報道各位

2020 年 8 月 28 日
株式会社インプレスR&D
https://nextpublishing.jp/

RISC-V ハードウェア構築言語 Chisel が手を動かしてわかる！

『Chisel を始めたい人に読んで欲しい本』発行

技術の泉シリーズ、8 月の新刊

インプレスグループで電子出版事業を手がける株式会社インプレス R&D は、『Chisel を始めたい人に読んで欲しい本』(著者:七夕 雅俊)を発行いたしました。

最新の知見を発信する『技術の泉シリーズ』は、「技術書典」や「技術書同人誌博覧会」をはじめとした各種即売会や、勉強会・LT 会などで頒布された技術同人誌を底本とした商業書籍を刊行し、技術同人誌の普及と発展に貢献することを目指します。

『Chiselを始めたい人に読んで欲しい本』
https://nextpublishing.jp/isbn/9784844378693

著者:七夕 雅俊
小売希望価格:電子書籍版 2000 円(税別)／印刷書籍版 2500 円(税別)
電子書籍版フォーマット:EPUB3／Kindle Format8
印刷書籍版仕様:B5 判／カラー／本文 234 ページ
ISBN:978-4- 8443-7869-3
発行:インプレス R&D

＜＜発行主旨・内容紹介＞＞

RISC-V 実装の一つである Rocket Chip や BOOM と言った CPU の名前と共に、チラホラ目に入る「Chisel」の文字。調べてみたけど資料も少なくて、どういう言語かわからない…という方に向けた書籍です。

インストールに始まり、学ぶ上で避けて通れない Scala の文法や基本的な Chisel の文法、Chisel の力が発揮されるテストとパラメタライズの方法までを 1 冊にまとめています。

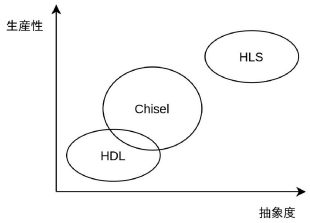
(本書は、次世代出版メソッド「NextPublishing」を使用し、出版されています。)

Chisel の基本的な情報や環境構築を OS ごとに解説

生産性という面で、HLS（High Level Synthesis）を思い浮かべた方もいると思います。ではChiselはHLSなのか？というと、HLSとは明確に異なる、というのが著者の見解です。HLSは、実現したいアルゴリズムを記述し、ハードウェアに変換しますが、Chiselにはこのような機能は備わっていません。

その代わりに、ChiselではHDLと同等の記述をパラメタライズして抽象度を上げ、生産性を向上させることができます。これらのことを踏まえて、Chiselのポジションを図示すると、図1.1のようになるでしょうか。

図1.1: 著者の考えるChiselのポジション



だいたい、HDLとHLSの中間あたりに存在する、とイメージしてもらえるとよいかと思います。図ではどちらかといえば、HDLよりに配置しました。HLSは、実現したいアルゴリズムを回路に落としこむという、トップダウン的なイメージです。いっぽうChiselは、HDL起点でもっと柔軟に回路を構成するために機能を追加した、ボトムアップ的なアプローチであるように感じます。Chiselでは基本的な文法を使うと、HDLと等価なデザインを作ることができます。Chiselの持つパラメタライズの機能をフルに使うことで抽象度を高め、生産性を向上することが可能です。

1.3 簡単なサンプル

Scalaや関数型言語やら、耳馴染みのない言葉と思われた方もいるかもしれません。ですが、実際に設計するのはハードウェアです。まずはどんな言語なのか、簡単なサンプルで確認してみましょう。ソースの細かい部分は置いておいて、普段ご自身が設計に使用している言語と比較しながら、Chiselで記述するとどんな風に書けるのかを、見てみてください。

