Iroha を使ってみた話

2016 年 11 月 22 日

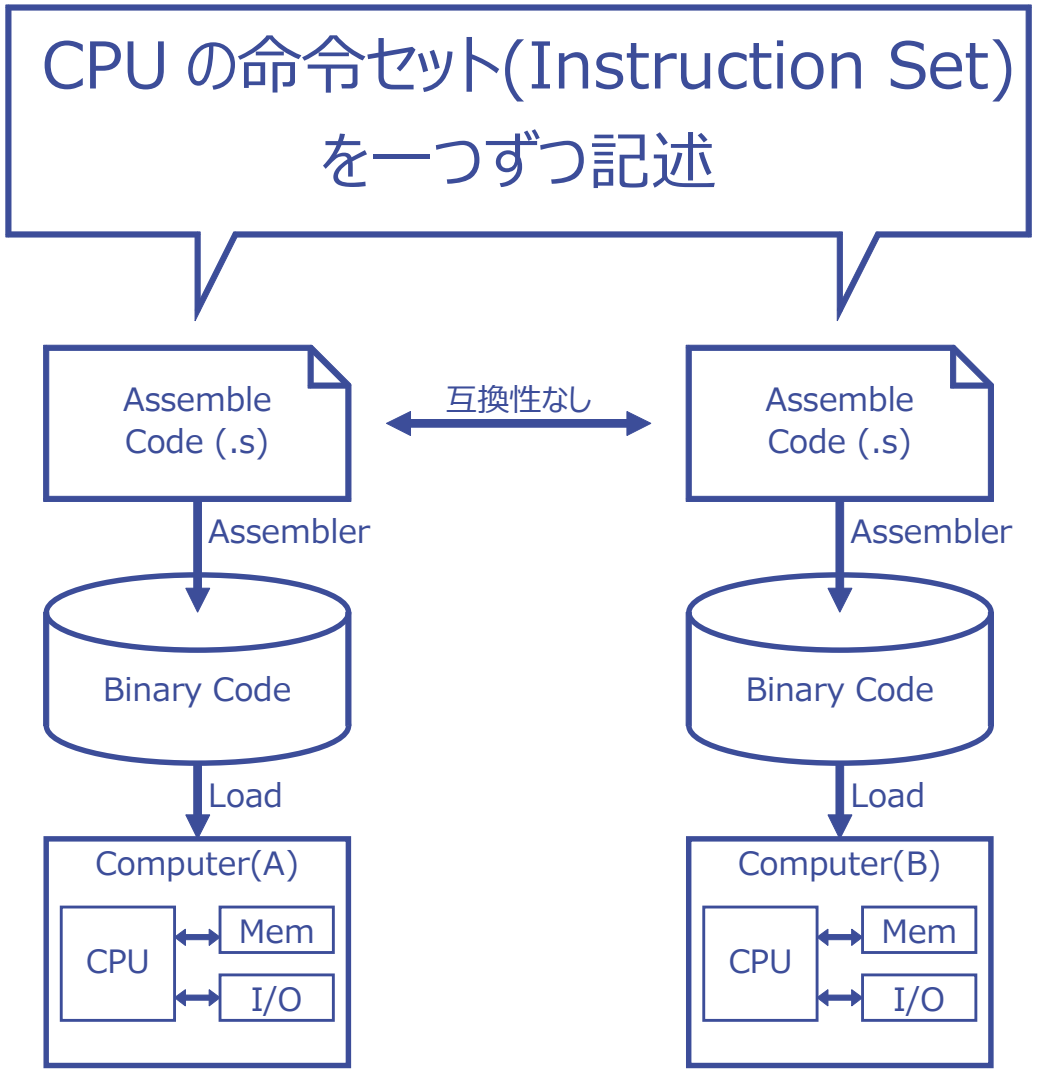
@ikwzm

自己紹介みたいなもの

- ハンドルネーム ikwzm
- 現在隠居中
- もうすぐ 52 才(けっこう年)
- 主に論理回路設計 (回路図～VHDL)
- たまにプログラム(アセンブラ～ C /Ruby)
- 言語の設計は素人

