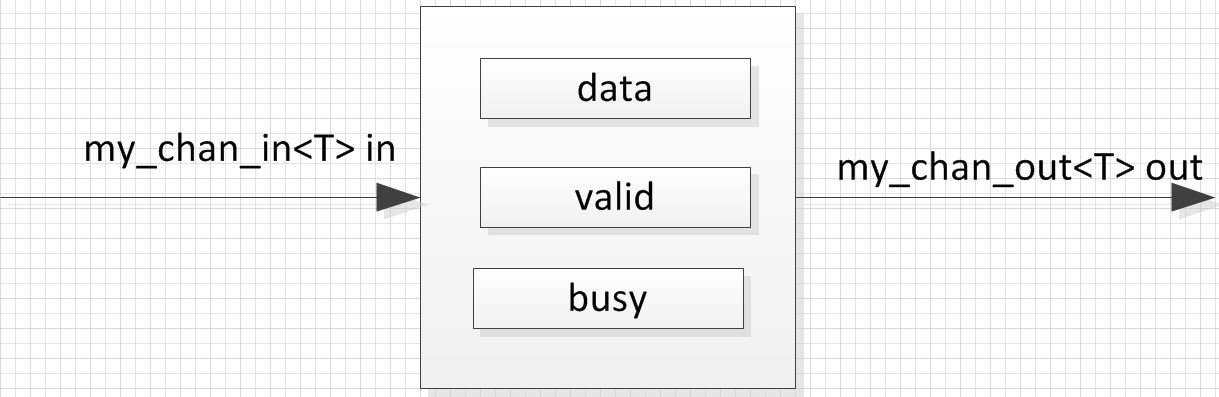


図2-1　 my\_chan

* 1. **設計分析**

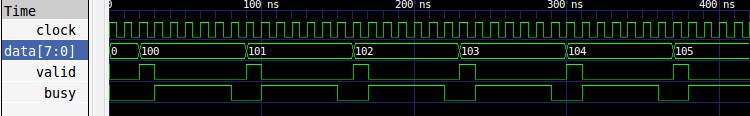
設計説明：

valid信号はハイレベルの時、データを送るのを始めて、「データをput」の動作を実行します。逆にこの信号はローレベルの時、データを送るのができません。

そして、busy信号はvalid信号と相反します。この信号はローレベルの時、データを受信するのを始めて、「データをget」の動作を実行します。逆に、この信号はハイレベルの時、いくらvalid信号がハイレベルでも、データを受信できません。他に、「データをget」の動作はクロックサイクルが一つしかかかりません。

最後二つ信号の間に以下のように関係があります。

二つ信号は同じなクロックサイクルに変わらなければならないです。以下の図に示します。



valid信号はハイレベルの時、一つクロックサイクルに101のようなデータを送る動作を完成します。同じな時間に101データを受信するの動作も完成します。それで、次のクロックが役に立つとき、二つ信号は同时に変わります。

* 1. **標準の入力出力のタイミング図**

実現タイミングは図2-2のように：

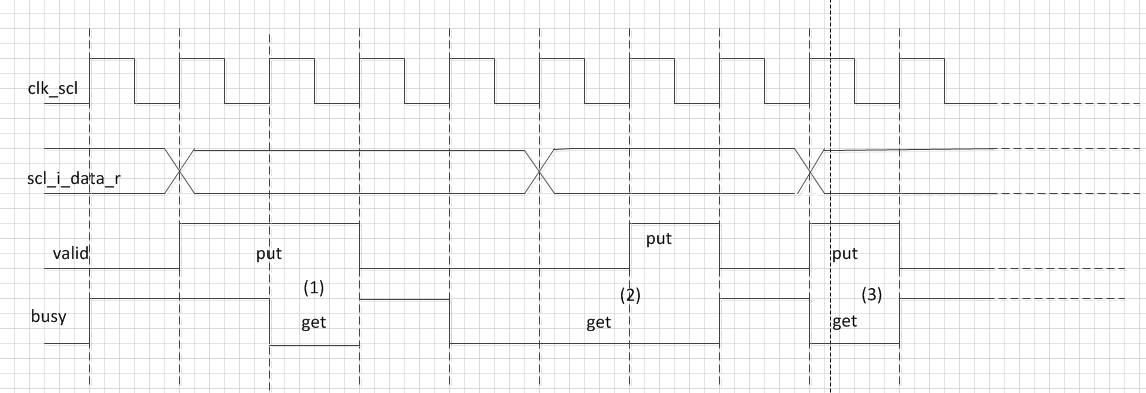


図2-2　タイミング

説明：

まず、このタイミングに特別な場合が二つあります。一つ目はbusy信号が忙し時、valid信号は持たなければなりません。逆にvalid信号が忙しい時、busy信号も待たなければなりません。それから、詳しく説明します。