1.3.1 Chiselで実装する簡単なFIFO

例題にするのは、ハードウェア設計では皆さんお馴染みのFIFOです。

リスト1.1: src/main/scala/chapter1/FIFO.scala

```
import chisel3._
import chisel3.util._

/**
  * FIFO リード側 I/O
  */
class FIFORdIO(bits: Int) extends Bundle {
  val enable = Input(Bool())
  val empty = Output(Bool())
  val data = Output(UInt(bits.W))
}

/**
  * FIFO ライト側 I/O
  */
class FIFOWrIO(bits: Int) extends Bundle {
  val enable = Input(Bool())
  val full = Output(Bool())
  val data = Input(UInt(bits.W))
}

/**
  * FIFO I/O
  * @param bits データのビット幅
  * @param depth FIFOの段数
  * @param debug trueでデバッグモード
  */
class FIFOIO(bits: Int, depth: Int = 16, debug: Boolean = false)
  extends Bundle {

  val depthBits = log2Ceil(depth)

  val wr = new FIFOWrIO(bits)
  val rd = new FIFORdIO(bits)

  val dbg = if (debug) { Some(Output(new Bundle {
    val r_wrptr = Output(UInt(depthBits.W))
    val r_rdptr = Output(UInt(depthBits.W))
    val r_data_ctr = Output(UInt((depthBits + 1).W))
  })) } else {
```


Scala の文法と基本的な Chisel の文法を解説

第3章　Scalaの基本

第1章に記載したとおり、ChiselはScalaの内部DSLとして実装されています。そのため、Chiselでハードウェアを実装するにあたり、Scalaの文法をある程度知っておく必要があります。

本章では、Chiselでハードウェアの設計を行うにあたり、必要な基本的文法を解説します。

3.1 Hello, World

まずは様式美ということで、Scalaで"Hello, World"を書くとどうなるのか、見てみましょう。

リスト3.1: Scalaの"Hello, World"：src/main/scala/chapter3/HelloWorld.scala:L5-L10

```
/**
 * 普通のメイン関数
 */
object HelloWorld {
  def main(args: Array[String]): Unit = println("Hello, World!!")
}
```

ご覧いただいたように、Scalaのメイン関数は、`object`で囲った空間の中に`def main`を宣言する形になります。

これを先ほどインストールしたsbtを使って、実行してみましょう。sbtを使ってScalaのファイルを実行する場合は、sbtからScalaのファイルを認識させるために、リスト3.2のようなディレクトリー構造を作成し、ソースファイルを配置する必要があります。

リスト3.2: sbtプロジェクトのソースコードの構成

```
./<sbtプロジェクトのルートディレクトリー>
└── src
    └── main
        └── scala # この下にScalaのソースコードを格納する
            └── HelloWorld.scala
```

リスト3.2のように、<sbtプロジェクトのルートディレクトリー>の指定の場所に"HelloWorld.scala"を配置したら、sbtコマンドを引数なしで実行してください。

リスト3.3: sbtの起動

```
$ sbt
[info] [launcher] getting org.scala-sbt sbt 1.3.10  (this may take some time)...
# ダウンロードは初回起動時のみ
downloading https://repo1.maven.org/maven2/org/scala-sbt/sbt/1.3.10/sbt-1.3.10.jar
-中略-
[warn] No sbt.version set in project/build.properties, base directory:
/scala_chk
[info] Loading global plugins from /home/diningyo/.sbt/1.0/plugins
[info] Set current project to scala_chk (in build file:/chisel-template-release/)
[info] sbt server started at
 local:///home/diningyo/.sbt/1.0/server/5c275feaef900662f851/sock
sbt:scala_chk>
```

初回起動時のみ、依存ライブラリーのダウンロードが自動で実行された後に`sbt:[ルートディレクトリー名]>`のような形で、sbtシェルが起動します。シェルが起動したら、リスト3.4のコマンドを実行してください。

リスト3.4: sbtを使ったメイン関数の実行

```
sbt:scala_chk> runMain main
[info] running Main
Hello, World!!
[success] Total time: 0 s, completed 2020/05/23 23:13:17
```

これで、sbtを使ったScalaファイルの実行ができるようになりました。お気づきとは思いますが、第1章で使用したchisel-templateは、sbtの機能でChiselの実行に必要な依存ライブラリーをダウンロードし、実行する環境を提供してくれるテンプレートとなっています。