昔話を少し

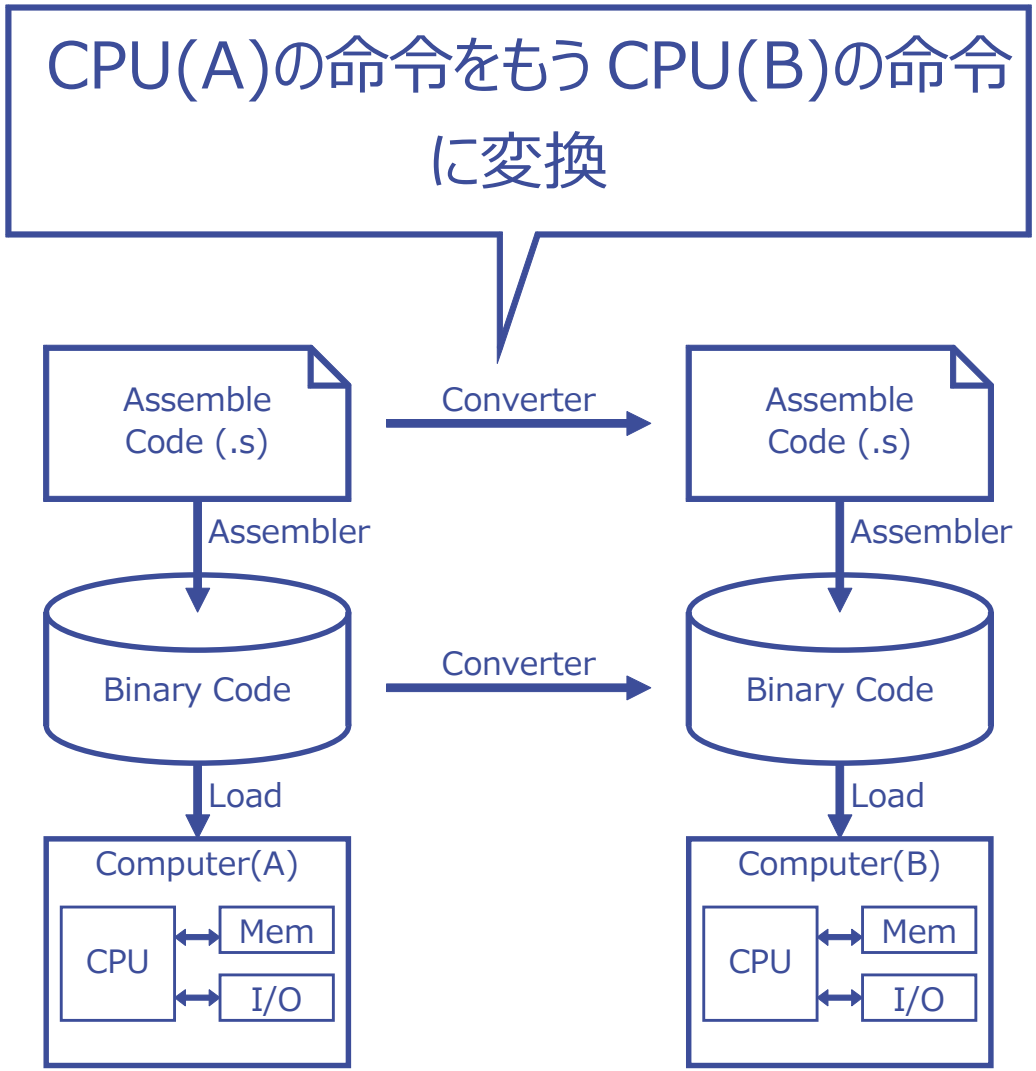
アセンブリ言語(機械語)によるプログラミング(暗黒?)時代



◇ 問題点

- 生産性
  - いちいち命令を並べるの？
    （面倒くせ~）
- 互換性
  - CPU 間で互換性無し
    （メーカーによる囲い込み）

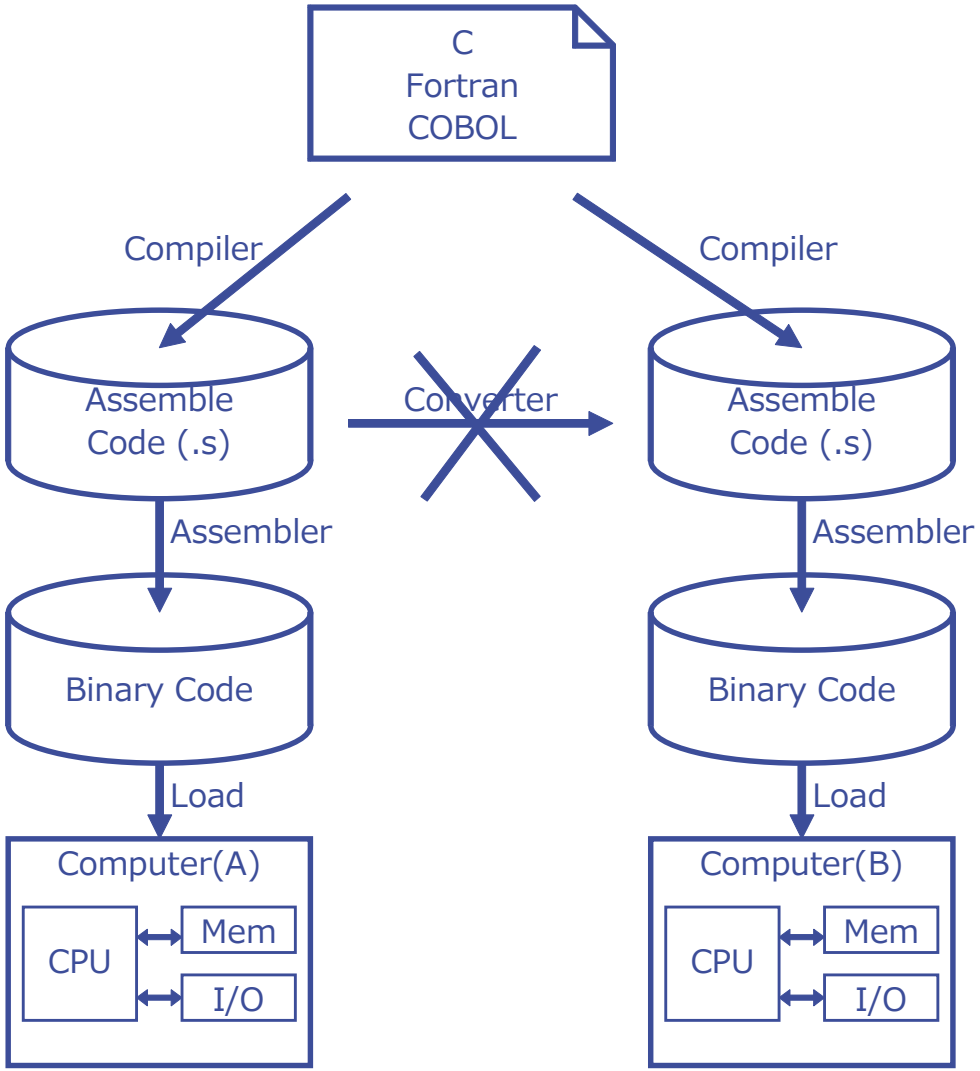
機械語コンバーター



◇ 問題点

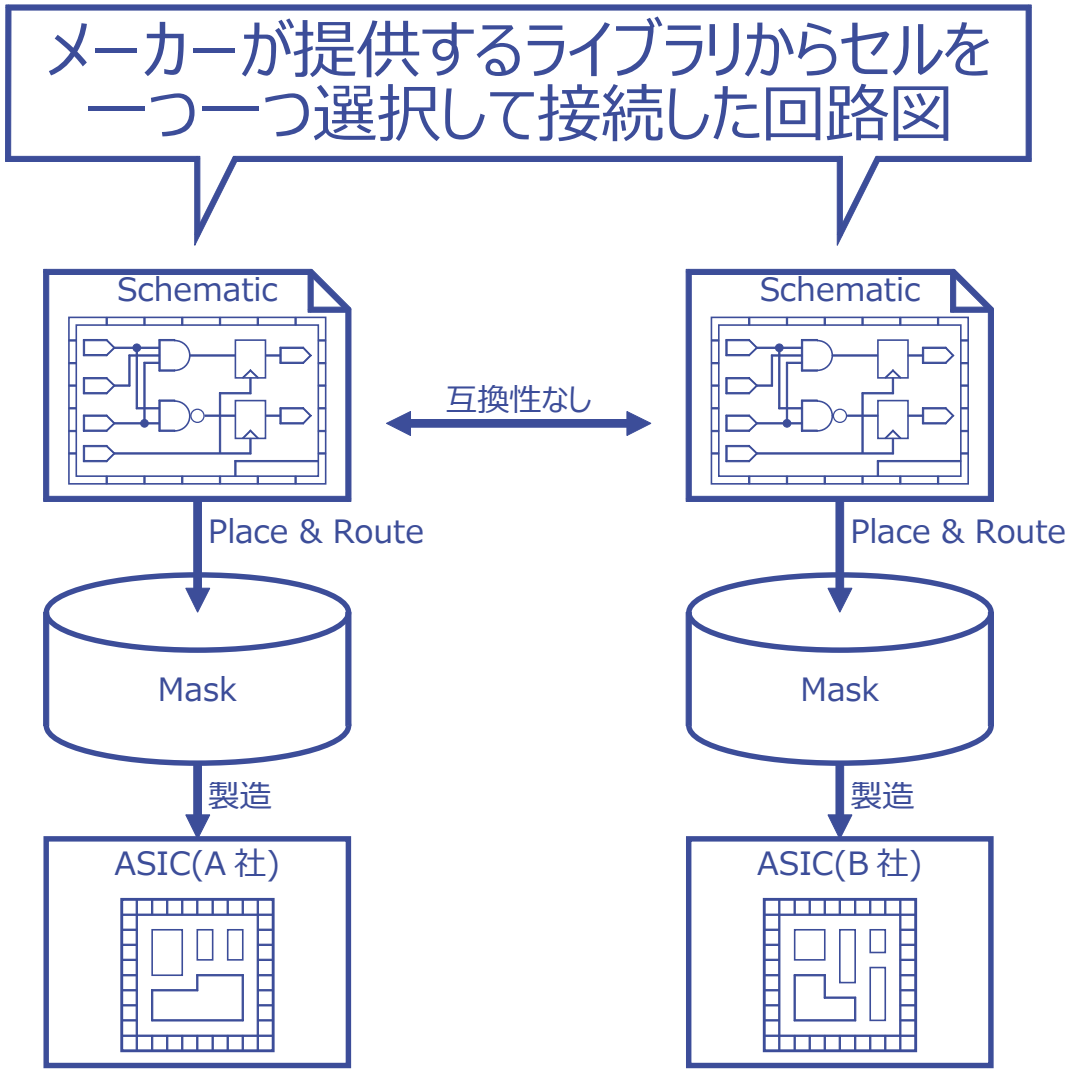
- 生産性
  - アセンブラで書くことには変わりない (なんの解決にもなってない)
- 性能
  - 変換したコードは CPU が本来持つ能力を発揮できなかった

高位言語とコンパイラによるプログラミング



- いくら命令セットが違うと言っても、同じように命令をフェッチしてレジスタやメモリからデータを読んで演算してレジスタやメモリに書き戻すってのは同じ。
- だったら、同じ部分を抽象化して記述して、異なる命令セットなどはコンパイラに任せればいい。

