

(Chiselを使って)FPGAで動く何かを作った話

ゆるい話をします

Scope

Chiselの紹介と、使ってみた所感についてお話しします。

- FPGAに興味があるかもしれない人
 - 何ができるか多めにしました
- Verilog HDLやVHDLを使ったことがある人
- Chiselを使ってなにか作ってみたい人

Agenda

- 前置き
 - FPGAとは
 - 開発について
- Chiselについて
- 作った話
- まとめ

What is an FPGA?

Field Programmable Gate Arrayの略。論理回路を書き換え可能なLSI

What is an FPGA?

Field Programmable Gate Arrayの略。論理回路を書き換え可能なLSI

何がすごい

- 書き換え可能
 - ASICだとMask変更 \$
- 回路を作れる
 - CPUではできない並列/高速処理 😊
- 個人で使える(最高 ❤️)
 - 安いものなら秋月電子でも600円から

The screenshot shows a product page from the Akihabara Denki website. The top navigation bar includes links for 'マイページ', '注文履歴', 'お問い合わせ', 'かごの中身', 'トラック広告', and '回路図集'. Below the navigation is a breadcrumb trail: 'トップ > 半導体 > CPLD・FPGA > FPGA(MachXO2) LCMXO2-256HC-4TG100C'. The main content features a large image of the Lattice chip, which is a black package with a metal lead frame (MLF) and the text 'LATTICE' and 'LCMXO2-256HC-4TG100C-A6241R70' printed on it. Below the chip are five smaller images of the chip at different angles. To the right of the chip, there is detailed product information:

FPGA (MachXO2) LCMXO2-256HC-4TG100C
[LCMXO2-256HC-4TG100C]
通販コード I-11003
発売日 2016/09/06
メーカーカテゴリ [Lattice Semiconductor Corporation](#)

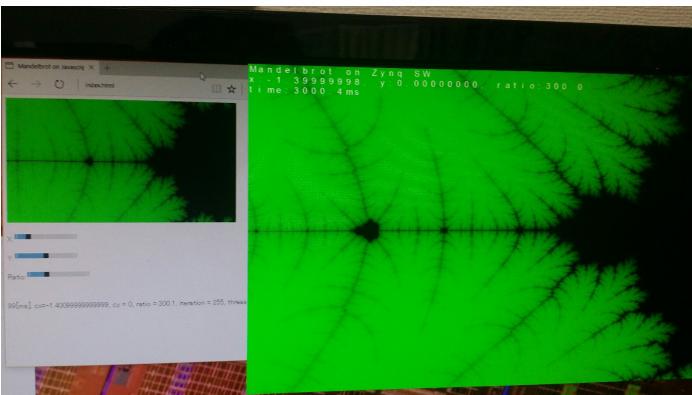
Lattice社のFPGA(フィールドプログラマブルゲートアレイ)です。3.3V单っています。外部部品が少なく手軽にFPGAを動かすことが可能です。また、[07225](#)などを御利用頂けます。

■主な仕様
ファミリ:MachXO2
ランク:HC高性能版
スピード:-4
LAB数:32
LUT数:256
I/O数:55
分配RAM:2Kbit
ハードウェアマクロ:I2C x2
ハードウェアマクロ:SPI x1
ハードウェアマクロ:タイマーx1
電源電圧:2.375v~3.465v

[この商品を友達に教える](#)
[お気に入りに追加する](#)
[店舗情報](#)

