

自作 CPU 上で 例のレイトレを動かした話

第5回 自作CPUを語る会

2025.04.12

@htkymtks

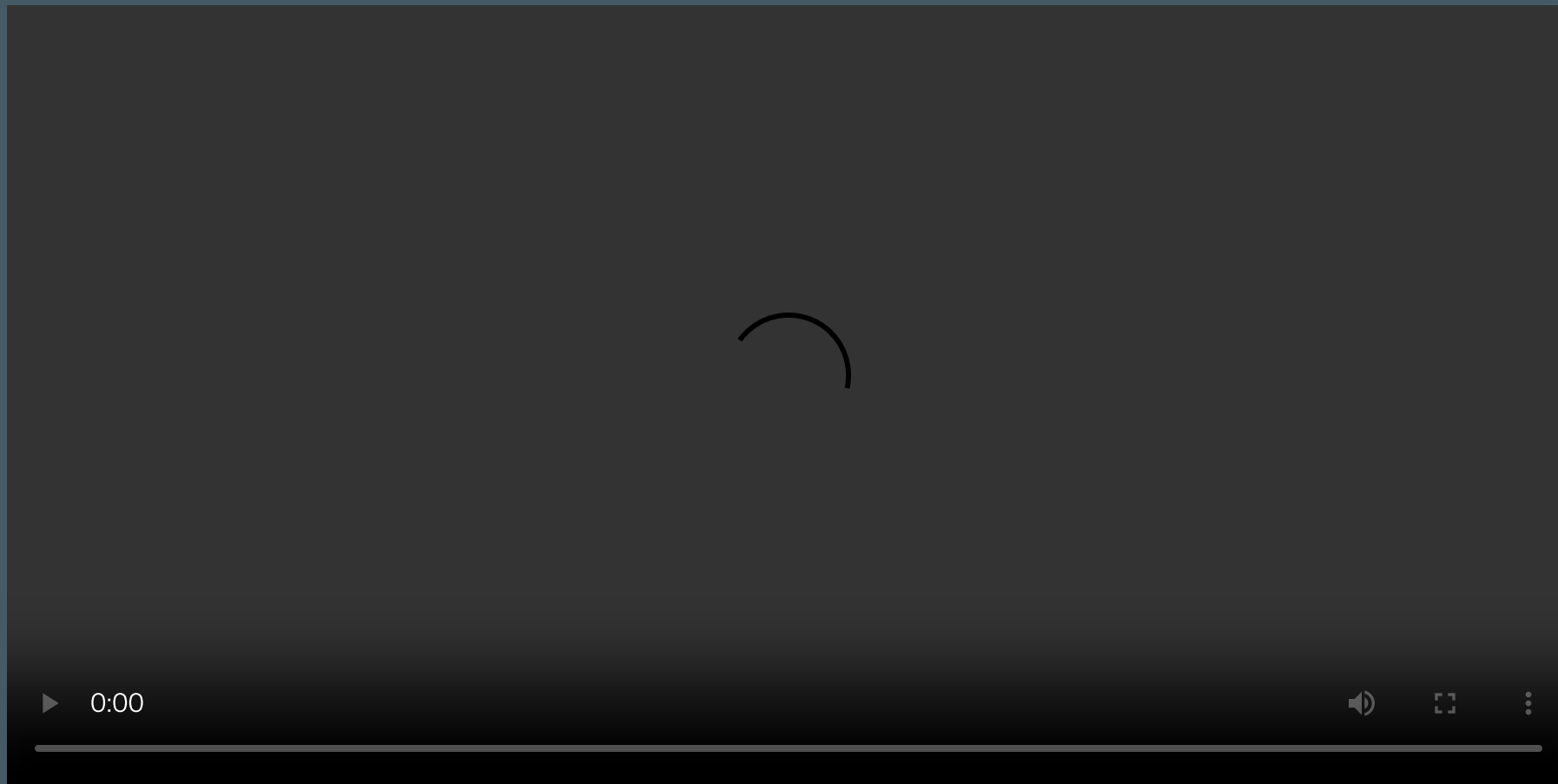
自己紹介

- はたけやまたかし
- 永和システムマネジメント 所属
 - RubyでWEBアプリケーション開発
- Twitter（現X）： @htkymtks

今日お話しすること

- 作成したCPUについて
- 自作CPU上でのレイトレを動かすまでの流れ

ライトレが動く様子



東大CPU実験とは？

- 東大理学部情報科学科の名物実験
- 自分たちでCPU、コンパイラ、シミュレーターなどを作成
- レイトレーシングの速さを競う
- 年度末に投稿されるブログ記事が毎年楽しみ 😊

CPU実験の4つの係

- コア係
 - FPGAボード上でHDLでCPUを実装
- コンパイラ係
 - MinCamlコンパイラを、自分たちのCPU向けに移植
- シミュレータ係
 - アセンブラとデバッグ用のシミュレータの作成
- FPU係
 - 浮動小数点演算器の作成
 - ライブラリ関数の実装

CPU実験の4つの係

- コア係 →  CPUを実装
- コンパイラ係 →  MinCamlコンパイラを移植
- シミュレータ係 →  既存のRISC-Vシミュレータを利用
- FPU係
 - 浮動小数点演算器 →  既存のFPUコアを利用
 - ライブラリ関数の実装 →  実装する

レイトレを動かす手順

- **1** FPGAボードを用意
- **2** CPUを作成
- **3** MinCamlコンパイラの移植
- **4** ライブラリ関数の実装
- **5** レイトレの組み込み

1 FPGAボードを用意



使ってるFPGAボード

- Radiona ULX3S 85F
 - FPGA : Lattice ECP5
 - LUT数 : 85K LUT
 - SDRAM : 32MB
 - ブロックRAM : 486KB
- 価格
 - 2021年: 1万8000円
 - 2025年: 3万8000円 

ULX3S 以外でもいける？

以下の条件を満たせば他のFPGAボードでも大丈夫そう

- 必須

- LUT数: 12000（作成したCPUが 1万2000 LUT）
 -  Tang Nano 9K、 Tang Primer 20K