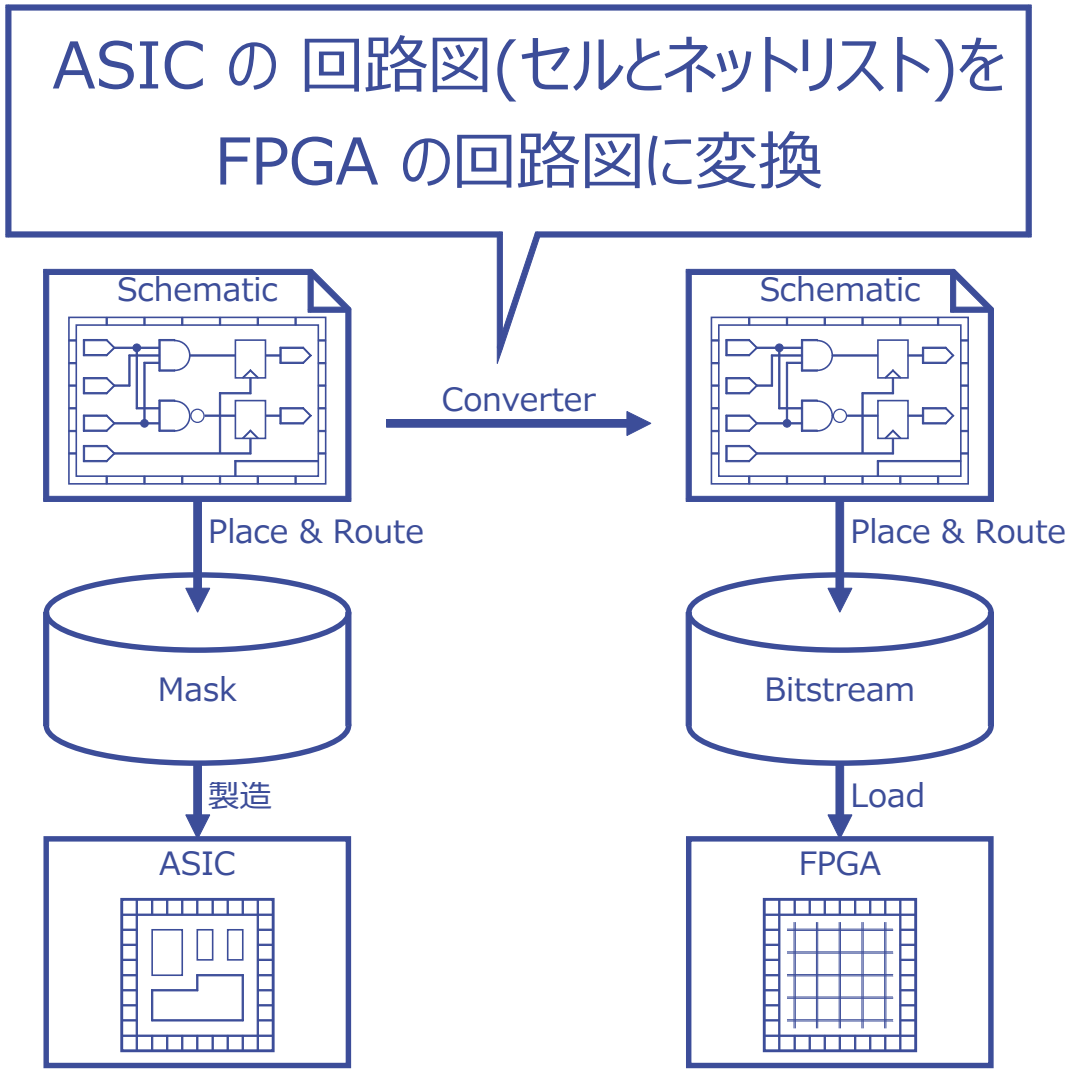
回路図による論理回路設計(暗黒?)時代



◇ 問題点

- 生産性
  - いちいちセルを並べるの？
    （面倒くせ~）
- 互換性
  - メーカー間(またはファミリー間でさえも)互換性無し
    （メーカーによる囲い込み）

