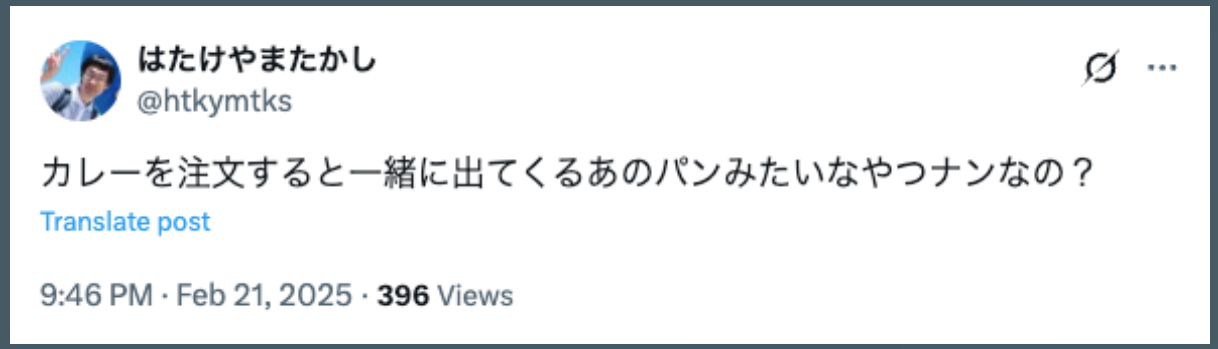
自作 CPU 上で
例のレイトレを動かした話 🚀

@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



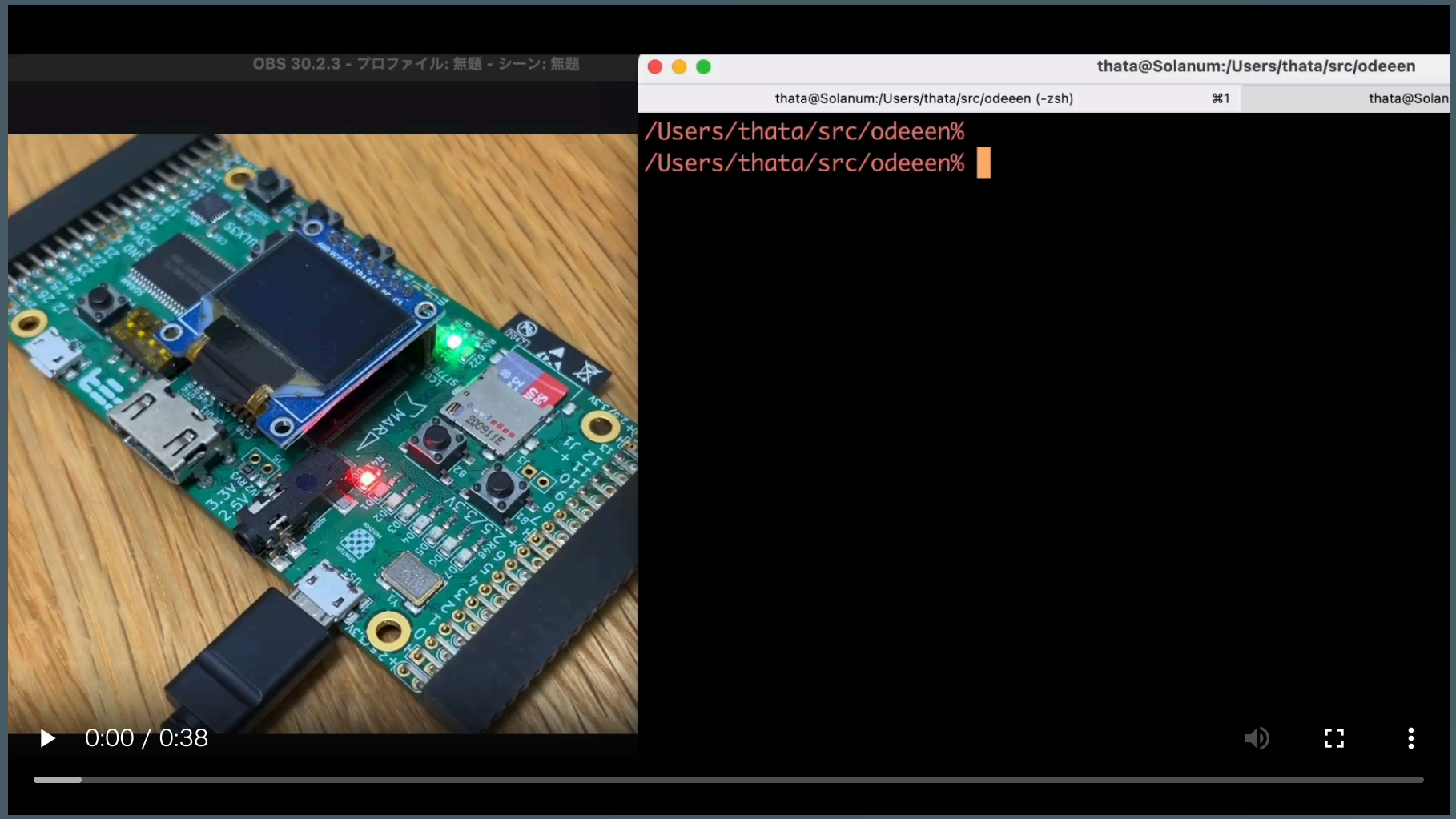
🛠️ 趣味

- 低レイヤプログラミング
  - 自作CPU
    - TD4, RISC-V, Turing Complete
  - コンパイラ
    - MinCaml移植（AArch64, RISC-V）
    - minrubyc
      - https://blog.agile.esm.co.jp/entry/minruby-compiler
  - RISC-Vシミュレータ
    - https://blog.agile.esm.co.jp/entry/2022/06/10/115327

🙂 今日お話しすること

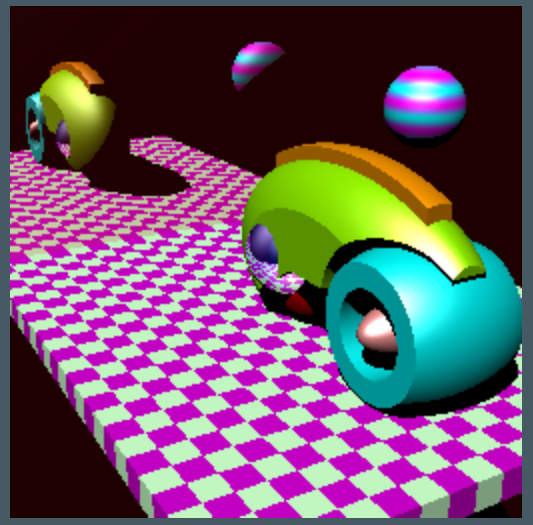
- CPUの自作からレイトレまでの流れ
- 苦労したところ

レイトレが動く様子


OBS 30.2.3 - プロファイル: 無題 - シーン: 無題
thata@Solanum:/Users/thata/src/odeeen
thata@Solanum:/Users/thata/src/odeeen (-zsh)
⌘1
thata@Solan
/Users/thata/src/odeeen%
/Users/thata/src/odeeen%
0:00 / 0:38

🗼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
- 毎年投稿されるCPU実験の記事が楽しみ😄



CPU実験の４つの係

- コア係
  - FPGAボード上でVerilogなどで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（FPU）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