一つクロックサイクルにデータを返します。しかし、必要な場合にとって、同じなデータを送ってもいいです。

# デザイン仕様

* 1. **最終デザインの回路図**

最終回路図3-1のように：



図3-1 最終回路

説明：

設計の考え方：まず一つピクセルの分量によって、アルゴリズム処理と加重平均の全部過程を完成します。それから、トップモジュールで一つピクセルの分量のinstanceを三回呼び出します。最後、トップモジュールでカウンタを使います。scl\_o\_data\_enをコントロールして、間引きの機能を実現します。

* 1. **ピクセルの单个分量を処理する詳細設計モジュール図**

ピクセル処理図3-2のように：

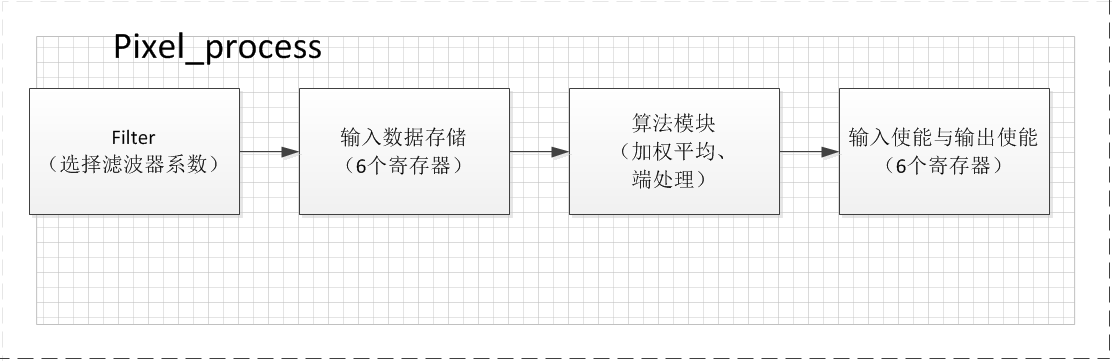


図3-2　ピクセル処理

説明：

このデザインで存在する難点は端処理の実現です。lanczosフィルターは4tapですから、このアルゴリズムはピクセルが四つあります。

処理しているピクセルによって前の一つピクセルと後の二つピクセルを選びます。それからフィルターの係数と別に相乗して、最後加重平均します。アルゴリズムで処理した結果は新しいピクセルです。

* 1. **端処理状況**

端処理は表3-1のように：

表3-1　端処理

|  |  |  |
| --- | --- | --- |
| ピクセル | 条件 | 方法 |
| 第一個のピクセル | o\_dff2==1 && o\_dff3==0 | re\_r２を２回計算する |
| 逆数第２個ピクセル | o\_dff0==0 && o\_dff1==1 | re\_r１を２回計算する |
| 逆数第１個ピクセル | o\_dff1==0 && o\_dff2==1 | re\_r２を３回計算する |

説明：

其中o\_dff0、o\_dff1、o\_dff2、o\_dff3は入力scl\_o\_data\_enについてのレジスタ番号。図3-3のように：



図3-3　 scl\_o\_data\_enレジスタ

re\_r0、re\_r1、re\_r2、re\_r3は入力データを出力の時、一つ符号ビットを拡張したレジスタ番号です。

プログラミングの実現過程：re\_r0 = {1'b0,in\_data\_r0}。図3-4のように：

図3-4　 scl\_i\_data\_enレジスタ

説明：

1. 第一個のピクセルによって入力scl\_i\_data\_en信号は低レベルから高いレベルまで数値になる時、一つ無効データが入力して、アルゴリズムでre\_r２レジスタを２回計算します。
2. 逆数第２個のピクセルによって入力scl\_i\_data\_en信号は高レベルから低レベルまで数値になる時、、一つ無効データが入力して、アルゴリズムでre\_r１レジスタを２回計算します。
3. 逆数第１個のピクセルによって入力scl\_i\_data\_en信号は高レベルから低レベルまで数値になる時、二つ無効データが入力して、アルゴリズムでre\_r２レジスタを３回計算します。
   1. **全体の内部設計図**