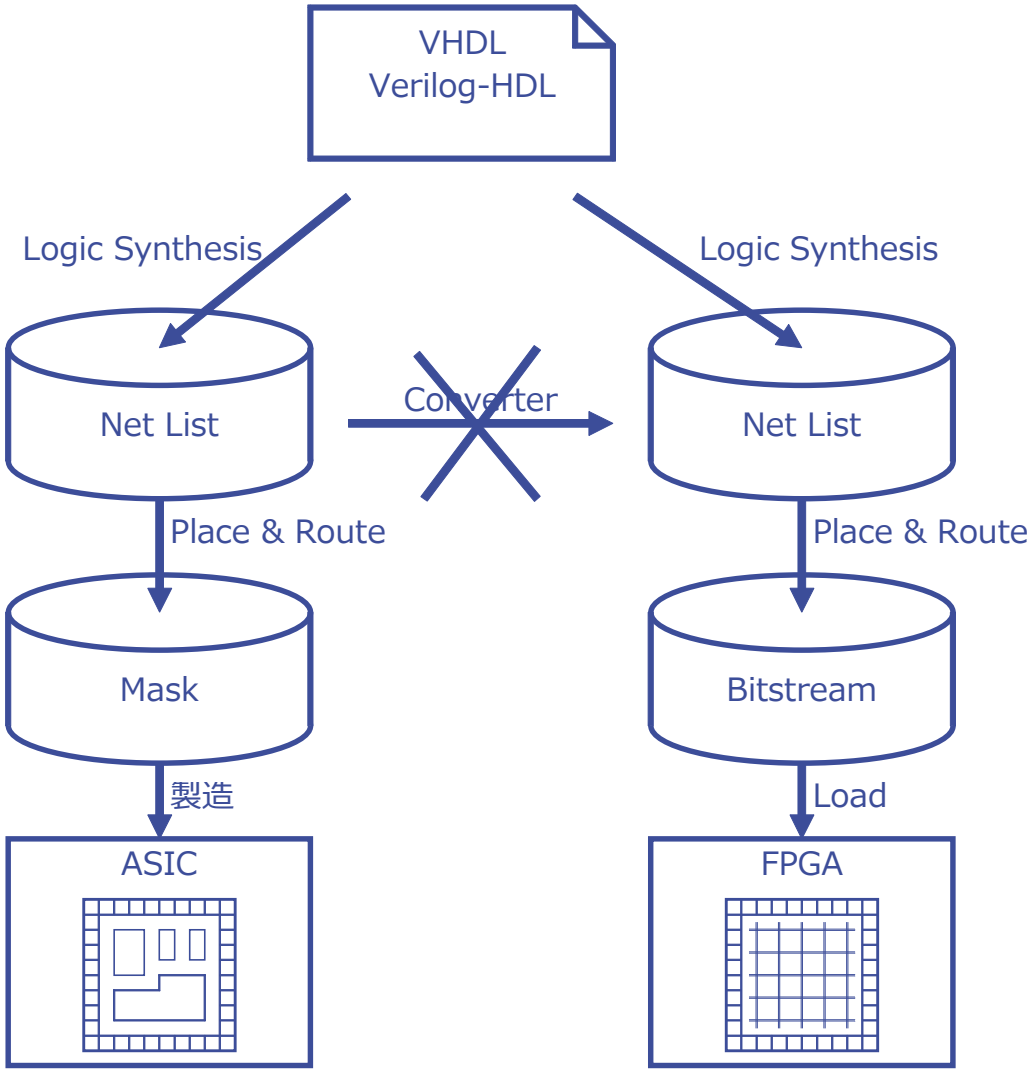
論理回路コンバーター



◇ 問題点

- 生産性
  - 回路図を書くことにはかわりない。
- 性能
  - 変換したコードは FPGA が持つ本来の性能を生かしきれなかった。

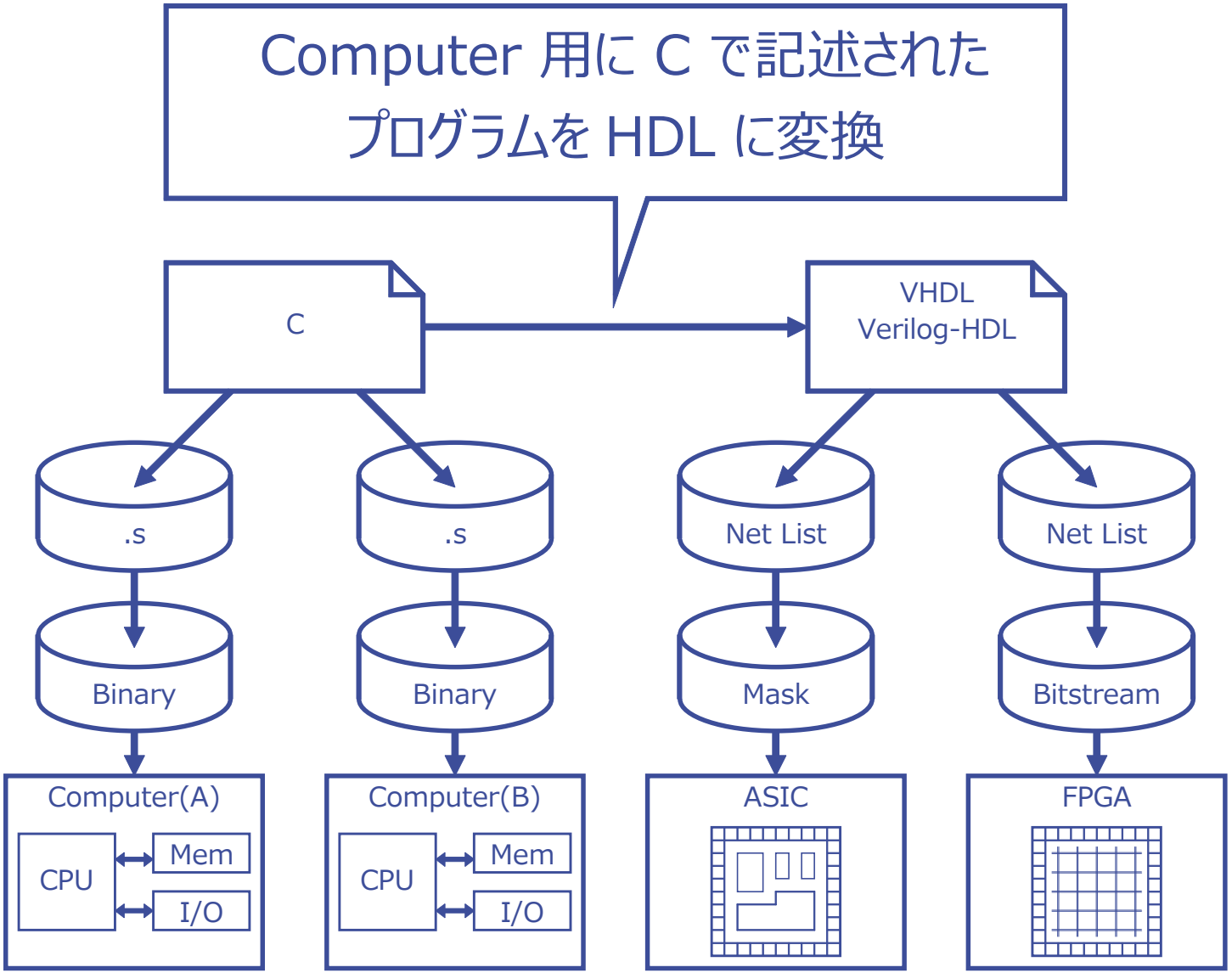
HDL(Hardware 記述言語)と論理合成による回路設計



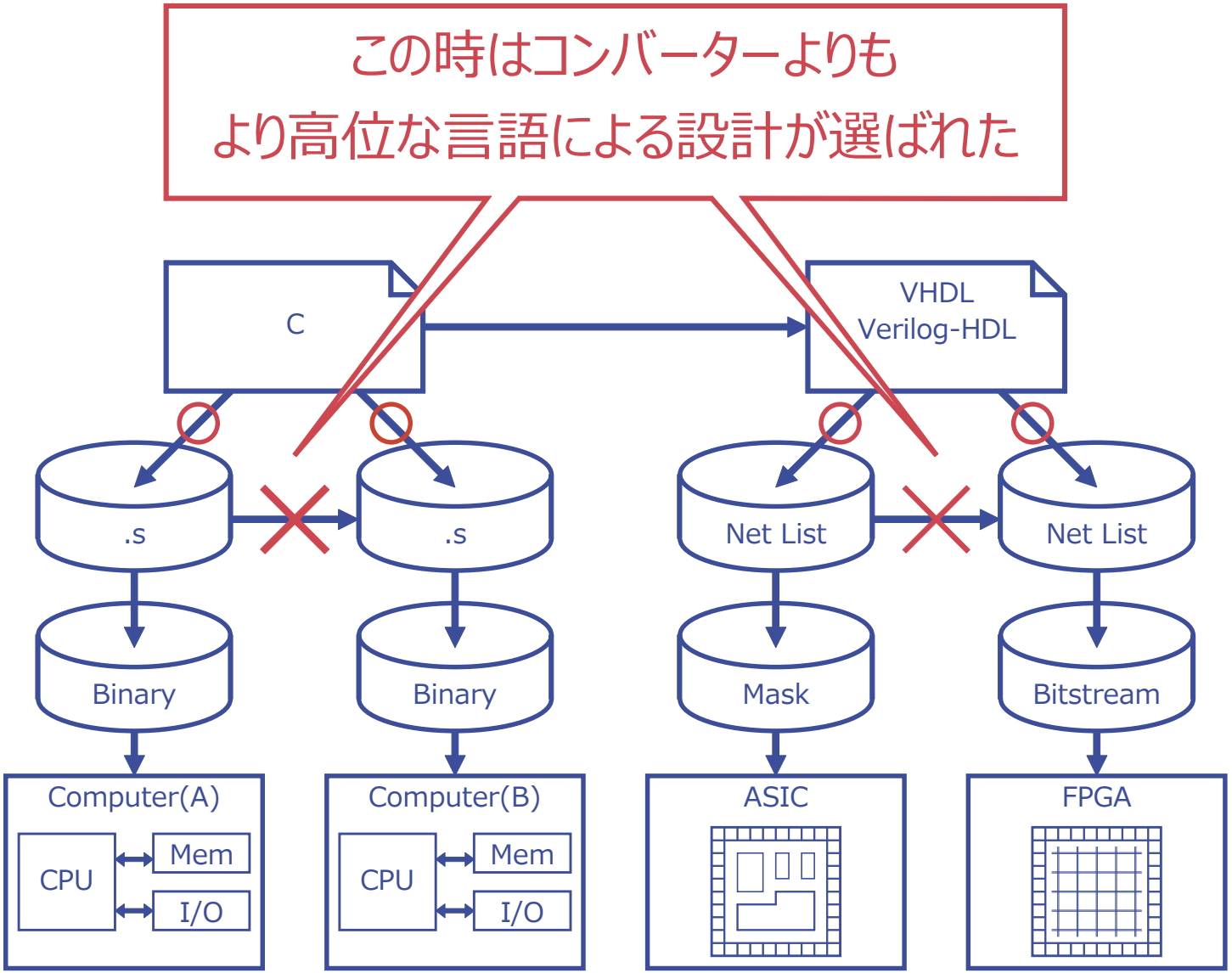
- いくらセルの種類が違うと言っても、同じようにレジスタがあって、レジスタ間の論理を記述するという点は同じ。いわゆる RTL(Register Transfer Level)。
- だったら、同じ部分を抽象化して記述して、異なるセルの選択などは論理合成に任せればいい。