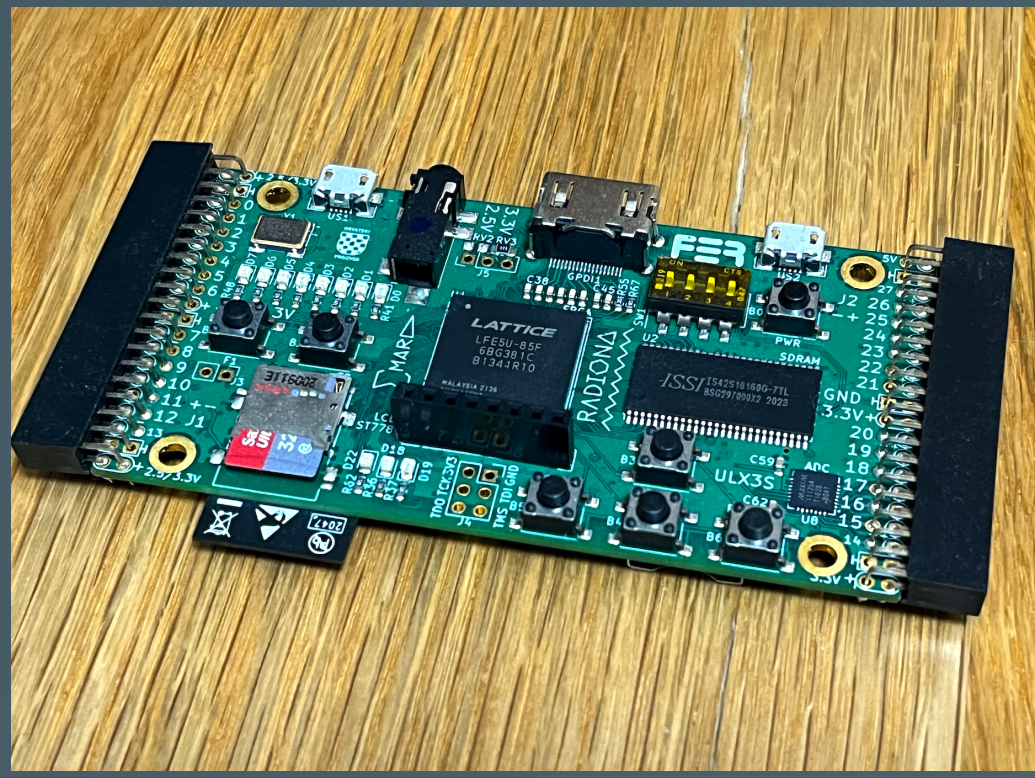
🌐 レイトレを動かす手順

- 1️⃣ FPGAボードを用意
- 2️⃣ CPUを作成
- 3️⃣ MinCamlコンパイラの移植
- 4️⃣ ライブラリ関数の実装
- 5️⃣ 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 以外のFPGAボードでも動く？

- 必須
  - LUT数: 12K程度
  - SDRAM必須（スタック1MB + ヒープ4MBで5MB以上）
- 推奨
  - ブロックRAM: 250KB以上（レイトレのサイズがこれくらい）
    - 足りなければローダーが必要になるかも
- ❌ Tang Nano 9K
- ✅ Tang Primer 20K ならいけるかも？

2 CPUの作成

🧮 作成した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
    - コンパクト命令には非対応
- バスインターフェース
  - PicoRV32 Native Memory Interface
- メモリ構成
  - ROM（ブロックRAM）：256KB
  - RAM（SDRAM）：32MB
- 周辺機器（メモリマップトIO）
  - UART
  - LED（8ビット）

🧰 開発ツール

- HDL
  - System Verilog
- 合成・配置配線ツール
  - yosys + nextpnr
  - macOSで動く🍎
- OSS CAD Suite をインストールすると全部ついてくる
  - https://github.com/YosysHQ/oss-cad-suite-build
  - （Gowin の FPGA も一部サポート）

📙CPU作成の参考書

- CPU作成の基本
  - ディ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の仕様
  - RISC-V原典
- System Verilog の書き方
  - FPGA Prototyping by SystemVerilog Examples https://www.wiley.com/en-us/FPGA+Prototyping+by+SystemVerilog+Examples%3A+Xilinx+

📜 自作CPUに実装した命令一覧

- 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, fcvt_s_w, fcvt_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コンパイラとは

- https://github.com/esumii/min-caml
- 教育用コンパイラ
- OCamlのサブセット
- MinCamlコンパイラは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
- レイトレはMinCamlリポジトリの中に
  - 📂 min-caml/
    - 📂 min-rt/
      - 📄 min-rt.ml （レイトレーサーのソースコード）

✨🍎 MinCamlの対応アーキテクチャ

オリジナルのMinCamlの対応しているCPUアーキテクチャ

- UltraSPARC
- PowerPC
- 32ビット x86

📂 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/ （1️⃣ PowerPCディレクトリを複製）
    - 📄 asm.ml （2️⃣ レジスター覧を修正）
    - 📄 emit.ml （3️⃣ 埋まってるアセンブリをRISC-Vに修正）
    - 📄 libmincaml.S （4️⃣ ライブラリ関数をRISC-Vで書き換え）
    - 📄 ...

