

# 「Chiselで始めるデジタル回路設計」 の翻訳作業について

Japanese translation work on  
"Digital Design with Chisel"

宗藤 誠治/日本IBM、Chisel勉強会  
RISC-V Day Tokyo 2020/11/5

# Digital Design with Chisel

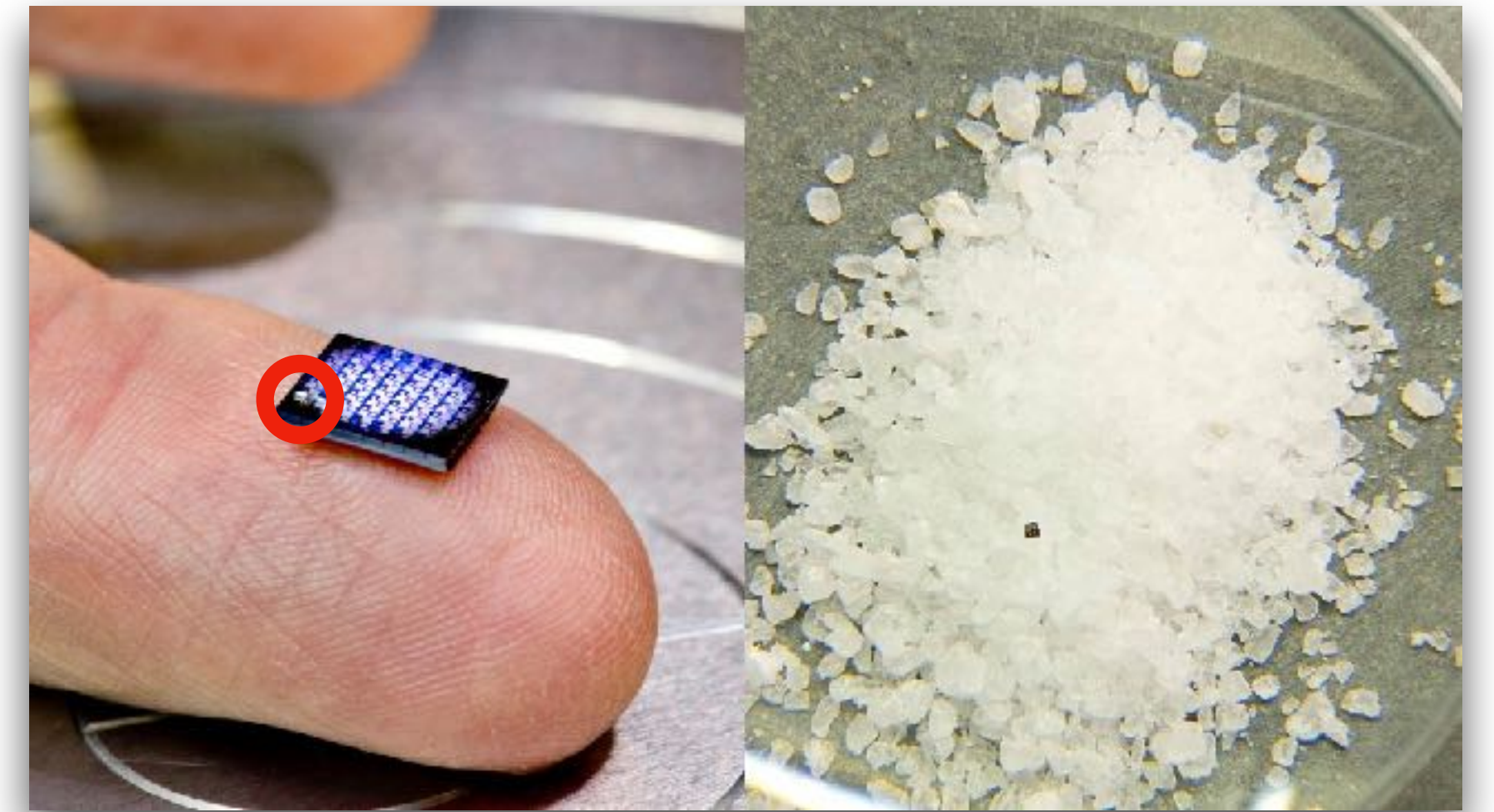
Chiselで始めるデジタル回路設計

Martin Schoeberl  
Chisel勉強会 訳



# 自己紹介

- 宗藤 誠治
  - 所属：日本IBM 東京基礎研究所
  - 業務：各種半導体の設計
  - RISC-V歴：2015年末から、PulpinoやFreedomをベースにエッジ向けの小型プロセッサを試作
- Chisel 勉強会（新型コロナウイルスの影響で休業中）
  - 登録URL <https://chisel-jp-slackin.herokuapp.com/>