今の話を少し

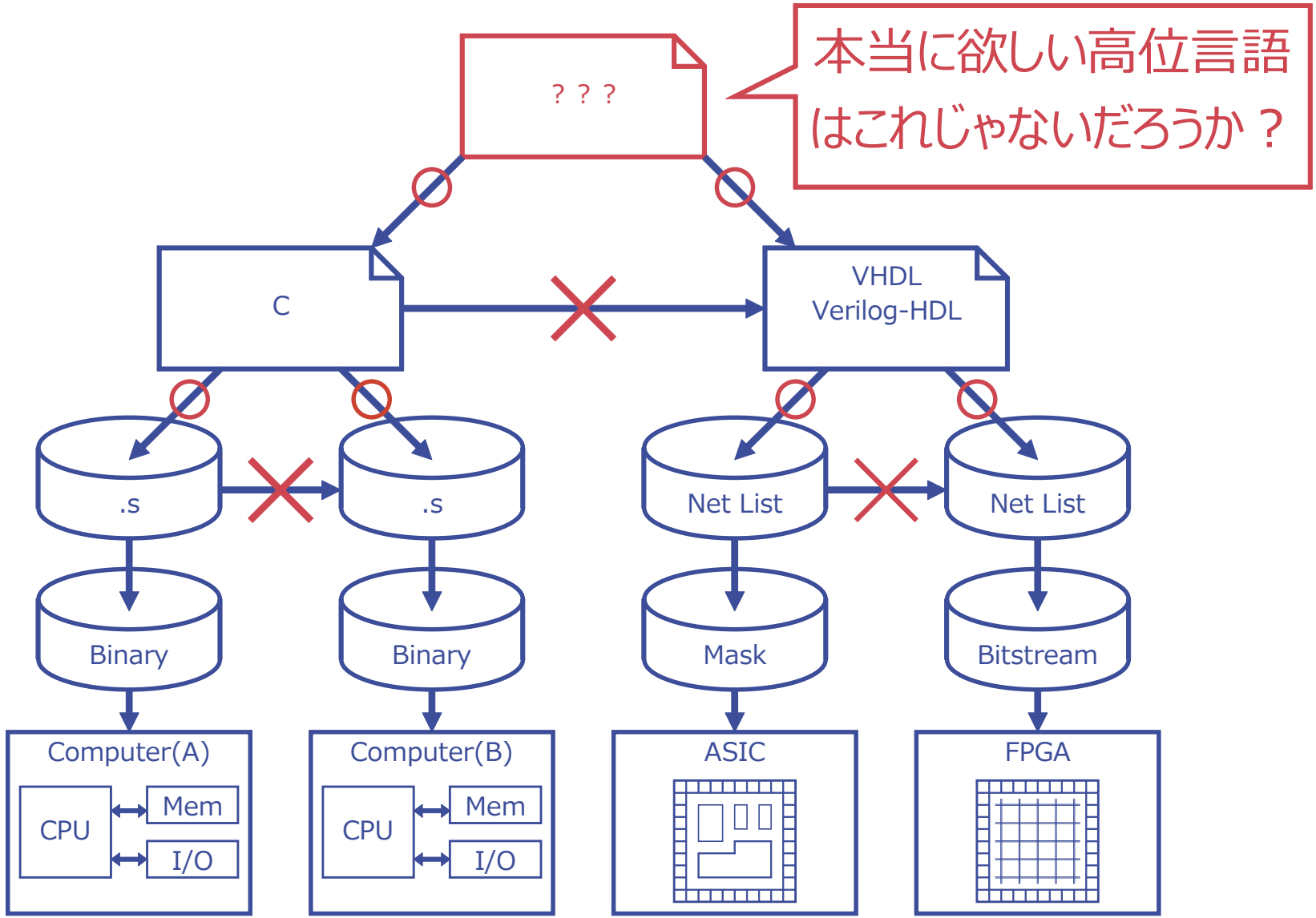
C 言語を使った論理回路設計



あ、それ見たことある　一昔前に一つ下の階層でやってた



本当に欲しい高位言語

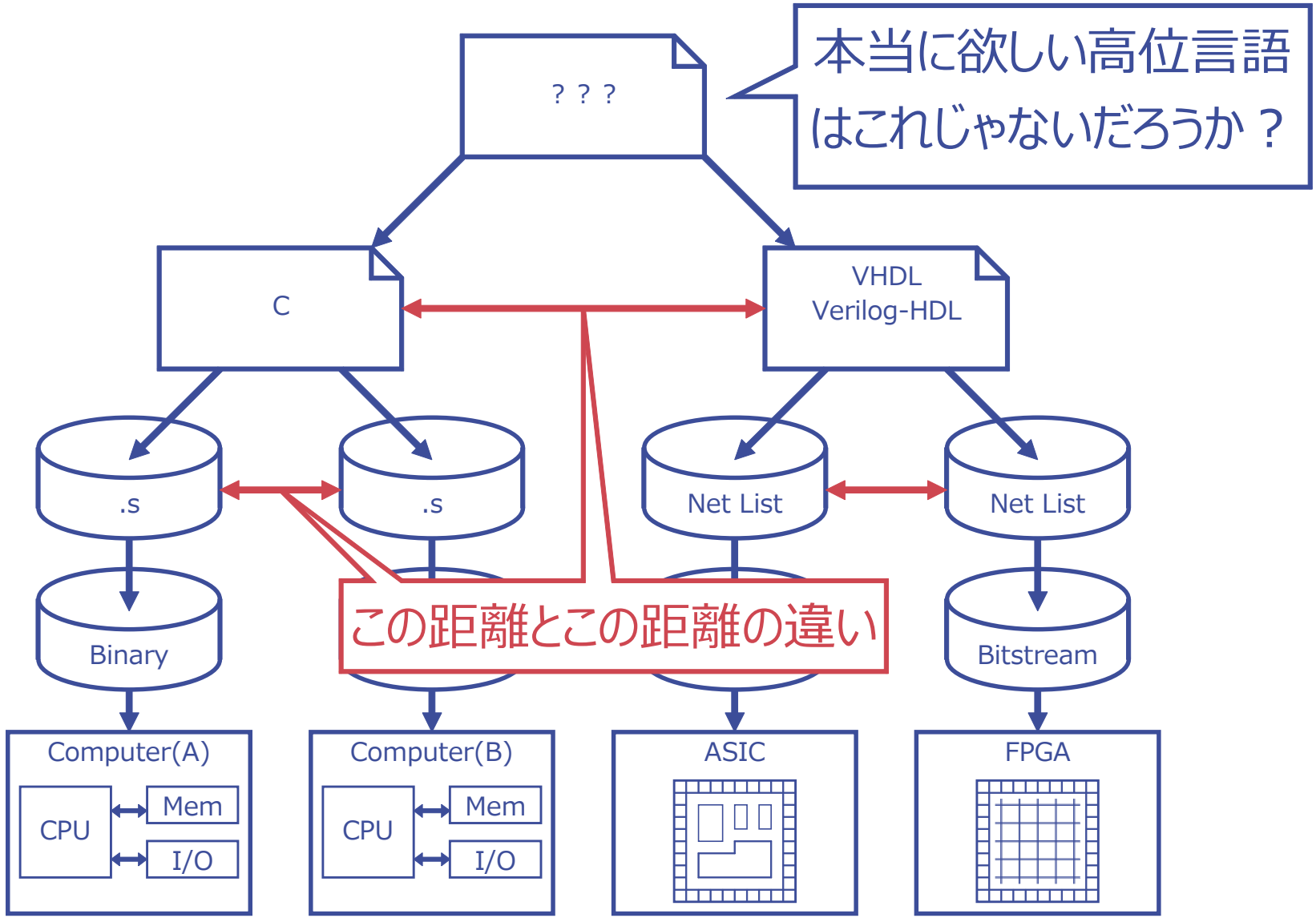

本当に欲しい高位言語
はこれじゃないだろうか？
？？？
C
VHDL
Verilog-HDL
.s
.s
Net List
Net List
Binary
Binary
Mask
Bitstream
Computer(A)
CPU
Mem
I/O
Computer(B)
CPU
Mem
I/O
ASIC
FPGA

昔は昔、今は今　－　C-HDL コンバーターの良い点と悪い点

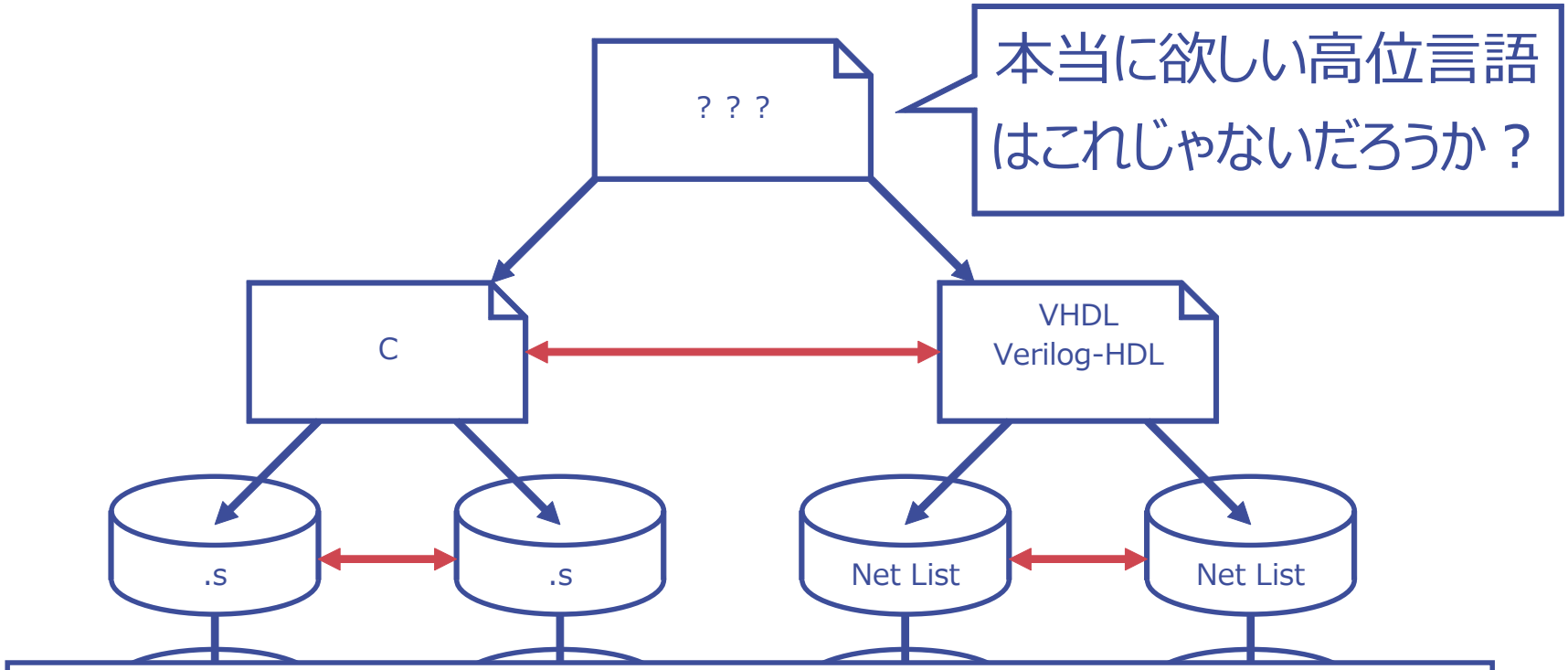
- 良い点（別に C 合成を dis ってるわけじゃない)
  - エンジニア数
    - HDL を書けるエンジニアに比べて C を書けるエンジニアの方が圧倒的に多い。
  - 生産性
    - HDL に比べて C の方が生産性は高い。

- 悪い点（実はコンバーターと同じ問題があるんじゃないか？）
  - 生産性
    - C は本当に生産性は良いのか？
    - Python、Ruby、Scala などもっと生産性の良い言語があるじゃないか？
  - 性能
    - 最初から HDL で書いたコードと比較してどうなの？