🐫🐏 MinCamlの開発環境

- OCaml
  - MinCamlコンパイラ自体がOCamlで書かれている
- RISC-V GNUツールチェイン
  - 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

🐫 OCamlわからん

- MinCaml移植の最初の壁
- OCamlはとっつきにくい（→ 今は好きだよ）
  - 文法やモジュールシステムが独特
  - 関数型言語/再帰
- プログラミングの基礎（通称：浅井本）がオススメ

🦀 Rustで書かれたMinCamlコンパイラ

- mincaml-rs
  - https://github.com/utokyo-compiler/mincaml-rs
- 昨年度のCPU実験から利用可能に

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
- 浮動小数点数変換
  - floor, abs_float, float_of_int, int_of_float

🥕 平方根

- 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
```

📐 三角関数とマクローリン展開

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

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

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

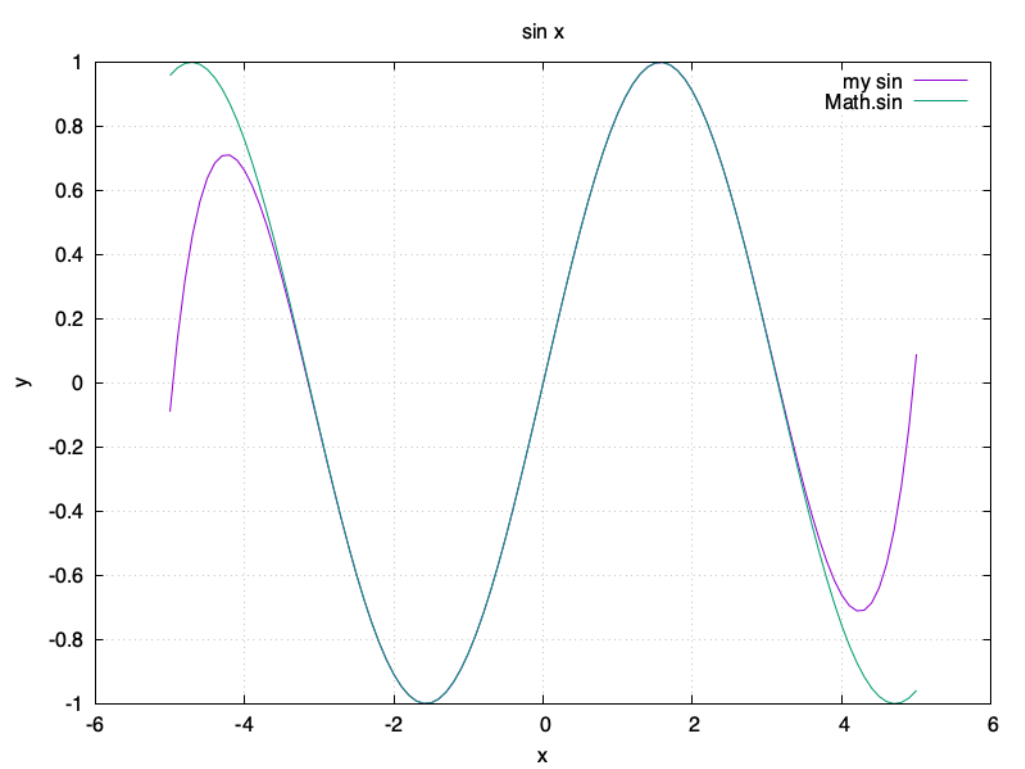
📐 sin 関数

$$\sin x = x - \frac{1}{3!}x^3 + \frac{1}{5!}x^5 - \frac{1}{7!}x^7 + \frac{1}{9!}x^9 \cdots$$

