# CodeSqueezer

ご紹介と使用の手引

シグナル・プロセス・ロジック株式会社

# *CodeSqueezer* とその特徴

*CodeSqueezer* は数式を Verilog HDL に変換する IP ツールです

数式は演算子を用いて記述します。例: x = (a + b) * (a – b)

特徴

- 数式のみの記述にも、言語形式での記述にも対応
- 整数、固定小数点数、浮動小数点数が利用できます
- 入力信号の仕様からその他の数値型を自動最適化
- タイミング制約の解決も自動的に行います
- 小規模な論理開発にも利用できる安価なツールです

# 例:二元連立方程式を解く

以下はソースコードの一例です（黄色の部分は必須、オレンジ色の部分はオプションです）

```
// Solution of dual simultaneous equations by Cramer's fomula
```

“//” 以降は行末まではコメントです。コメントは“/*”と“*/”の間に記述することもできます

```
#inport f1 man(1, 2, 0)
#inport f2 man(2, 3, 0)
#inport a1 man(1, 2, 0)
#inport a2 man(2, 3, 0)
#inport b1 man(3, 4, 0)
#inport b2 man(4, 5, 0)
#nonstop
```

#で始まる行はコンパイラ指示で、省略可能です
（コンパイル時に対話形式で指定することもできます）

入力信号の数値型を与えます （詳細は後述します）
manは仮数部（整数の場合は仮数部のみを与えます
(1, 2, 0) は、最小値1、最大値2、定数部0 を意味します

#nonstopを指定しておくと“compile” ボタンで直ちにコンパイルします

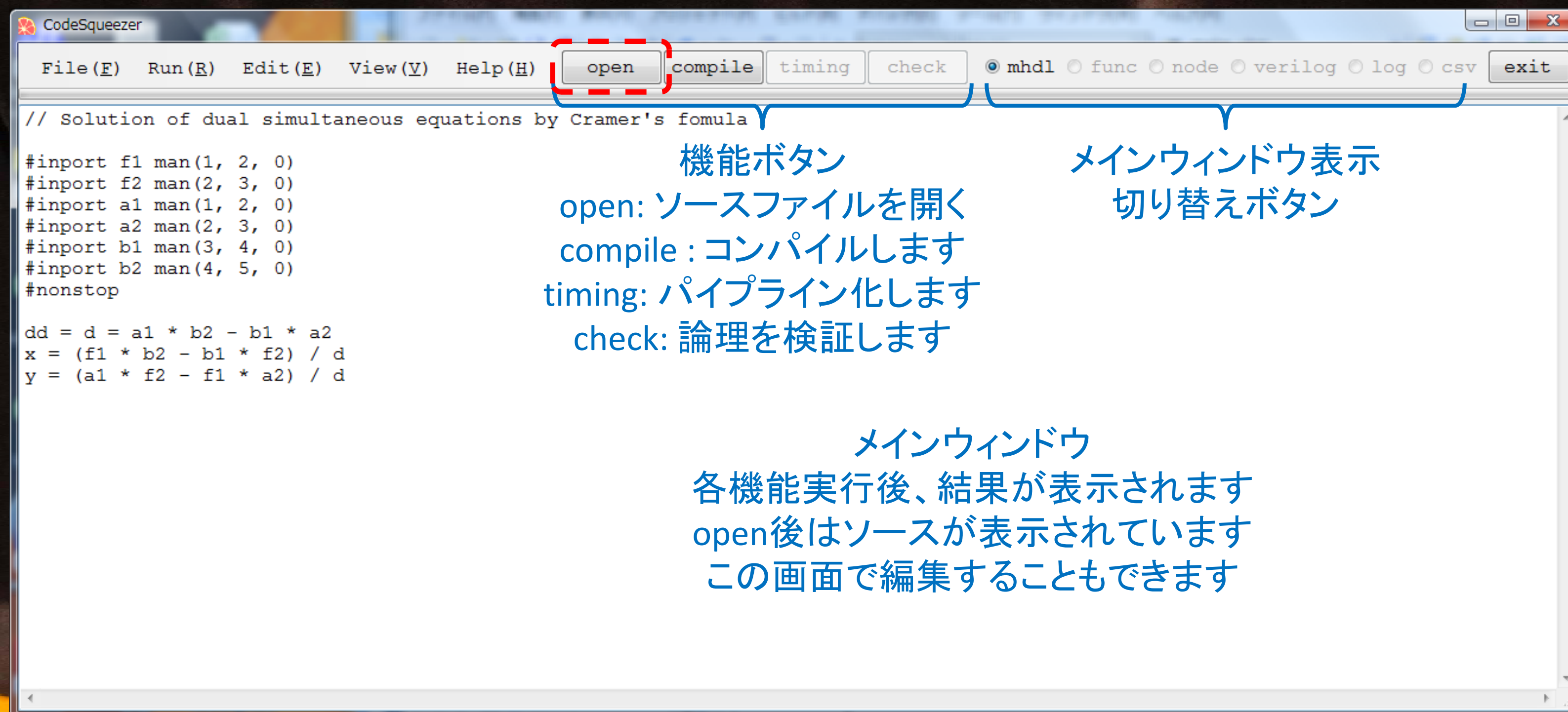
```
dd = d = a1 * b2 - b1 * a2
x = (f1 * b2 - b1 * f2) / d
y = (a1 * f2 - f1 * a2) / d
```