昔は昔、今は今　－　下位層と上位層で距離が違う

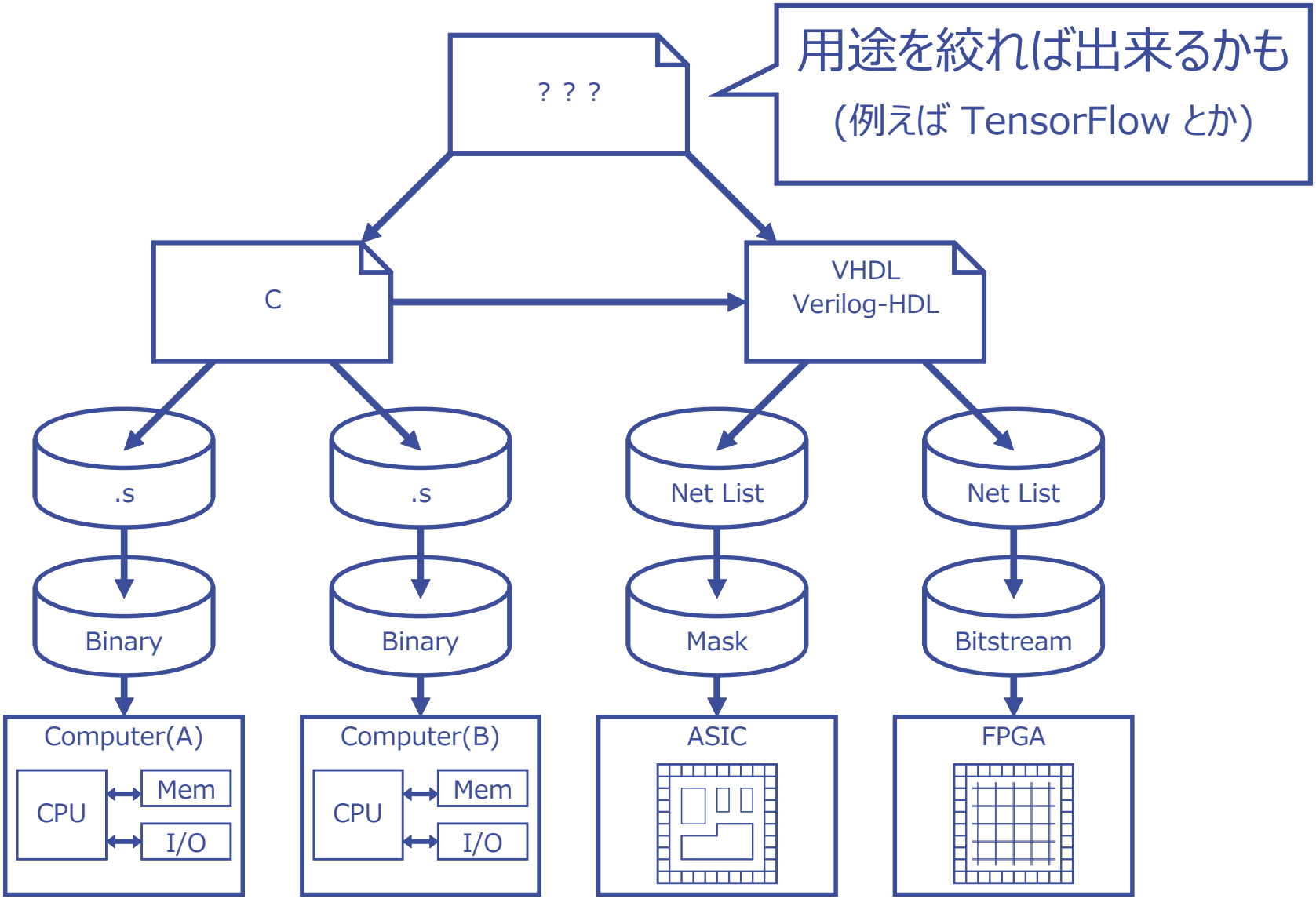


本当に欲しい高位言語　－　そうは言っても

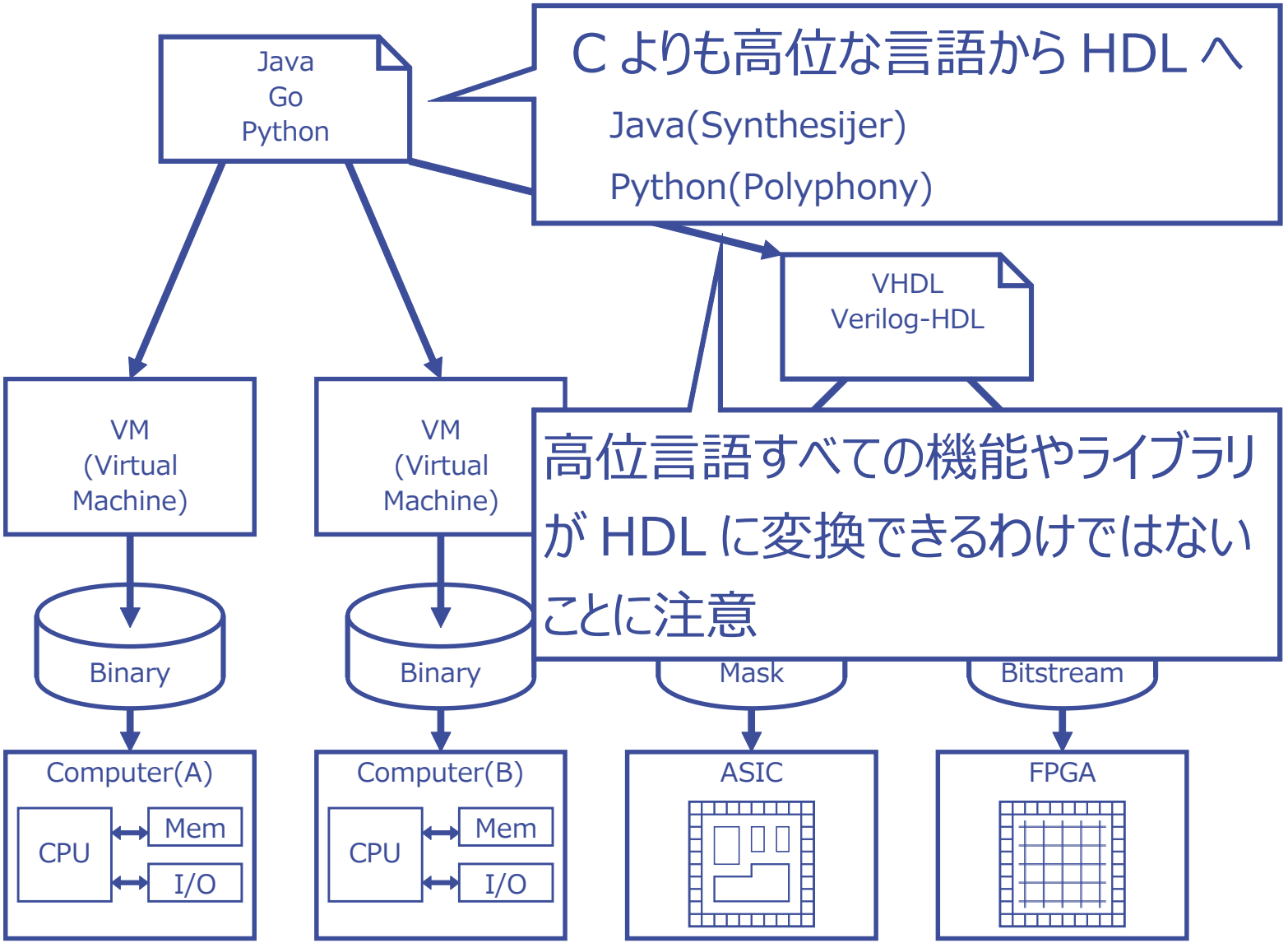


- 私には具体的なイメージがわかない
  - 共通点を抽象化して高位言語として昇華する必要
  - 下層の事を知りすぎた私には共通点よりも相違点のほうが目についてしまう