```
def sin(x)
  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
```

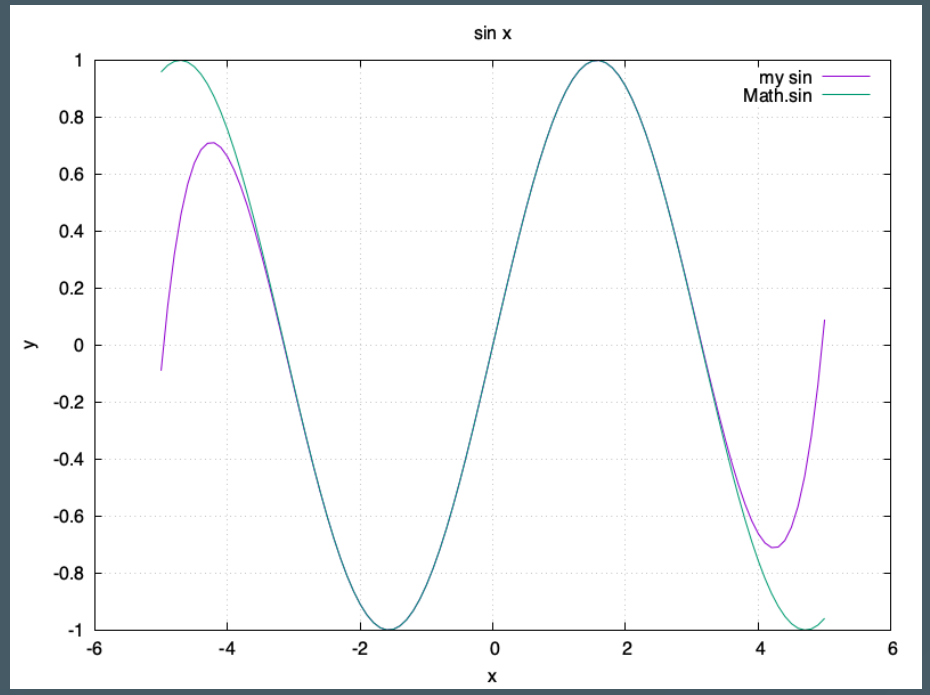
📐角度の調整

xの値が0から離れるほど誤差が大きくなる

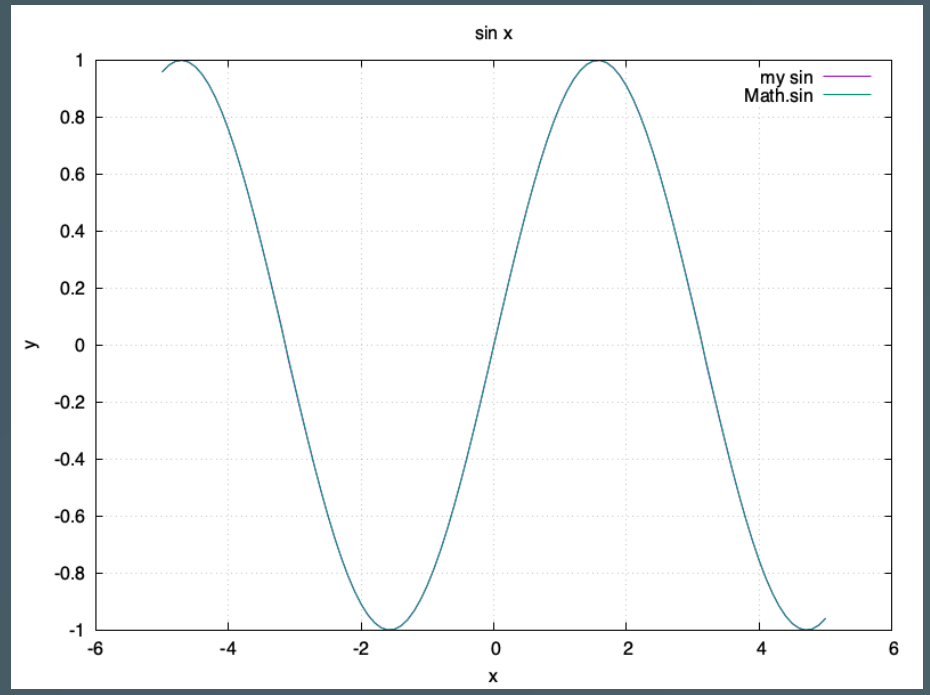


📐角度の調整

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）

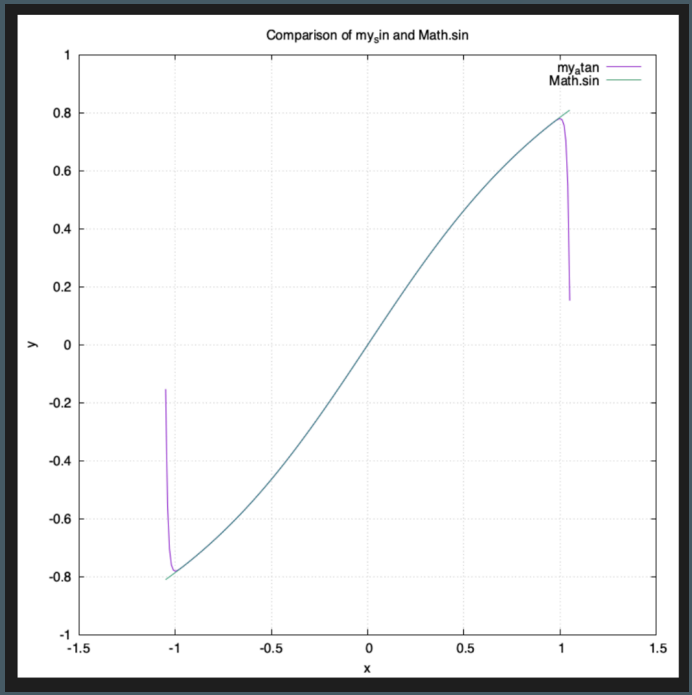


（角度調整後）

🍵 atan関数

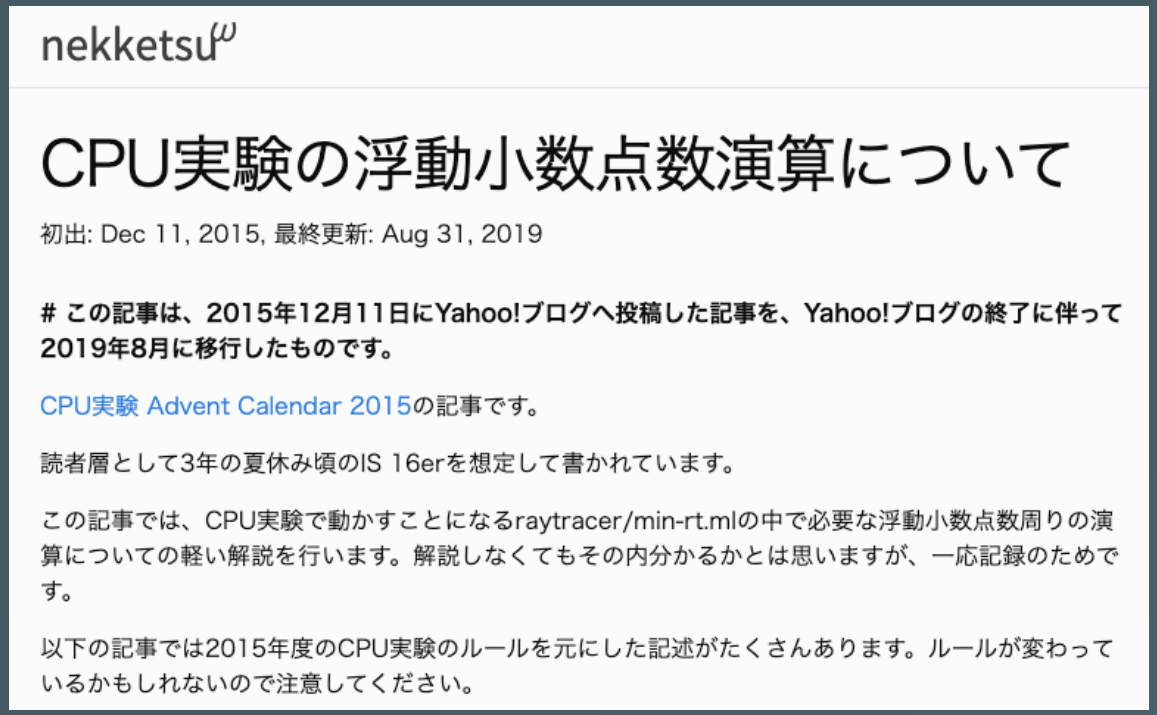