数式を記述します
左辺にのみ現れる名前は出力、
右辺にのみ現れる名前は入力となります
“dd =” は、中間結果を出力するために指定しています

必須な記述はこれだけです

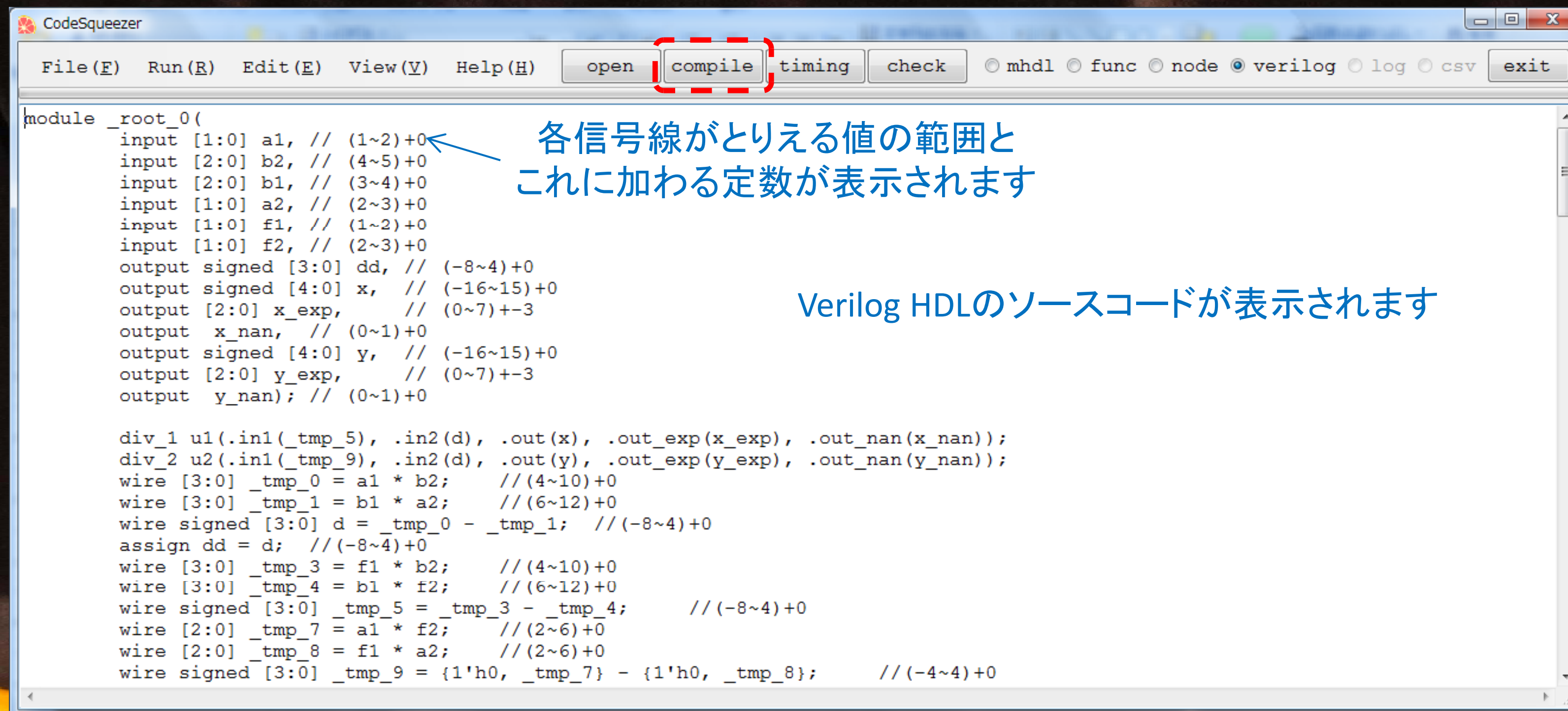
# CodeSqueezer のメイン画面

CodeSqueezer を起動するとメイン画面が表示されます
まず、"open" ボタンによりソースファイルを開きます



# コンパイル結果（非同期コード）

**compile** ボタンを押すとコンパイルが実行されます
メインウィンドウには変換結果（非同期コード）が表示されます



# コンパイルエラーの表示

エラーがある場合は、エラーウィンドウが開きエラーの内容が表示されます

CodeSqueezer

File(F) Run(R) Edit(E) View(V) Help(H) open compile timing check mhdl func node verilog log csv exit

```
line(10): illegal unary operator(/)
    line(10): illegal form
```