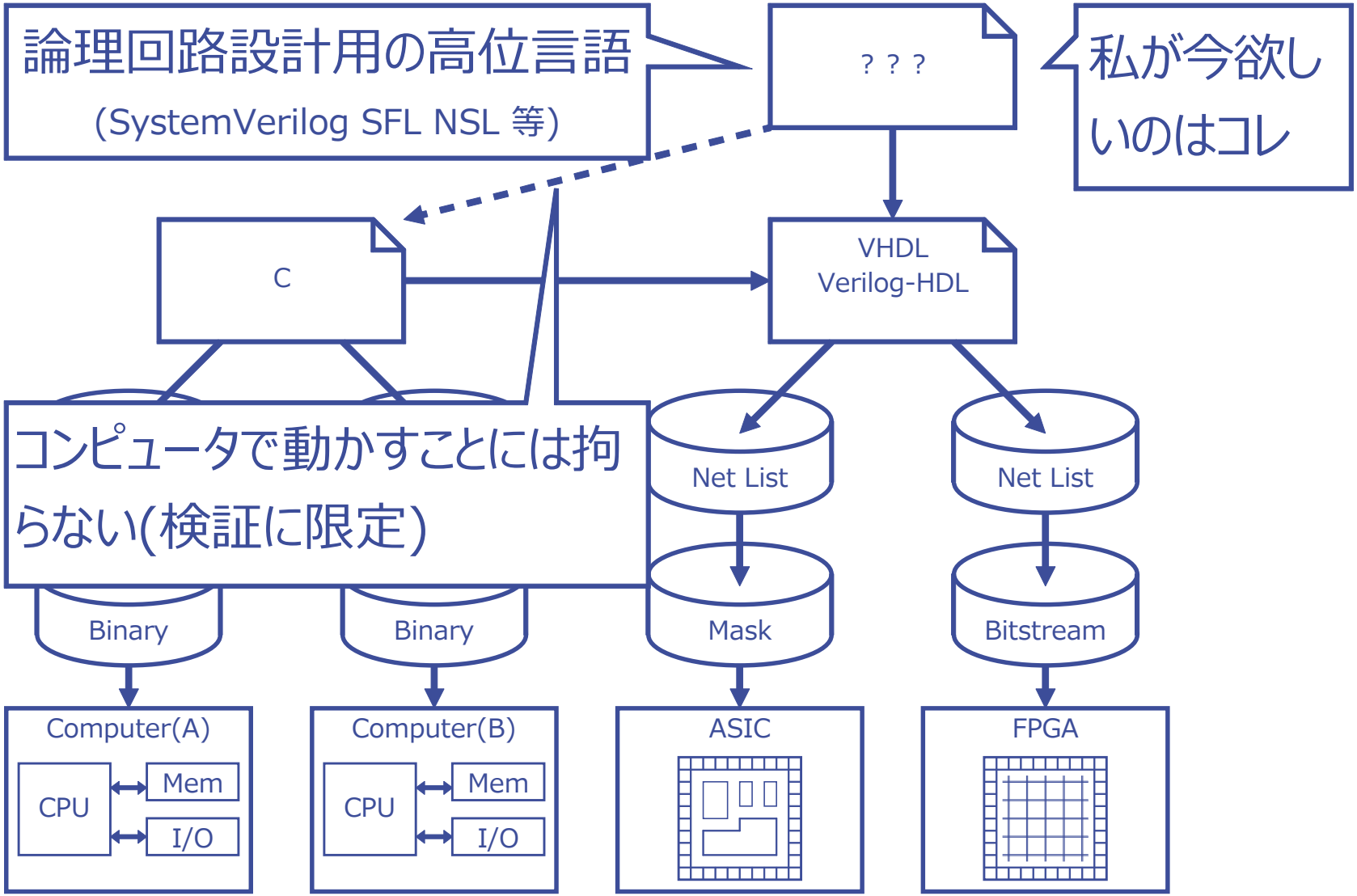
理想の高位言語　－　用途限定



理想の高位言語　－　C よりも高位なプログラミング言語をベース



理想の高位言語　－　論理回路設計専用



Iroha て何？

Iroha - FPGA エクストリーム・コンピューティング(2016/8/24)より抜粋

- Iroha: **I**ntermediate **R**epresentation **O**f **H**ardware **A**bstraction
- URL: https://github.com/nlsynth/iroha
- Author: Yusuke TABATA
- 高位合成の共通バックエンド(LLVM 的な物)を目指して
- 現状
  - Synthesijer と Neon Light が iroha 出力を実装
  - Neon Light の最適化コードを流用
- そのうち
  - フロントエンドの開発が楽になる
  - 最適化を改善して遊べるようになる
  - 高位合成友の会が盛り上がる
  - 海外の大学かどっかが潤沢なリソースで同じようなものを開発する...

Iroha の構造

```
(MODULE mod-id mod-name
    (PARAMS (PARAM name value) ...)
    (TABLE tab-id tab-name
        (REGISTERS
            (REGISTER reg-id reg-name {REGS/WIRE/CONST} type init-value)
                :
        )
        (RESOURCES
            (RESOURCE res-id res-name input-types output-types params)
                :
        )
        (STATE state-id
            (INSN inst-id res-name res-id res-op (state-id ...)(reg-id ...)(reg-id ...))
                :
        )
        :
    )
)
```

mod-id はファイル内でユニークな番号
mod-name は文字列

一つの MODULE に複数の TABLE
tab-id はモジュール内でユニークな番号
tab-name は文字列

一つの TABLE に複数の REGISTER
reg-id は TABLE 内でユニークな番号 reg-name は文字列 () でも可
type は現在 (INT width) か (UINT width)

一つの TABLE に複数の RESOURCE
res-id は TABLE 内でユニークな番号 res-name はリソース名
input-types と output-types は入出力レジスタの type

一つの STATE に複数の INST(Instruction)
inst-id は TABLE 内でユニークな番号
res-id は RESOURCE で宣言した番号
reg-id は REGISTER で宣言した番号

一つの TABLE に複数の STATE
state-id は TABLE 内でユニークな番号

Iroha の構造

- Lisp みたいな文法
- テーブル単位で レジスタ、リソース、ステートを管理
- (REGISTER ...)でテーブル内で使用するレジスタを宣言
  - reg-id は TABLE 内でユニークな番号。
  - reg-name は任意の文字列。() も可。
  - REGS/WIRE/CONST はクラス。
  - type は今のところ (UINT width) か (INT width) のみ。width はビット数。
  - init-value は今のところ整数。

Iroha の構造

- (RESOURCE ...) でテーブル内で使用するリソースを宣言
  - res-id は TABLE 内でユニークな番号。
  - res-name はリソース名。
  - 現時点で用意されているリソース
    - set　などのレジスタ間データ転送
    - add, sub, mul などの数値演算
    - eq, gt, gte などの比較演算
    - bit-and,bit-or,bit-inv,bit-xor,bit-sel,bit-concat,shift などのビット演算
    - ext-input, ext-output などの外部入出力ポート
    - channel-read, channel-write などのチャネル入出力
    - shared-reg, shared-reg-reader などの共有レジスタ
    - submodule-task, submodule-task-call, sibling-task, sibling-task-call などのタスク制御
  - その他、いろいろ追加予定？

Iroha の構造

- (STATE ...) でステート(状態)を定義して、そのステートで実行する命令を宣言

- (INST ...) でステートで実行する命令を宣言
  - 命令は次のようなものを指定する
    - 使用するリソースを res-id(リソース番号)で指定
    - リソースにデータを入力するレジスタを reg-id(レジスタ番号)で指定
    - リソースからの出力を保持するレジスタを reg-id(レジスタ番号)で指定
    - 次にどのステートに移行するかを state-id(ステート番号)で指定

iroha-ruby - Iroha を Ruby で扱うライブラリ

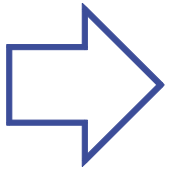
- Iroha を(私が)理解するためのトライアル的な位置づけ
- URL: https://github.com/ikwzm/iroha-ruby
- Iroha の各種構造を Ruby のクラスとして定義
  - MODULE → ./iroha-ruby/lib/iroha/iroha/i_module.rb
  - REGISTER → ./iroha-ruby/lib/iroha/iroha/i_register.rb
  - TABLE → ./iroha-ruby/lib/iroha/iroha/i_table.rb
  - 等々
- Parser (Iroha 形式を Ruby のクラスとして入力)
  - treetop(https://github.com/nathansobo/treetop) を使用。
  - Grammer → ./iroha-ruby/lib/iroha/builder/exp_parser.tt
- Writer (Ruby のクラスから Iroha 形式に出力)
  - 各クラス に_to_exp メソッドを定義

iroha-ruby - Iroha を Ruby で扱うライブラリ

- Simple Builder
  - 簡易的な Iroha 形式の構築ツール
  - Iroha からどんな HDL が生成されるかを見るために、Iroha を直接記述してみたい。しかし、Iroha ではリソースやレジスタをユニークな番号で管理しているので直接記述するのは面倒。そのため、簡易的な構築ツールを作ってみた。
  - 当初は本当に簡易的なものだったが、なんだか知らないうちに色々と拡張してしまった。
  - Ruby の内部 DSL
  - ./iroha-ruby/lib/iroha/builder/simple.rb
  - サンプル
    - ./iroha-ruby/examples/*.rb