- SDRAM: 5MB以上（スタック1MB + ヒープ4MBが必要）

- 推奨

- ブロックRAM: 250KB以上（レイトレのサイズがこれくらい）
 - 足りなければブートローダーが必要

2 CPUの作成

CPUのスペック

- CPUアーキテクチャ
 - 32ビットRISC-V (RV32IF)
 - 整数演算 + 単精度浮動小数点演算
- メモリ構成
 - ROM (ブロックRAM) : 256KB
 - RAM (SDRAM) : 32MB
- 周辺機器
 - UART
 - LED (8ビット)

メモリマップ

- プログラムは0番地からスタート
- SDRAMの領域はヒープとスタックの領域
- UARTやLEDはメモリマップトI/Oでアクセス

開始アドレス	終了アドレス	サイズ	説明
0x00000000	0x0003ffff	256KB	ブロックRAM領域（プログラム/データ）
0x40000000	0x4fffffff	64MB	SDRAM領域（ヒープ/スタック）
0xf0000000	0xf0000000	1	UART データレジスタ
0xf0000004	0xf0000004	1	UART コントロールレジスタ
0xf0001000	0xf0001000	1	LED コントロールレジスタ

開発ツール

- HDL
 - System Verilog
- 合成ツール
 - yosys
- 配置配線ツール
 - nextpnr

OSS CAD Suite をインストールすると全部ついてくる

CPUの作り方

- 以下の本を参考にすれば、ある程度は作れる
 - デジタル回路設計とコンピュータアーキテクチャ（RISC-V版）
 - RISC-V原典
 - FPGA Prototyping by SystemVerilog Examples: Xilinx MicroBlaze MCS SoC Edition

既存のものを利用

- FPUコア
 - <https://github.com/dawsonjon/fpu>
- UARTコントローラ
 - <https://github.com/freecores/osdву>
- SDRAMコントローラ
 - <https://github.com/machdyne/zucker/blob/main/rtl/sdram.v>