エラー表示をダブルクリックすると
該当する行が強調表示されます

エラーウィンドウ

```
// Solution of dual simultaneous equations by Cramer's fomula

#inport f1 man(1, 2, 0)
#inport f2 man(2, 3, 0)
#inport a1 man(1, 2, 0)
#inport a2 man(2, 3, 0)
#inport b1 man(3, 4, 0)
#inport b2 man(4, 5, 0)
#nonstop

dd = d = a1 * b2 - b1 * / a2
x = (f1 * b2 - b1 * f2) / d
y = (a1 * f2 - f1 * a2) / d
```

ここを修正すればOK

メインウィンドウ

# チェッカー（簡易検証機能）

check ボタンを押すとテストベクターを自動生成して検証が行われます

自動生成された
テストベクター

出力

CodeSqueezer

File(F) Run(R) Edit(E) View(V) Help(H)

open compile timing check mhdl func node verilog log csv exit

| a1+0 | b2+0 | b1+0 | a2+0 | f1+0 | f2+0 | dd+0 | x+0 | x exp+-3 | (x) | x nan+0 | y+0 | y exp+-3 | (y) | y nan+0 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 4 | 3 | 2 | 1 | 2 | -2 | 8 | -3 | 1 | 0 | 0 | -3 | 0 | 0 |
| 2 | 4 | 3 | 2 | 1 | 2 | 2 | -8 | -3 | -1 | 0 | 8 | -3 | 1 | 0 |
| 1 | 5 | 3 | 2 | 1 | 2 | -1 | 8 | -3 | 1 | 0 | 0 | -3 | 0 | 0 |
| 2 | 5 | 3 | 2 | 1 | 2 | 4 | -2 | -3 | -0.25 | 0 | 4 | -3 | 0.5 | 0 |
| 1 | 4 | 4 | 2 | 1 | 2 | [illegible] | 8 | -3 | 1 | 0 | 0 | -3 | 0 | 0 |
| 2 | 4 | 4 | 2 | 1 | 2 | 0 | -16 | 0 | -16 | 1 | 15 | 0 | 15 | 1 |
| 1 | 5 | 4 | 2 | 1 | 2 | -3 | 8 | -3 | 1 | 0 | 0 | -3 | 0 | 0 |
| 2 | 5 | 4 | 2 | 1 | 2 | 2 | -12 | -3 | -1.5 | 0 | 8 | -3 | [illegible] | 0 |
| 1 | 4 | 3 | 3 | 1 | 2 | -5 | 4 | -3 | 0.5 | 0 | 2 | -3 | 0.25 | 0 |
| 2 | 4 | 3 | 3 | 1 | 2 | -1 | [illegible] | [illegible] | 2 | 0 | -8 | -3 | -1 | 0 |
| 1 | 5 | 3 | 3 | 1 | 2 | -4 | 2 | -3 | 0.25 | 0 | 2 | -3 | 0.25 | 0 |
| 2 | 5 | 3 | 3 | 1 | 2 | 1 | -8 | -3 | -1 | 0 | 8 | -3 | 1 | 0 |
| 1 | 4 | 4 | 3 | 1 | 2 | -8 | 4 | -3 | 0.5 | 0 | 1 | -3 | 0.125 | 0 |
| 2 | 4 | 4 | 3 | 1 | 2 | -4 | 8 | -3 | 1 | 0 | -2 | -3 | -0.25 | 0 |
| 1 | 5 | 4 | 3 | 1 | 2 | -7 | 4 | -3 | 0.5 | 0 | 2 | -3 | 0.25 | 0 |
| 2 | 5 | 4 | 3 | 1 | 2 | -2 | 12 | -3 | 1.5 | 0 | -4 | -3 | -0.5 | 0 |
| 1 | 4 | 3 | 2 | 2 | 2 | -2 | -8 | -3 | -1 | 0 | 8 | -3 | 1 | 0 |
| 2 | 4 | 3 | 2 | 2 | 2 | 2 | 8 | -3 | 1 | 0 | 0 | -3 | 0 | 0 |
| 1 | 5 | 3 | 2 | 2 | 2 | -1 | -16 | -2 | -4 | 0 | 8 | -2 | 2 | 0 |
| 2 | 5 | 3 | 2 | 2 | 2 | 4 | 8 | -3 | 1 | 0 | 0 | -3 | 0 | 0 |
| 1 | 4 | 4 | 2 | 2 | 2 | -4 | 0 | -3 | 0 | 0 | 4 | -3 | 0.5 | 0 |
| 2 | 4 | 4 | 2 | 2 | 2 | 0 | 0 | -3 | 0 | 0 | 0 | -3 | 0 | 0 |

xの仮数部

xの指数部

()を付した列は
仮数部と指数部を
合成した結果です

タイトル表示は
“ポート名＋定数部”