どんな物が作れる

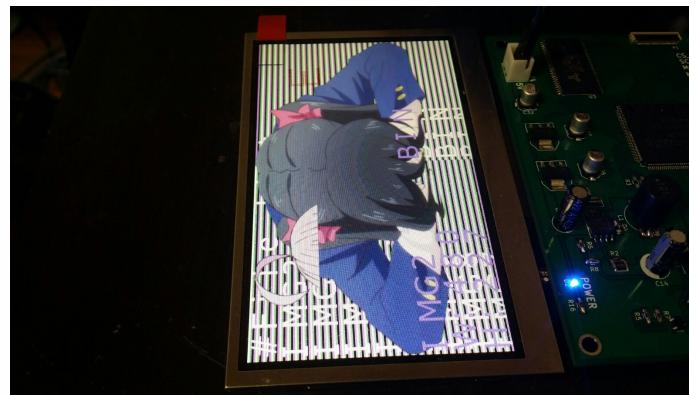
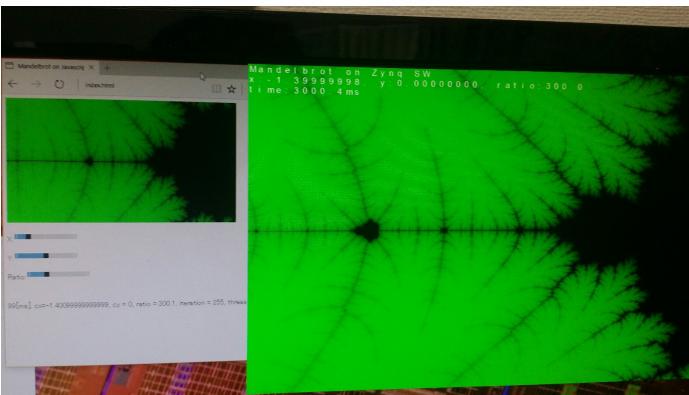
計算アクセラレータ



- Mandelbrot計算専用HW
- HDMI出力も自作

どんな物が作れる

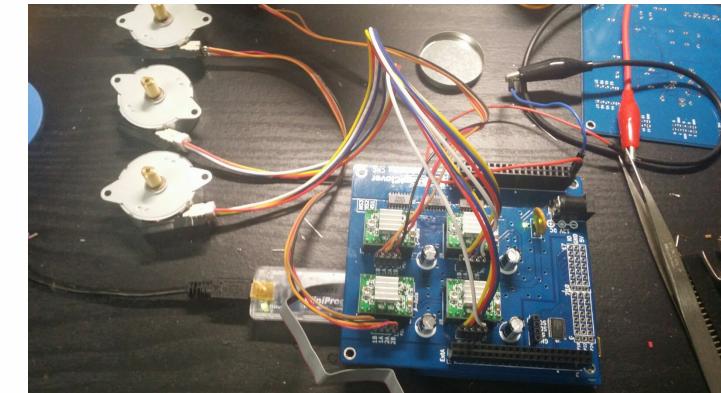
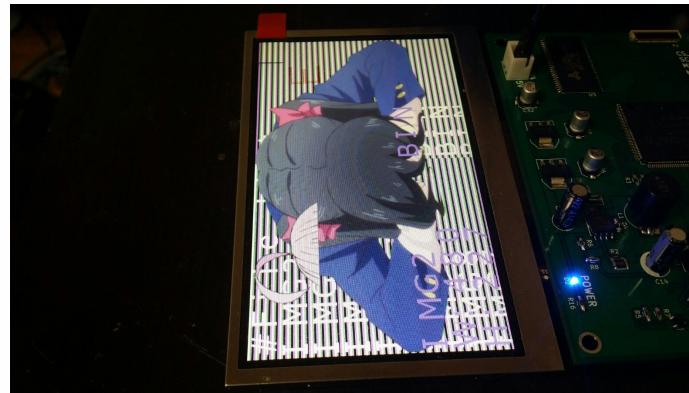
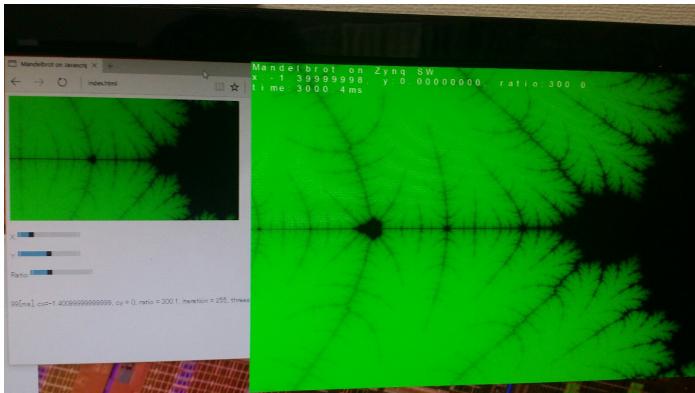
計算アクセラレータ コントローラ(はやい)