Simple Builder の記述例

```
require_relative '../lib/iroha/builder/simple'
include Iroha::Builder::Simple
design = IDesign :design do
  IModule  :mod do
    ITable     :tab  do
      Wire          :req          => Unsigned(1)
      Register      :counter      => Unsigned(32) <= 0
      Constant      :one          => Unsigned(32) <= 1
      Constant      :done         => Unsigned(1) <= 1
      ExtInput      :data_in      => Unsigned(1)
      ExtOutput     :flow_out     => Unsigned(1) <= 0
      DataFlowIn    :flow_in      => Unsigned(1)
      IState        :state1
      IState        :state2
      IState        :state3
      state1.on {
        req <= data_in
        flow_in <= req
        Goto state2
      }
      state2.on {
        counter <= counter + one
        Goto state3
      }
      state3.on {
        flow_out <= done
      }
    end
  end
end
```



```
(MODULE 1 mod
 (TABLE 1 tab
  (REGISTERS
    (REGISTER 1 req  WIRE  (UINT  1) ())
    (REGISTER 2 counter REG   (UINT 32) 0 )
    (REGISTER 3 one  CONST (UINT 32) 1 )
    (REGISTER 4 done CONST (UINT  1) 1 )
  )
  (RESOURCES
    (RESOURCE 1 ext-input   () () (PARAMS  ..
    (RESOURCE 2 ext-output () () (PARAMS  ..
    (RESOURCE 3 dataflow-in ((UINT 1)) ()   ..
    (RESOURCE 4 tr () () (PARAMS))
    (RESOURCE 5 add ((UINT 32) (UINT 32))..
  )
  (INITIAL 1)
  (STATE 1
    (INSN 1 ext-input   1 () () () (1))
    (INSN 2 dataflow-in 3 () () (1) ())
    (INSN 3 tr          4 () (2) () ())
  )
  (STATE 2
    (INSN 4 add 5 () () (2 3) (2))
    (INSN 5 tr  4 () (3) () ())
  )
  (STATE 3
    (INSN 6 ext-output 2 () () (4) ())
  )
 )
)
```

Simple Builder の サンプル

- 作ったら使ってみたい

- というわけで、いくつか簡単な回路を書いてみた
  将来の Iroha への布石(Iroha で必要になるかもしれない演算)
  - div.rb　　- 整数除算器
  - fadd.rb　　- 浮動小数点加減算器
  - fmul.rb　　- 浮動小数点乗算器
  - fdiv.rb　　- 浮動小数点除算器

fadd.rb - 浮動小数点加減算器

- ./iroha-ruby/examples/fadd.rb
  (さすがにスライドじゃ入りきらない)

- Simple Builder 用に記述したソースを Ruby で Iroha に変換
  shell$ ruby ./examples/fadd.rb > ./examples/fadd.iroha

- Iroha を iroha で Verilog-HDL に変換
  shell$ iroha -v ./examples/fadd.iroha > ./test/fadd/src/fadd.v

- Vivado でシミュレーション
  ./test/fadd/Readme.md 参照

fmul.rb - 浮動小数点乗算器

- ./iroha-ruby/examples/fmul.rb
  (さすがにスライドじゃ入りきらない)

- Simple Builder 用に記述したソースを Ruby で Iroha に変換
  shell$ ruby ./examples/fmul.rb > ./examples/fmul.iroha

- Iroha を iroha で Verilog-HDL に変換
  shell$ iroha -v ./examples/fmul.iroha > ./test/fmul/src/fmul.v

- Vivado でシミュレーション
  ./test/fmul/Readme.md 参照

fdiv.rb - 浮動小数点除算器

- ./iroha-ruby/examples/fdiv.rb
  (さすがにスライドじゃ入りきらない)

- Simple Builder 用に記述したソースを Ruby で Iroha に変換
  shell$ ruby ./examples/fdiv.rb > ./examples/fdiv.iroha

- Iroha を iroha で Verilog-HDL に変換
  shell$ iroha -v ./examples/fdiv.iroha > ./test/fdiv/src/fdiv.v

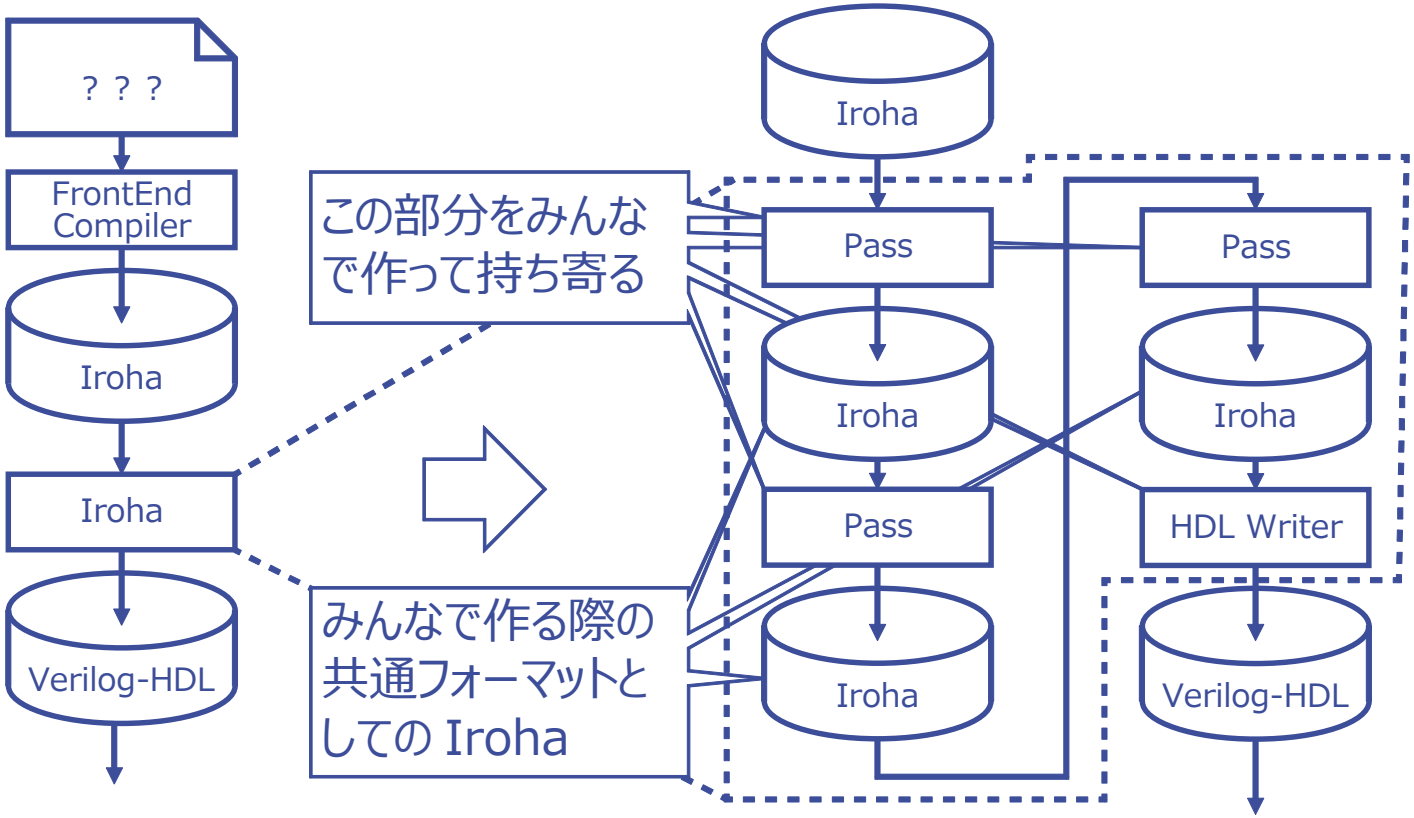
- Vivado でシミュレーション
  ./test/fdiv/Readme.md 参照

で、結局 Iroha って？

- どうやら ステート(状態)とリソースを色々と良きに計らってくれるらしい
  - Scheduling + Allocation
  - ステートの結合と分割
  - リソースの共通化(逐次処理)か分散化(並列処理)を選択
  - 抽象的なステート(状態)をデバイスに合わせて最適なクロック単位のステートに変換してくれる…ことはやってくれるのかな？
    現時点では不明、出来ればしてほしい(希望)

Iroha こうなってほしいな～(3 か月しか触っていないのにおこがましいけど)

- 現状、 最適化と HDL 出力は Tabata さん頼み
- みんなで弄れるようになれば高位合成友の会で盛り上がるかも



現状　　こうなったらいいな