内部設計図3-5のように：

****

図3-5　内部設計

説明：

この設計図は内部モジュールによって詳しい実現方案を出します。そしてコネクトをします。端処理の実現は複雑なので、前の部分でもう詳しく説明していました。それで、この設計図は通常のピクセルによって処理している時、普遍の状況を書きました。

# 検証結果

* 1. **検証環境と方法**

全体図4-1のように：

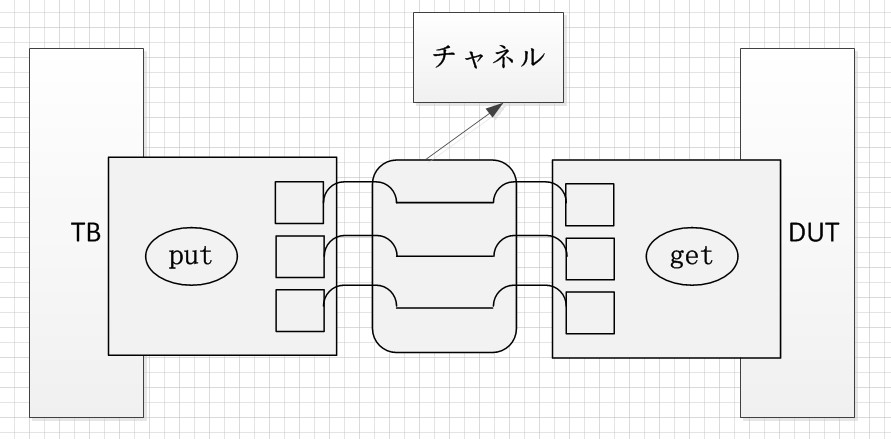


図4-1全体図

まずtbにデータをあげます。そして、tbの設計でチャネルmy\_chanを呼ばれたので、otmp変数を通して、putのセンテンスを使って、データはotmpに書き込みます。

他に、dutの設計でチャネルmy\_chanも呼ばれます。それで、同じな方法を使って、getのセンテンスを使って、データは読み出します。それから、itmp変数を通して、読み出したデータはitmpに入れます。

最後、main関数の中にmy\_chanチャネルを呼ばれて、inとout連接します。そうすると、チャネルを通して、データは送れます。interfaceの機能を実現できます。

* 1. **検証プロジェクト**

機能の検証表は表4-1のように：

表4-1検証

|  |  |
| --- | --- |
| 検証プロジェクト | 詳細 |
| busyはvalidを待つ | valid信号が忙しい |
| validはbusyを待つ | ｂusy信号が忙しい |

説明：

Validとbusy信号はどちらが先に役に立つか、確かめます。二つ場合を別れて、処理します。

* 1. **検証リスト**

機能の検証表は表4-2のように：

表4-2機能検証

|  |  |  |
| --- | --- | --- |
| Pattern name | データ | comment |
| 先にvalidが有効 | 100(1+＋) | busyはvalidを待つ |
| 先にbusyが有効 | 100(1+＋) | validはbusyを待つ |

説明：

このデザインに場合が二つあります。

一つ目は：busy信号が有効の前に、valid信号が有効します。正しいデータを受信するために、valid信号はbusy信号を待たなければなりません。busy信号がローレベルになると、一つクロックサイクルにデータを受信します。最後、同時に変わるとかそのままにします。

二つ目は： valid信号が有効の前に、busy信号が有効します。正しいデータを送るために、busy信号はvalid信号を待たなければなりません。valid信号がローレベルになると、一つクロックサイクルにデータを送ります。同時にデータを受信します。最後、同時に変わるとかそのままにします。