- Mandelbrot計算専用HW
- HDMI出力も自作
- SDRAM->LCD描画HW
- DMAみたいな

どんな物が作れる

計算アクセラレータ コントローラ(はやい) 汎用ペリフェラル



- Mandelbrot計算専用HW
- HDMI出力も自作
- SDRAM->LCD描画HW
- DMAみたいな
- Stepper Moter HW×4基
- LEが許す限り増やせる

作り方

- 方式検討
 - アイデアとかこねこねする

作り方

- 方式検討
 - アイデアとかこねこねする
- 実装
 - RTL(レジスタ転送レベル)で設計。 HDLで記述
 - 流行ってるCとかC++などから変換するのは高位合成と呼ばれている

作り方

- 方式検討
 - アイデアとかこねこねする
- 実装
 - RTL(レジスタ転送レベル)で設計。 HDLで記述
 - 流行ってるCとかC++などから変換するのは高位合成と呼ばれている
- 論理テスト
 - 正しく動くか試す。 HDLで記述するかC++に変換してからやったりとか

作り方

- 方式検討
 - アイデアとかこねこねする
- 実装
 - RTL(レジスタ転送レベル)で設計。 HDLで記述
 - 流行ってるCとかC++などから変換するのは高位合成と呼ばれている
- 論理テスト
 - 正しく動くか試す。 HDLで記述するかC++に変換してからやったりとか
- 実機テスト
 - 動けば神

HDL(ハードウェア記述言語)

```
module divider(
    input clk, input rstn, output sig
);
reg [7:0] counter;
assign sig = counter[7];
always @ (posedge clk) begin
    if (rstn != 1'b1) begin
        counter <= 8'h0;
    end else begin
        counter <= counter + 8'h1;
    end
end
endmodule
```

分周器

- 周波数を $1/(2^n)$ して出力
- Verilog HDLで書いた例

記述のつらみ

VHDL, Verilog HDL, SystemVerilogが主流

- プログラミング言語ではない
 - 透けて見えるのはアセンブラーではなくフリップフロップ
 - テストを書くのが大変になりがち
- 記述の規格が曖昧
 - VHDLは厳格で冗長すぎるし、Verilogはガバガバ(型チェックとか)
 - オブジェクト指向とは🤔

※個人の感想です

Chiselとは

ChiselはScalaでHDLが書ける(組み込みDSL)。Verilog HDLにTranslateできる。

Edge TPU



RISC-V実装

A screenshot of a GitHub repository page for 'UC Berkeley Architecture Research'. The header includes a search bar, 'Pull requests', 'Issues', 'Marketplace', and 'Explore'. Below the header is the repository logo, which is a yellow bear icon, and the repository name 'UC Berkeley Architecture Research'. It shows location 'Berkeley, CA', URL 'http://bar.eecs.berkeley....', and email 'info@riscv.org'. Below this are tabs for 'Repositories 70', 'People 10', and 'Projects 0'. A section titled 'Pinned repositories' lists three repositories: 'chisel-tutorial', 'chisel-testers2', and 'dsptools'. Each pinned repository card includes a thumbnail, the repository name, a brief description, and statistics: 'chisel-tutorial' has 257 stars and 117 forks; 'chisel-testers2' has 6 stars and 1 fork; and 'dsptools' has 58 stars.