うまく実装できなかったので、1.0 を返してお茶を濁した...

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



🔥 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



nekketsuω

CPU実験の浮動小数点数演算について

初出: Dec 11, 2015, 最終更新: Aug 31, 2019

 # この記事は、2015年12月11日にYahoo!ブログへ投稿した記事を、Yahoo!ブログの終了に伴って2019年8月に移行したものです。

CPU実験 Advent Calendar 2015の記事です。

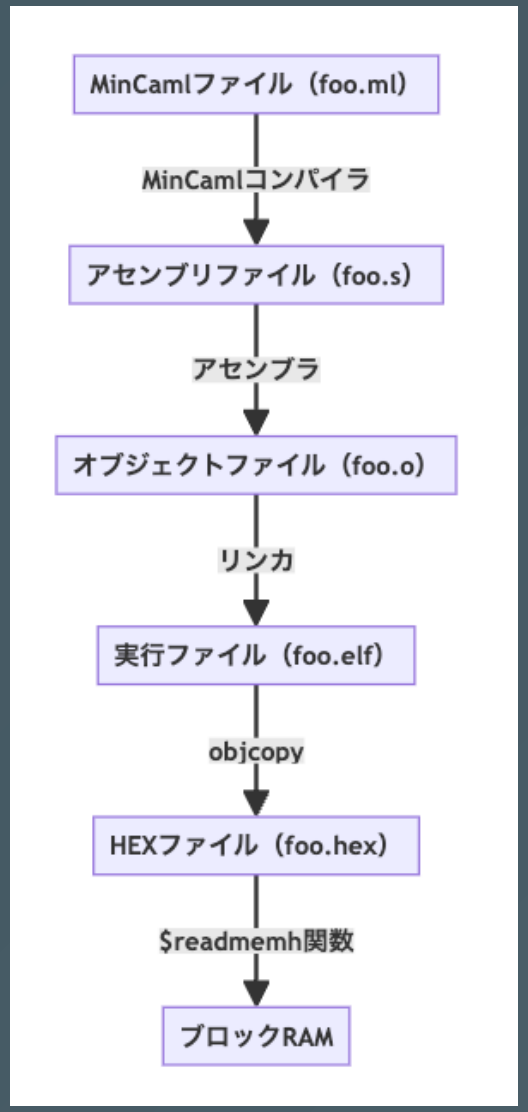
読者層として3年の夏休み頃のIS 16erを想定して書かれています。

この記事では、CPU実験で動かすことになるraytracer/min-rt.mlの中で必要な浮動小数点数周りの演算についての軽い解説を行います。解説しなくてもその内分かるかとは思いますが、一応記録のためです。

以下の記事では2015年度のCPU実験のルールを元にした記述がたくさんあります。ルールが変わっているかもしれないので注意してください。

5 レイトレの組み込み

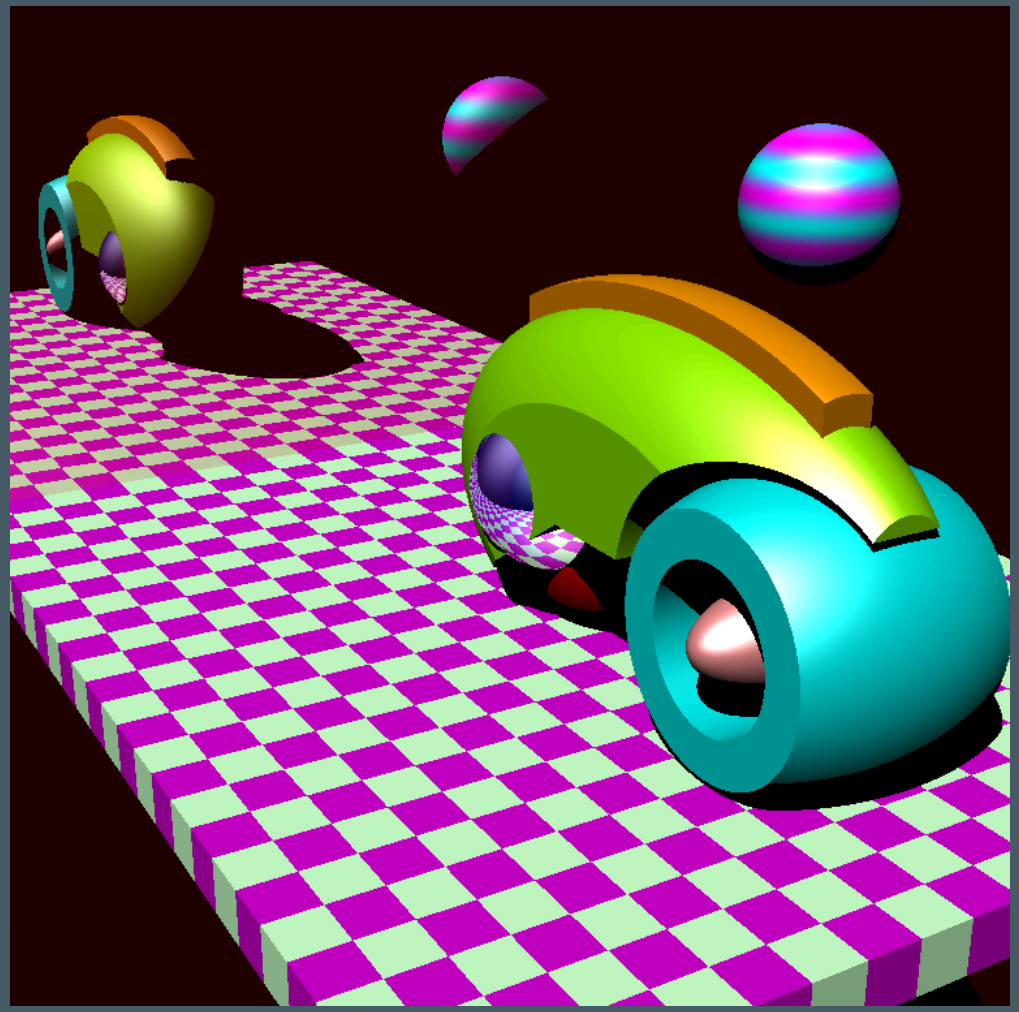
📝 プログラム書き込みの流れ



レイトレサーバ