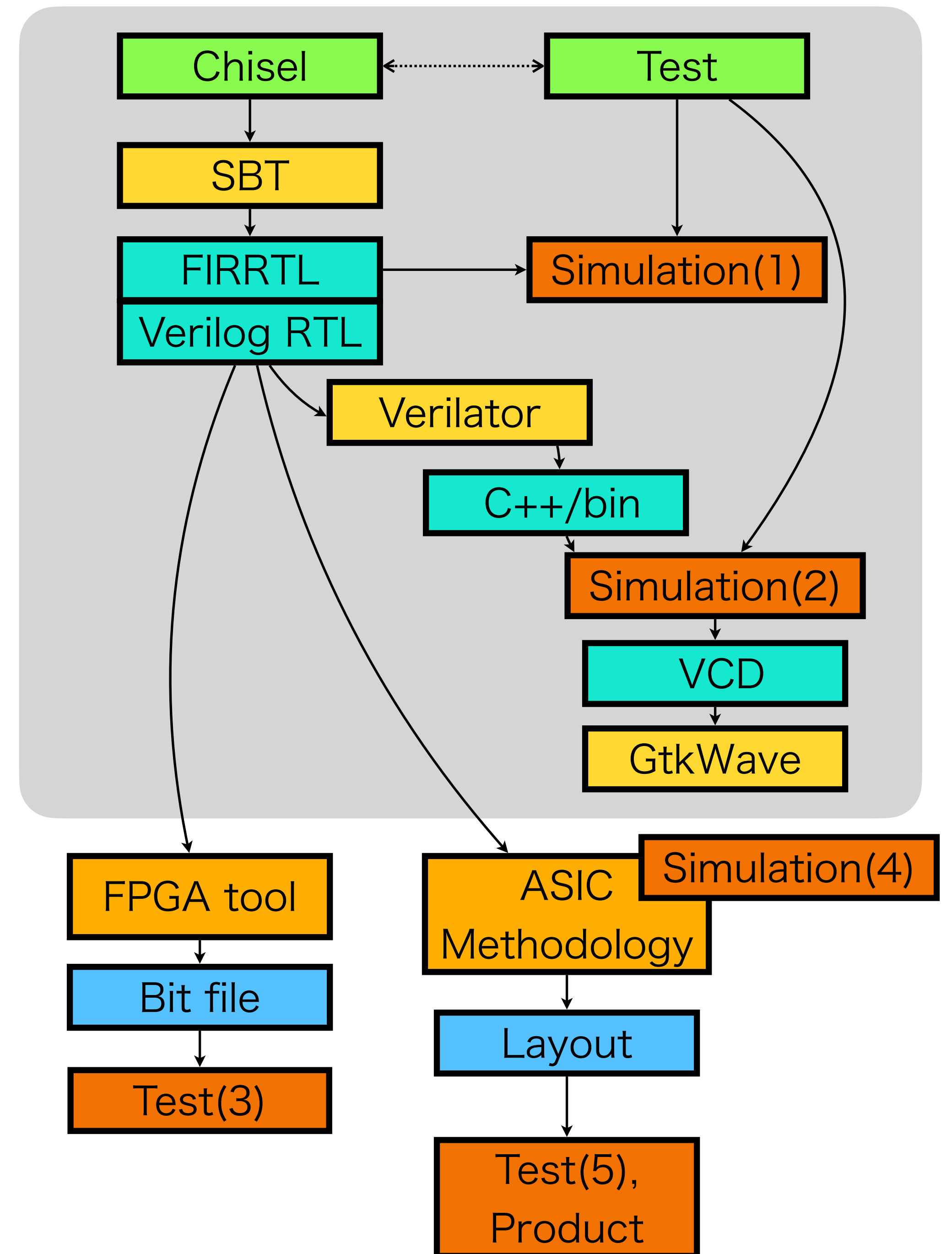
# Chiselはハードウェア「構築」言語

## VerilogやVHDLとの違い

- VerilogやVHDLは Hardware **Description** Language 「ハードウェア**記述**言語」
  - つまり、RTLやNetlistなどのハードウェアを記述するための言語
- ChiselはHardware **Construction** Language 「ハードウェア**構築**言語」
  - ハードウェア（Verilog RTL）を効率よく生成するための言語
  - Chiselはそのために設計されたScala(関数型プログラミング言語)ベースのドメイン特化言語(DSL)