Chiselの構文

```
class Counter(width: UInt) {
    val io = IO(new Bundle {
        val dout = Output(Bool())
    })
    val counter = RegInit(UInt(width.U), 0.U)
    io.dout <> counter(7)
    counter := counter + 1.U
}
```

分周器

- scalaの値がマクロになる
 - widthとかifとか
- 値の集合をBundleで定義
 - 共通化できる
- 同期回路前提
- Scalaのままテスト可

つくったもの

Brainf**k言語処理系 on FPGA

BF処理系

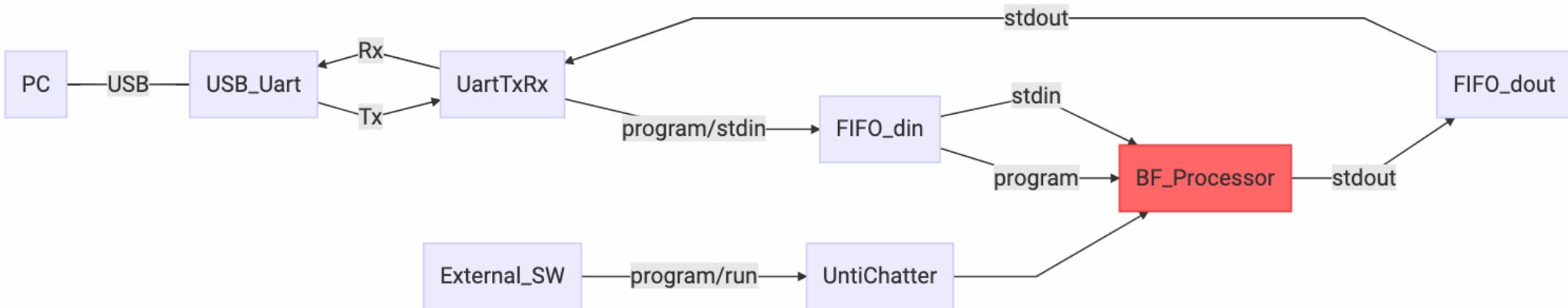
- <, > データのポインタをincrement/decrement
- +, - データをincrement/decrement
- , 入力値をデータに上書き
- . データを出力(putchar相当)
- [,] branch(while文相当)

BF処理系

- >, < データのポインタをincrement/decrement
- +, - データをincrement/decrement
- , 入力値をデータに上書き
- . データを出力(putchar相当)
- [,] branch(while文相当)
- "Hello World!"
 - ">++++++[<+++++>-]<.>>+++++[<++++>-]<+.+++++.+++.[-]>++++++[<++++>-]<.>>++++++[<++++>-]<.>>++++++[<+++>-]<.+++.-----.-----.[-]>++++++[<+++>-]<+.[-]+++++++. "

全体設計

- PCとはUARTで通信してプログラムの書き換えができるようにする
- プロセッサの動作を止めないようにFIFOを入れる



コード例: Processor

```
switch(inst) {
    is(0.U, "#".U) {
        halted := true.B
        errorCode := 0.U // no error
    }
    is('>'.U) {
        when (stackPtr === (stackMemSize - 1).U) {
            halted := true.B
            errorCode := 3.U // stack addr overflow
        } .otherwise {
            pc := (pc + 1.U)
            inst := instMem.read(pc + 1.U)
            stackPtr := (stackPtr + 1.U)
            stackData := stackMem.read(stackPtr + 1.U)
        }
    }
}
```

命令デコーダ

- BFのコードを読んで実行

いいところ

- 同期回路前提
 - 見通しがいい
- 型がはっきりしている
 - 曖昧だとエラー

コード例: Unit Test

```
// rx data
poke(c.io.rxReady, true.B)
poke(c.io.rxAck, false.B)
step(5)
for(d <- data) {
    println(s"\t[TEST] Data:$d")
    val sendData = Seq(
        false, // startbit
        (d & 0x01) != 0x00,
        (d & 0x02) != 0x00,
        (d & 0x04) != 0x00,
        (d & 0x08) != 0x00,
```