```ruby
# 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
```

ugoita !!



😅 レイトレを動かすのに苦労したところ

- RAMが足りない
- メモリのバスインターフェース
- 開発マシンがmacOS

⚡RAMが足りない

- レイトレには5MBのRAMが必要（スタック1MB、ヒープ4MB）
- ブロックRAM → 全然足りない
- SDRAM
  - SDRAMコントローラーの難易度が高い
    - 挑戦してみるもうまく動かず、2〜3年停滞
    - （XilinxのMIGの存在は知っていたが、ULX3Sでやりたい）
  - いい感じにSDRAMコントローラーを発見 → SDRAM動いた
    - https://github.com/machdyne/zucker

🚚 メモリのバスインターフェース

- CPU自作入門書に出てくるバスは、だいたい非同期バス
- 同期バスへのステップアップに苦労
  - AXIやWishboneも難しくてよく分からん...
- 理解のきっかけ
  - PicoRV32のバス
  - Interface 2024年12月号
    - 「VALIDとREADYを使ったインターフェース」（IDAさん）

🍎 開発マシンがmacOS

- 最初は Altera や Xilinx のボードを使ってた
  - Altera も Xilinx も macOS には未対応
    - 仮想環境は遅いし面倒...
- ULX3S（Lattice ECP5）
  - yosys + nextpnr は macOS にネイティブ対応

🚀 続きはWEBで