# Chiselを使った開発フロー

- テスト駆動が(個人的には)おすすめ
- 基本的にRTLレベルの機能検証に必要なツールはすべてOSSで提供されるため、手元のPCで「いつでも」「どこでも」実行できる
- テスト実行は次の2つの形態がある
  - (1) Scalaコードの実行 (JVM上で)
  - (2) Verilogの実行 (Verilatorで)
    - 波形で動作確認
- 生成されたVerilog RTLは従来のFPGA(3)やASIC(4,5) のデザインフローで扱う



# Chiselの「壁」

## ハードウェア開発者の視点から

- ハードウェア開発者には馴染みのない世界
  - Scalaって何？ SBTって何？ Tilelinkって何？
  - なんでこのコードがこうなるの？ なんでこれで動いてるの？
- Chiselを使ったデジタル回路開発と、Chiselへの取り組みのレベル
  - とりあえず、FPGAやASICが動けばいい => Verilog RTL風なコーディングも可能です
  - Chiselを活用してもっと設計効率を高めたい => Rocket Core のようなコーディング
  - Chiselの機能を拡張する => Chisel自体を、、、



# Chiselの「入門」

## いきなりRocketはきつい

- Chisel boot camp
  - <https://github.com/freechipsproject/chisel-bootcamp>
- Chisel book
  - <https://github.com/schoeberl/chisel-book>
- 日本語の情報源
  - Blog: FPGA開発日記
    - <https://msyksphinz.hatenablog.com/>
  - 書籍：Chiselを始めた人に読んでほしい本
    - <https://diningyo.booth.pm/items/1718207>
  - 書籍：Chisel bookの日本語訳（翻訳中）
    - <https://github.com/chisel-jp/chisel-book/tree/japanese>



# Chisel book の「翻訳」

LaTeXのソースコードはすべてGithubで公開されています

- 本家 <https://github.com/schoeberl/chisel-book>
  - Creative Commons Attribution-ShareAlike 4.0 International License
- 日本語版 <https://github.com/chisel-jp/chisel-book/tree/japanese>
  - 本家の b20a791 (2020-03-02) がベース
  - 日本語版は japanese ブランチにあります
  - 日本語版のビルド方法は README\_jp.md に記載しています
    - ここでは Centos7, OSXについて記載していますが、UbuntuやWindowsでもビルド可能だと思います。



表示 - 継承 4.0 国際 (CC BY-SA 4.0)

これは人が読んでわかりやすいようにしたライセンスの要約です。(ライセンスの代わりになるものではありません。) [免責事項](#)

あなたは以下の条件に従う限り、自由に：



**共有** — どのようなメディアやフォーマットでも資料を複製したり、再配布できます。

**翻案** — マテリアルをリミックスしたり、変更したり、別の作品のベースにしたりできます。営利目的も含め、どのような目的でも。

あなたがライセンスの条件に従っている限り、許諾者がこれらの自由を取り消すことはできません。

あなたの従うべき条件は以下の通りです。



**表示** — あなたは [適切なクレジット](#) を表示し、ライセンスへのリンクを提供し、[変更があったらその旨を示さなければなりません](#)。これらは合理的であればどのような方法で行っても構いませんが、許諾者があなたやあなたの利用行為を支持していると示唆するような方法は除きます。