UART送受信

- タイミング生成
- データ送受

いいところ

- Scalaでテストが書ける
 - コード補完が効く
- デバッグ可

コード例: 結合テスト(1/2)

```
def run(src: String, dst: String, printDetail: Boolean = true) = {  
    var stdoutList = "" :: Nil  
  
    val result = Driver(() => new BrainfuckProcessor()) {  
        c => new PeekPokeTester(c) {  
            // initialize  
            poke(c.io.run, false.B)  
            poke(c.io.program, false.B)  
            poke(c.io.programData, 0.U)  
            poke(c.io.programValid, false.B)  
            poke(c.io.stdinData, 'X'.U)  
        }  
    }  
    result  
}
```

コード例: 結合テスト(2/2)

```
"Brainfuck" should "Program Data and stdout 'Hello World!'" in {
    val src = ">+++++[<++++++>-]<.>+++++[<++++>-]<+.+++++.+++. [-]>+++++[<.
    val dst = "Hello World!"
    run(src, dst, false)
}

"Brainfuck" should "nested loop" in {
    // !をつくる
    val src = ">+++[<++++++>-]<+++>+++++[<+++[<. >-]<+>>-]"
    val dst = (0 until 10).map(_ + '!').map(x => s"${x.toChar}" * 3).mkString
    println(dst)

    run(src, dst, false)
}

"Brainfuck" should "continuous stdout" in {
    // !をつくる
    val src = ">+++[<++++++>-]<+++.....>+++++[<. >-]"
    val dst = "!" * 20
    println(dst)
```

run()