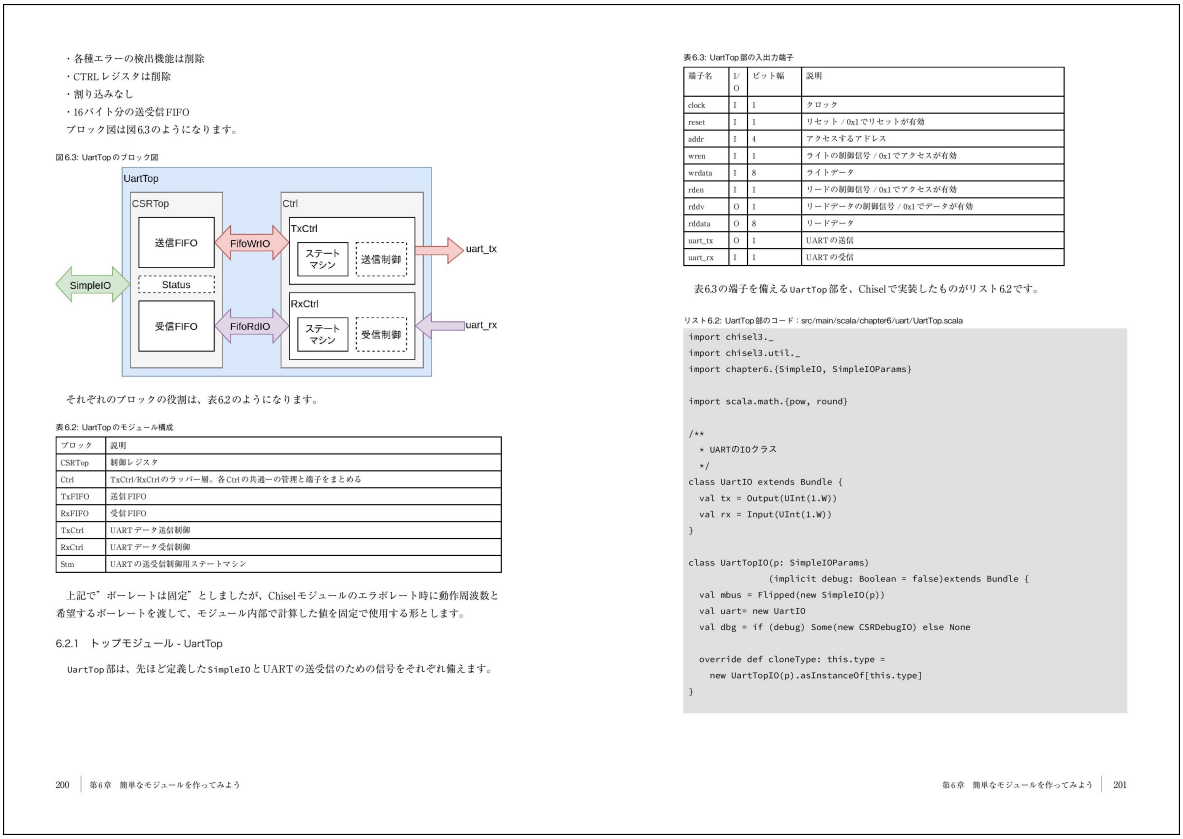
3.1.1 sbtを使ったプロジェクト

ここでsbtのプロジェクトの構成と、よく使用するコマンドについて簡単に解説します。まずプロジェクトのディレクトリー構成は、リスト3.5のようになります。

リスト3.5: sbtを使ったプロジェクトの構成

```
./プロジェクトのルートディレクトリー
  |
  ├── project : ビルドサポートファイル用ディレクトリー（Chiselでは気にしなくてOK）
  ├── src/ : ソースコードを格納するディレクトリー
  |    |
  |    ├── main
  |    |    |
  |    |    ├── resources : メインに含むデータ（ChiselではRTLを置いたりする）
  |    |    └── scala : メインのScalaソースファイル
  |    |
  |    └── test
  |         |
```

Chisel をもっと便利につかうための情報を紹介



・各種エラーの検出機能は削除
・CTRL レジスタは削除
・割り込みなし
・16バイト分の送受信FIFO
ブロック図は図6.3のようになります。

図6.3: UartTopのブロック図

それぞれのブロックの役割は、表6.2のようになります。

表6.2: UartTopのモジュール構成

ブロック	説明
CSRTop	制御レジスタ
Ctrl	TxCtrl/RxCtrlのラッパー層。各Ctrlの共通一の管理と端子をまとめる
TxFIFO	送信FIFO
RxFIFO	受信FIFO
TxCtrl	UARTデータ送信制御
RxCtrl	UARTデータ受信制御
Stm	UARTの送受信制御用ステートマシン