実装した命令一覧

- 整数命令 (RV32Iのサブセット)
 - lw, sw, add, addi, sub, and, or, xor, slt, sltu, beq, bne, ble, bne, blt, bgt, bge, jal, jalr, lui, auipc, ori, srl, sra, sll
- 浮動小数点命令 (RV32Fのサブセット)
 - flw, fsw, fadd_s, fsub_s, fmul_s, fdiv_s, fcv_t_s_w, fcv_t_w_s, fsgnj_s, fsgnjn_s, feq_s, flt_s, fle_s, fmv_x_w, fmv_w_x

3 MinCamlコンパイラの移植

MinCamlとは

- <https://github.com/esumii/min-caml>
- 教育用コンパイラ
- OCamlのサブセット
- MinCamlはOCamlで書かれている

MinCamlとレイトレ









- CPU実験のレイトレはMinCaml製
- MinCamlを自作CPUへ移植することで、自作CPUの上でレイトレが動く

MinCamlの対応アーキテクチャ












オリジナルのMinCamlの対応CPUアーキテクチャは以下の3つ

- UltraSPARC
- PowerPC
- 32ビット x86
- （ここにRISC-Vを追加する）

アーキテクチャ依存コードの置き場所

-  min-caml/
 -  SPARC/
 -  x86/
 -  PowerPC/
 -  emit.ml (アセンブリコード生成部)
 -  libmincaml.S (ライブラリ関数)
 -  asm.ml
 -  ...

コンパイラ移植の流れ

-  min-caml/
 -  PowerPC/
 -  RV32/ ( PowerPCディレクトリを複製)
 -  asm.ml ( レジスター一覧を修正)
 -  emit.ml ( 埋まってるアセンブリをRISC-Vに修正)
 -  libmincaml.S ( ライブラリ関数をRISC-Vで書き換え)
 -  ...

MinCamlの開発環境

- OCaml
 - MinCaml自体がOCamlで書かれている
- RISC-V GNUツールチェーン
 - Cコンパイラ、アセンブラ、リンカ、その他バイナリユーティリティ等
 - シミュレータ (Spike) も付属

MinCamlコンパイラの移植で困ったこと

- OCamlわからん
- アセンブリわからん

OCamlわからん

- MinCamlはOCamlで書かれている
- OCamlは独特な世界観でとっつきづらい
- プログラミングの基礎（通称：浅井本）がオススメ
 - プログラミングの初心者向けの本ではあるが、関数型プログラミングやOCamlの初心者にもおすすめ
 - OCamlの入門をいくつか読んでもピンと来なかったが、この本の練習問題をこなしてるうちにMinCamlのソースが読めるようになった