- 文字列をMEMにロード
- haltするまでクロック供給
- 期待値検査

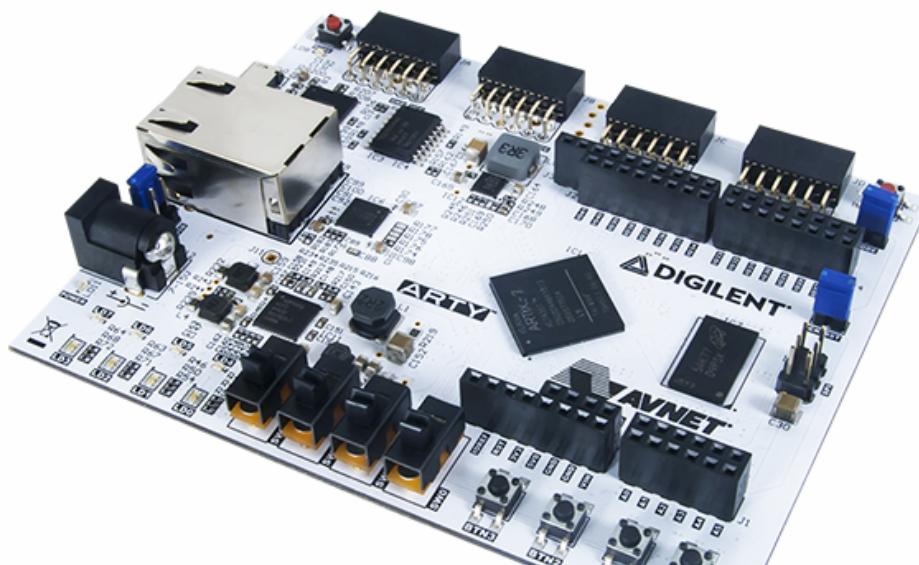
いいところ

- テストの関数/クラス化
 - 見通しがいい

できた

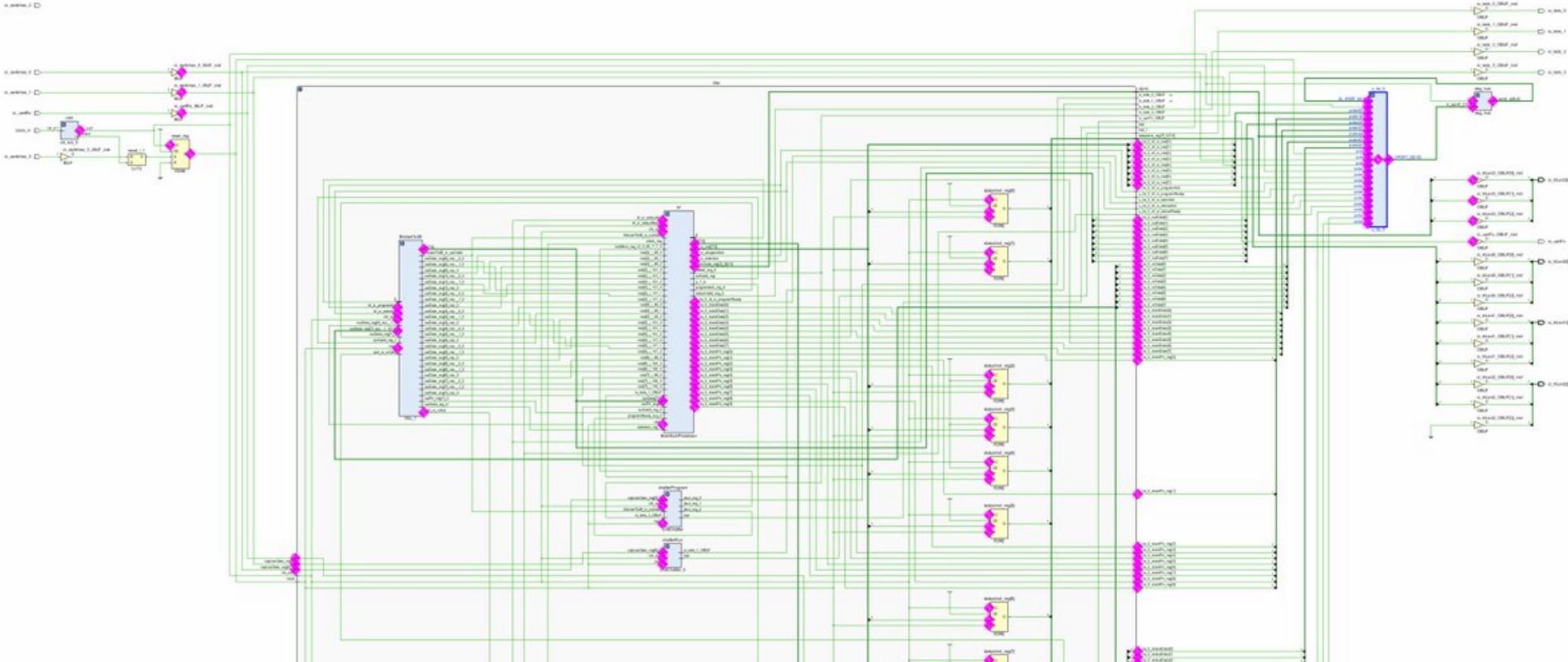
Verilog HDLで出力できるので、FPGAに書き込んで動かしてみる

ボード



- Digilent Arty A7
- Xilinx XC7A35Tが乗ってる
- 豪華なのでLE(リソース)死ぬほど余った

開発環境(Vivado)でちょちょいとすると



うございた！

まとめ

- Chisel楽しい
 - FPGA楽しい、処理系自作楽しい

まとめ

- Chisel楽しい
 - FPGA楽しい、処理系自作楽しい
- 同期回路前提の記述なので迂闊なバグを作り込みにくい
 - もちろんInter-Clock Path用の構文もある

まとめ

- Chisel楽しい
 - FPGA楽しい、処理系自作楽しい
- 同期回路前提の記述なので迂闊なバグを作り込みにくい
 - もちろんInter-Clock Path用の構文もある
- ScalaとHDL設計、両方のスキルセットを持った人ってあまりいないんじゃないじゃ...
 - どちらかを知っていれば大丈夫そう

Fin.