上記で”ボーレートは固定”としましたが、Chiselモジュールのエラボレート時に動作周波数と希望するボーレートを渡して、モジュール内部で計算した値を固定で使用する形とします。

6.2.1 トップモジュール - UartTop

UartTop部は、先ほど定義したSimpleIOとUARTの送受信のための信号をそれぞれ備えます。

200 | 第6章 簡単なモジュールを作ってみよう

表6.3: UartTop部の入出力端子

端子名	I/O	ビット幅	説明
clock	I	1	クロック
reset	I	1	リセット / 0x1でリセットが有効
addr	I	4	アクセスするアドレス
wren	I	1	ライトの制御信号 / 0x1でアクセスが有効
wrdata	I	8	ライトデータ
rden	I	1	リードの制御信号 / 0x1でアクセスが有効
rddv	O	1	リードデータの制御信号 / 0x1 でデータが有効
rddata	O	8	リードデータ
uart_tx	O	1	UARTの送信
uart_rx	I	1	UARTの受信

表6.3の端子を備えるUartTop部を、Chiselで実装したものがリスト6.2です。

リスト6.2: UartTop部のコード：src/main/scala/chapter6/uart/UartTop.scala

```
import chisel3._
import chisel3.util._
import chapter6.{SimpleIO, SimpleIOParams}

import scala.math.{pow, round}

/**
  * UARTのIOクラス
  */
class UartIO extends Bundle {
  val tx = Output(UInt(1.W))
  val rx = Input(UInt(1.W))
}

class UartTopIO(p: SimpleIOParams)
               (implicit debug: Boolean = false)extends Bundle {
  val mbus = Flipped(new SimpleIO(p))
  val uart= new UartIO
  val dbg = if (debug) Some(new CSRDebugIO) else None

  override def cloneType: this.type =
    new UartTopIO(p).asInstanceOf[this.type]
}
```

第6章 簡単なモジュールを作ってみよう | 201

＜＜目次＞＞

＜＜著者紹介＞＞

七夕 雅俊

関東圏在住のエンジニア。面白そうな言語みつけた！とChiselを調べ始め、ブログの記事などを作成している。
もっと多くの方にChiselを知って欲しく、日々布教活動中。

＜＜販売ストア＞＞

電子書籍:

Amazon Kindle ストア、楽天 kobo イーブックストア、Apple Books、紀伊國屋書店 Kinoppy、Google Play Store、honto 電子書籍ストア、Sony Reader Store、BookLive!、BOOK☆WALKER

印刷書籍:

Amazon.co.jp、三省堂書店オンデマンド、honto ネットストア、楽天ブックス

※ 各ストアでの販売は準備が整いしだい開始されます。
※ 全国の一般書店からもご注文いただけます。

【インプレス R&D】 https://nextpublishing.jp/

株式会社インプレスR&D（本社：東京都千代田区、代表取締役社長：井芹昌信）は、デジタルファーストの次世代型電子出版プラットフォーム「NextPublishing」を運営する企業です。また自らも、NextPublishing を使った「インターネット白書」の出版など IT 関連メディア事業を展開しています。

※NextPublishing は、インプレス R&D が開発した電子出版プラットフォーム（またはメソッド）の名称です。電子書籍と印刷書籍の同時制作、プリント・オンデマンド（POD）による品切れ解消などの伝統的出版の課題を解決しています。これにより、伝統的出版では経済的に困難な多品種少部数の出版を可能にし、優秀な個人や組織が持つ多様な知の流通を目指しています。

【インプレスグループ】 https://www.impressholdings.com/

株式会社インプレスホールディングス（本社：東京都千代田区、代表取締役：松本大輔、証券コード：東証 1 部 9479）を持株会社とするメディアグループ。「IT」「音楽」「デザイン」「山岳・自然」「モバイルサービス」「学術・理工学」「旅・鉄道」を主要テーマに専門性の高いメディア&サービスおよびソリューション事業を展開しています。さらに、コンテンツビジネスのプラットフォーム開発・運営も手がけています。

【お問い合わせ先】

株式会社インプレス R&D NextPublishing センター
TEL 03-6837-4820
電子メール: np-info@impress.co.jp