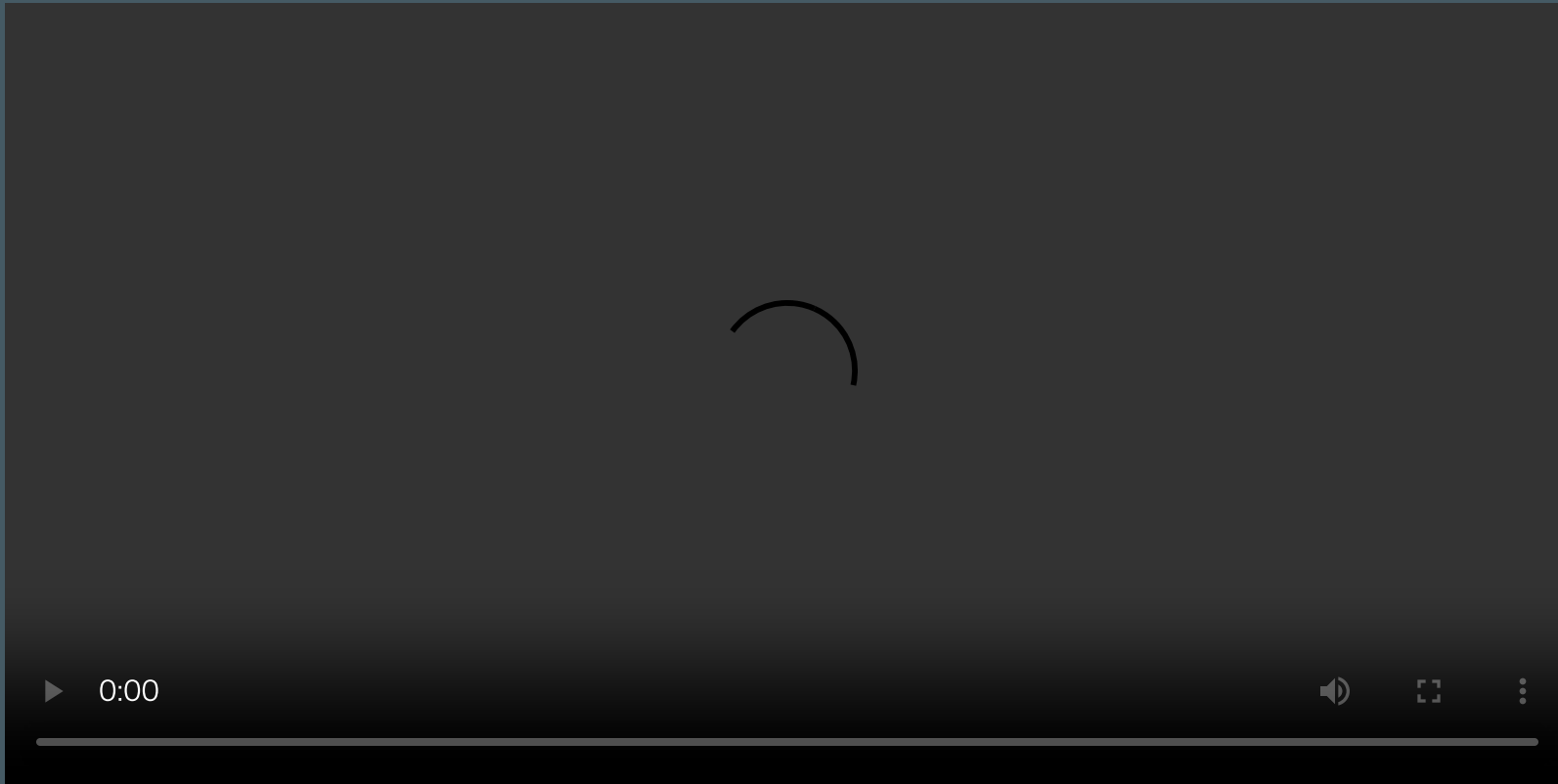
アセンブリわからん

アセンブリの書き方に悩んだら、Cコンパイラが出力するアセンブリを確認する

- 2つの確認方法
 - Compiler Explorer
 - GCCの -S オプション

⚡ Compiler Explorer (<https://godbolt.org/>)

様々な言語・様々なCPUのアセンブリ出力を確認できる神サイト



GCCの -S オプション

GCC の `-s` オプションで、Cからアセンブリを出力できる

```
// test.c
int return_100() {
    return 100;
}
```

```
$ gcc -S -masm=intel test.c
```

GCCの **-S** オプション

出力されたアセンブリコード

```
$ gcc -S -masm=intel test.c
$ cat test.s
        .intel_syntax noprefix
        .text
        .globl  return_100
        .type   return_100, @function
return_100:
        push    rbp
        mov     rbp, rsp
        mov     eax, 100
        pop     rbp
        ret
```

Compiler Explorer と GCC の使い分け

Compiler Explorer が出力したアセンブリは、出力オプションやプラットフォームの違いで、そのままでは動かないことがある

- 出力したアセンブリをそのままビルドにかけたい場合
 - → GCC
- それ以外
 - → Compiler Explorer

4 ライブラリ関数の実装

必要なライブラリ関数

- 入出力関数
 - `print_int`, `read_int`, `print_byte`, `read_byte`, `read_float`
- 三角関数
 - `sin`, `cos`, `atan`
- 平方根
 - `sqrt`
- その他浮動小数点関数
 - `floor`, `abs_float`, `float_of_int`, `int_of_float`