**継承** — もしあなたがこの資料をリミックスしたり、変更したり、加工した場合には、あなたはあなたの貢献部分を元の作品と [同じライセンス](#) の下に頒布しなければなりません。

**追加的な制約は課せません** — あなたは、このライセンスが他の者に許諾することを法的に制限するようないかなる法的規定も [技術的手段](#) も適用してはなりません。



# Chisel book の「翻訳」

## 翻訳しながら勉強しよう

- 機械翻訳
  - LaTeXは苦手
- 手動校正で手直し（今ここ）

```
1765 \ifshoworiginal
1766 {\bf Possible pitfall:} One possible error when defining constants with a dedicated width is missing the \code{.W}
1767 specifier for a width. E.g., \code{1.U(32)} will \emph{not} define a 32-bit wide constant representing 1.
1768 Instead, the expression \code{(32)} is interpreted as bit extraction from position 32, which results
1769 in a single bit constant of 0. Probably not what the original intention of the programmer was.
1770 \fi
1771
1772 \ifshowtransfirst %(自動翻訳)
1773 {\ \ BF可能な落とし穴：}一つの可能なエラー専用幅の定義定数は幅に対して\code{.W}指定子を欠落しています。
1774 え。
1775 グラム。
1776 、\code{1.U(32)}は\emph{not}は1を表す32ビット幅の定数を定義します。
1777 その代わりに、発現\code{(32)} 0の単一ビット一定になる位置32からビット抽出、と解釈されます。
1778 おそらくプログラマの本来の意図は何だったのではありません。
1779 \fi
1780
1781 \ifshowtranssecond %(校正 4/21 diningyo), BF修正5/4 mune10
1782 {\bf ハマリやすい落とし穴：}
1783 定数の宣言時に起こりがちなエラーとして、ビット幅指定のための\code{.W}を忘れることがあげられます。
1784 例えば\code{1.U(32)}のような表現は32bitの1を表す定義ではありません。
1785 その代わりに\code{(32)}は3 2 bit目のビットの指定として解釈されるため、結果的に1bitの0になります。
1786 これはおそらくプログラマが本来意図したものではないはずです。
1787 \fi
```

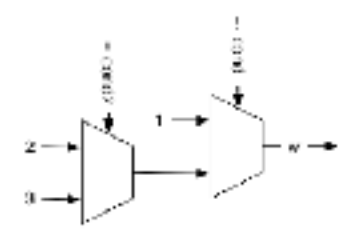
ハマリやすい落とし穴：定数の宣言時に起こりがちなエラーとして、ビット幅指定のための.Wを忘れることがあげられます。例えば1.U(32のような表現は32bitの1を表す定義ではありません。その代わりに(32)は3 2 bit目のビットの指定として解釈されるため、結果的に1bitの0になります。これはおそらくプログラマが本来意図したものではないはずです。



# Chisel book の「翻訳」 Overleaf

- クラウド上のLatex環境
- Latexの編集、ビルドが可能（日本語もOK）
- 共同(同時)編集が可能

The screenshot displays the Overleaf editor interface for the Chisel book. The left sidebar shows the file structure, including folders for 'chisel2', 'code', 'figures', 'shared', and 'src', and a file outline for 'chisel-book-jp.tex'. The main editor area is split into three panes: 'Source' (Chisel code), 'Rich Text' (Japanese translation), and 'Recompile' (PDF output). The 'Rich Text' pane shows a chapter titled '5 組合せ回路ブロック (L5130 mune10/diningyo 初回校正済)' with a detailed Japanese translation of the text. The 'Recompile' pane shows the resulting PDF output, including a diagram of a chain of multipliers (Figure 5.1) and the corresponding Chisel code for the multiplier chain.

```
5 (L5130 mune10/diningyo)

Figure 5.1: A chain of multipliers.

Chiselは演算した条件分岐 (if/else/when) である。alwaysなサポートしていません。
val w = Wire(UInt(1))
when (cond) {
  w := 1.0
}
```

# Chisel book の「翻訳」

## 翻訳状況

- ・ 「はじめに」から5章までと、13章以降の機械翻訳の校正が終了
- ・ その他
  - ・ 日本語の情報元の追加(完了)
  - ・ A4対応
  - ・ 日本語インデックスの追加（作業中）

章	機械翻訳	校正初回	コメント
はじめに	完了	完了	
序文	完了	完了	
1章	完了	完了	
2章	完了	完了	
3章	完了	完了	
4章	完了	完了	
5章	完了	完了	
6章	完了		
7章	完了		
8章	完了		
9章	完了		
10章	完了		
11章	完了		
12章	完了		
13章	完了	完了	
A	完了	完了	
B	完了	完了	
C	完了	完了	



# Chisel book の「翻訳」

## ボランティア募集中

- ・ すみません、まだ初回校正作業中です
- ・ 章単位でボランティア募集中
- ・ 全体の校正作業も、
  - ・ 読んでみて気になる箇所（内容でも訳でも）あればご連絡ください
- ・ 貢献方法
  - ・ GithubでのPull Request
  - ・ Overleafでの直接編集
  - ・ Slackでのコメント
  - ・ などなど、気軽にコンタクトください。この本もOpenです。

# Chisel book の「翻訳」 予定

- 2020年内に
  - 初回校正完了
  - 本家の更新分を取り込み
  - Chisel Book V2 日本語版完成
- V3?

**Chisel沼にようこそ**