* 1. **検証結果**

タイミング検証図が以下です：

1. busyはvalidを待つタイミングは図4-3のように：

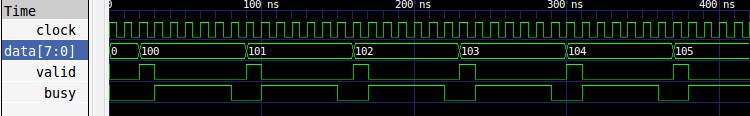


図4-3 busyはvalidを待つ

説明：

Valid信号が有効の前に、ｂｕｓｙ信号はもう有効しました。それで、ｂｕｓｙ信号はずっと待っています。ｖａｌｉｄ信号がハイレベルになる時、一つクロックサイクルにデータを送る動作を完成します。同時にｂｕｓｙ信号は一つクロックサイクルにデータを受信する動作を完成します。それから、次のクロックが役に立つ時、二つ信号は同時に変わります。最後Tbによって、システムディレイを確かめます。新しいデータを送るのを続けます。

1. validはbusyを待つタイミングは図4-4のように：

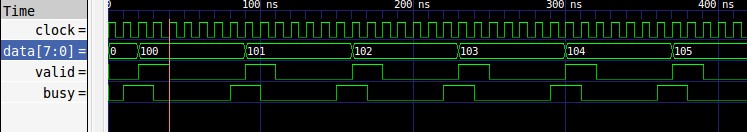


図4-4 validはbusyを待つ

説明：

ｂｕｓｙ信号が有効の前に、ｖａｌｉｄ信号はもう有効しました。それで、ｖａｌｉｄ信号はずっと待っています。ｂｕｓｙ信号がローレベルになる時、ｖａｌｉｄ信号は一つクロックサイクルにもうデータを送る動作を完成します。それでｂｕｓｙ信号も一つクロックサイクルにデータを受信する動作を完成します。最後、次のクロックが役に立つ時、二つ信号は同時に変わるとかそのままにします。他に、Tbによって、システムディレイを確かめます。新しいデータを送るのを続けます。

# 論理合成結果

* 1. **タイミングレポート**

タイミングレポートは表5-2のように：

表5-2 Timing Summary

|  |  |
| --- | --- |
| Minimum period | 3.392ns |
| Maximum Frequency | 294.806MHz |
| Minimum input arrival time before clock | 1.049ns |
| Maximum output required time after clock | 0.511ns |

# 設計環境

* 1. **設計環境の紹介**

このデザインはUbuntu14.04環境のしたでコードを書いて、systemc-2.3.1の環境で機能検証をしました。波形はgtkwaveツールを使います。

表6-1のように：

表6-1 環境紹介チャート

|  |  |
| --- | --- |
| ソフト | バージョン |
| Ubuntu | 14.04 |
| systemc | 2.3.1 |
| gtkwave |  |

# 設計履歴

* 1. **第一段階**

あった問題：SystemCを使って、hscalerの機能を実現した時、原図が出力できます。しかし、縮小したあとで、画像が出力できません。

解決方法：初めてSystemCの文法を勉強したので、たくさん知りませんでした。画像を縮小する時、VerilogとSystemCの文法を間違えてしまったので、以上の問題を出しました。最後、ずっとbugしたあとで、問題を知っていたあとで、この問題を解決しました。

* 1. **第二段階**

あった問題：初めてinterfaceを使ったので、理解できませんでした。

解决方法：同僚と一緒に相談しながら資料を調べて、も理解できませんでした。最後私たちは先輩に頼まれたあとで、先輩は私たちに授業を受けてくれました。だんだん理解できました。前の考えがめったに正しくなかったです。SystemCにとって今後の勉強でＣ++をよく分かるために、頑張ります。

# 残件

**8.1 実現ない機能**

機能拡張は表8-1のように：

表8-1 機能拡張

|  |  |
| --- | --- |
| 項目 | 詳細 |
| チャネルでhscalerを実現 | ｉｎｔｅｒｆａｃｅを使って、画像を縮小できる |

# 更新履歴

**9.1 設計仕様書の更新履歴**

更新履歴は表9-1のように：

表9-1 更新履歴

|  |  |
| --- | --- |
| 日付 | バージョン |
| 2016.9.25 | 初版 |
| 2016.9.29 | 修正版 |
| 2016.10.05 | 最終版 |