ライブラリの実装言語

- オリジナルのMinCamlのライブラリ
 - アセンブリで実装
 - libmincaml.S
- 移植したMinCamlのライブラリ
 - MinCamlで実装（一部アセンブリ）
 - min-rt/min-rt.ml と libmincaml.S

入出力関数

入出力関数は、UARTを利用して入出力を行う

- `print_int`
- `read_int`
- `print_byte`
- `read_byte`
- `read_float`

平方根を求める sqrt 関数

- Babylonian method

- https://cpplover.blogspot.com/2010/11/blog-post_20.html

```
def sqrt(s)
  x = s / 2.0
  last_x = 0.0

  while x != last_x
    last_x = x
    x = (x + s / x) / 2.0
  end
  x
end
```

三角関数

- \sin
- \cos
- atan

マクローリン展開

$$\cos x = 1 - \frac{1}{2!}x^2 + \frac{1}{4!}x^4 - \frac{1}{6!}x^6 + \dots$$

$$\sin x = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \dots$$

$$\arctan x = x - \frac{1}{3}x^3 + \frac{1}{5}x^5 - \frac{1}{7}x^7 + \dots$$

sin 関数

```
def sin(x)
  x = adjustx(x) # x の値が  $-\pi$  から  $\pi$  の間に収まるように調整
  x2 = x * x
  x3 = x2 * x
  x5 = x3 * x2
  x7 = x5 * x2
  x9 = x7 * x2
  x - x3 / 6 + x5 / 120 - x7 / 5040 + x9 / 362880
end
```


角度の調整

xの値が0から離れるほど誤差が大きくなる



角度の調整

sinやcosは同じ波が繰り返されるので、 x の値を-3.14から3.14の間に収まるよう調整



(角度調整前)



(角度調整後)

atan関数

(うまく実装できなかったなので、1.0 を返してお茶を濁した...)

```
let rec atan x = 1.0 in
```



その他浮動小数点関数

- `float_of_int` → (FPUを呼び出し)
- `int_of_float` → (FPUを呼び出し)
- `floor` → (`int_of_float` を使えば簡単)
- `abs_float` → (符号ビットに0 (正) をセットするだけ)

ライブラリ関数の参考資料

- CPU実験の浮動小数点数演算について
 - <https://nekketsuuu.github.io/entries/2015/12/11/cpu-experiment-floats.html>



5 レイトレの組み込み

プログラム書き込みの流れ



ブロックRAMの容量とバイナリサイズ

- ULX3SのブロックRAMサイズ → 400KB程度
- レイトレのバイナリサイズ → 200KB程度
- レイトレがブロックRAMに収まる場合
 - ブロックRAMからレイトレを起動可能
- 収まらない場合
 - プログラムローダーが必要
 - UARTやSDカードからプログラムを読み込み、SDRAMへ転送

レイトレサーバ

```
# レイトレサーバ
# usage: ruby script/server.rb > contest.ppm

require 'rubyserial'

begin
  serialport = Serial.new '/dev/cu.usbserial-D00084', 115200

  # 不要なデータを読み捨て
  while serialport.getbyte
    # 何もしない
  end

  # 自作CPUに送信開始を依頼
  serialport.write " "

  # UART読み込み
  while true
    puts serialport.gets
    \$stderr.puts "receiving..."
  end
rescue Interrupt
  serialport.close
end
```

 もっと詳細を知りたい人はこちら

(以下古いやつ)

開発環境の構築

- ハードウェア開発環境
- ソフトウェア開発環境

ハードウェア開発環境

開発言語は System Verilog で、開発ツールはオープンソースな合成・配置配線ツールである yosys と nextpnr を利用しています。

yosys と nextpnr は、インストールが容易で、macOS ネイティブでも動作するので、手軽に FPGA 開発を行うことができ気に入っています。

- HDL: System Verilog
- 開発ツール: yosys + nextpnr
 - オープンソースな合成+配置配線ツール
 - macOS で動く 🍏

ソフトウェア開発環境

- OCaml (MinCamlコンパイラの作成に利用)
- RISC-V GNU ツールチェーン (アセンブラ、リンカ等)

自己紹介

- はたけやまたかし
- 株式会社永和システムマネジメント
 - RubyでWEBアプリケーション開発
- Twitter（現X）： @htkymtks

趣味

- 低レイヤプログラミング
 - 自作CPU
 - 自作RISC-Vシミュレーター
 - 自作コンパイラ

今日お話しすること

- 東大CPU実験とレイトレーシングについて
- CPUを自作してレイトレを動かすまでの流れ
- 苦労したことやハマりどころ

東大CPU実験とは？

- 東大情報科学科の名物実験
- 自分たちでCPU、コンパイラ、シミュレーターを作成
- その上でレイトレーサーを動かして速さを競う

レイトレーサーとは？

- 物体と光源の光線の経路を追跡することで3D空間の見え方をシミュレートするソフトウェア
- MinCamlというプログラミング言語で書かれている

レイトレーサーの流れ

始めたきっかけ

- CPUの創りかたのTD4
 - はんだ付け難しそう → FPGAならいけるかも → できた！
- 毎年見かけるCPU実験のブログ記事
 - TD4が作れたのなら、RISC-Vも作れるのでは？



作成した CPU

- 32ビット RISC-V
 - RV32IF（整数演算と単精度浮動小数点演算）
- 動作周波数：25MHz
- メモリ：256KB（FPGA ブロックRAM） + 64MB（SDRAM）
- ノイマンアーキテクチャ
 - プログラムメモリとデータメモリでアドレス空間を共有
- LUT数: 12000
 - TangNano 9Kには収まらない...





MinCamlとレイトレーサー

- MinCamlコンパイラ
 - <https://github.com/esumii/min-caml>
 - CPU実験で作成するコンパイラのリファレンス実装
 - OCaml で書かれた OCaml のサブセット言語
 - 以下のアーキテクチャ向けのアセンブリを出力可能
 - PowerPC, UltraSPARC, x86
 - （ここに自分たちのアーキテクチャを追加するのがコンパイラ関係の仕事）

MinCamlとレイトレーサー

- レイトレーサー
 - CPU実験のベンチマークプログラム
 - MinCamlで書かれている
 - <https://github.com/esumii/min-caml/blob/master/min-rt/min-rt.ml>
 - オブジェクト定義ファイルを読み込んでレイトレーシングを実行
 - 結果をPPMフォーマット(テキスト形式の画像フォーマット)で出力

CPU開発環境

- FPGAボード: Radiona ULX3S 85F (Lattice ECP5 を搭載)
 - 2021年: 1万8000円
 - 2025年: 3万8000円 
- 開発言語: System Verilog
- 開発ツール: yosys + nextpnr
 - オープンソースな合成+配置配線ツール
 - インストールが容易
 - macOS で動く 

😊 今日話すこと

- TinyRubyの紹介
- コンパイラ作成Tips
- コンパイラはじめての一步

提供

株式会社 永和システムマネジメント

以下、ボツスライド



東大CPU実験の4つの班

- コア係（HDL を書いて CPU を作る）
- コンパイラ係（MinCamlコンパイラの移植）
- シミュレータ係（デバッグ用のシミュレータとアセンブラの作成）
- FPU係（浮動小数点演算器とライブラリ関数(sin, cos など)の作成）



やったこと

- コア係 → ○ RISC-V CPUを作成
- コンパイラ係 → ○ MinCamlコンパイラを移植
- シミュレータ係
 - シミュレータ → 既存の RISC-V シミュレータ (Spike) を利用
 - アセンブラ・リンカ → RISC-V GNUツールチェーンのものを利用
- FPU係
 - FPU → 既存の FPU コアを利用
 - ライブラリ関数 → ○ 書いた