# エクセルを用いた検証

**check** 結果は **.csv** ファイルにも出力されます
これを利用して **Excel** で演算結果が確認できます

ワークシート上での
計算結果（式を挿入）

検算：誤差/2指数部
絶対値が1以下なら正常

ゼロによる除算で
nanが立っています

| | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | a1+0 | b2+0 | b1+0 | a2+0 | f1+0 | f2+0 | dd+0 | [dd] | x+0 | x_exp+-3 | (x) | [x] | [dx] | x_nan+0 | y+0 | y_exp+-3 | (y) | [y] | [dy] | y_nan+0 |
| 2 | 1 | 4 | 3 | 2 | 1 | 2 | -2 | -2 | 8 | -3 | 1 | 1 | 0 | 0 | 0 | -3 | 0 | 0 | 0 | 0 |
| 3 | 2 | 4 | 3 | 2 | 1 | 2 | 2 | 2 | -8 | -3 | -1 | -1 | 0 | 0 | 8 | -3 | 1 | 1 | 0 | 0 |
| 4 | 1 | 5 | 3 | 2 | 1 | 2 | -1 | -1 | 8 | -3 | 1 | 1 | 0 | 0 | 0 | -3 | 0 | 0 | 0 | 0 |
| 5 | 2 | 5 | 3 | 2 | 1 | 2 | 4 | 4 | -2 | -3 | -0.25 | -0.25 | 0 | 0 | 4 | -3 | 0.5 | 0.5 | 0 | 0 |
| 6 | 1 | 4 | 4 | 2 | 1 | 2 | -4 | -4 | 8 | -3 | 1 | 1 | 0 | 0 | 0 | -3 | 0 | 0 | 0 | 0 |
| 7 | 2 | 4 | 4 | 2 | 1 | 2 | 0 | 0 | -16 | 0 | -16 | #DIV/0! | #DIV/0! | 1 | 15 | 0 | 15 | #DIV/0! | #DIV/0! | 1 |
| 8 | 1 | 5 | 4 | 2 | 1 | 2 | -3 | -3 | 8 | -3 | 1 | 1 | 0 | 0 | 0 | -3 | 0 | 0 | 0 | 0 |
| 9 | 2 | 5 | 4 | 2 | 1 | 2 | 2 | 2 | -12 | -3 | -1.5 | -1.5 | 0 | 0 | 8 | -3 | 1 | 1 | 0 | 0 |
| 10 | 1 | 4 | 3 | 3 | 1 | 2 | -5 | -5 | 4 | -3 | 0.5 | 0.4 | 0.8 | 0 | 2 | -3 | 0.25 | 0.2 | 0.4 | 0 |
| 11 | 2 | 4 | 3 | 3 | 1 | 2 | -1 | -1 | 8 | -2 | 2 | 2 | 0 | 0 | -8 | -3 | -1 | -1 | 0 | 0 |
| 12 | 1 | 5 | 3 | 3 | 1 | 2 | -4 | -4 | 2 | -3 | 0.25 | 0.25 | 0 | 0 | 2 | -3 | 0.25 | 0.25 | 0 | 0 |
| 13 | 2 | 5 | 3 | 3 | 1 | 2 | 1 | 1 | -8 | -3 | -1 | -1 | 0 | 0 | 8 | -3 | 1 | 1 | 0 | 0 |
| 14 | 1 | 4 | 4 | 3 | 1 | 2 | -8 | -8 | 4 | -3 | 0.5 | 0.5 | 0 | 0 | 1 | -3 | 0.125 | 0.125 | 0 | 0 |

# タイミング制約の解決 （同期コード）

**timing** ボタンを押すと、必要に応じてレジスタを挿入し、タイミング制約を解決します（論理遅延を制限以下とし、演算器入力のタイミングを調整します）

CodeSqueezer

File(F) Run(R) Edit(E) View(V) Help(H) | open | compile | timing | check | mhdl | func | node | verilog | log | csv | exit

```
module _root_0(
        input [1:0] a1, // (1~2)+0
        input [2:0] b2, // (4~5)+0
        input [2:0] b1, // (3~4)+0
        input [1:0] a2, // (2~3)+0
        input [1:0] f1, // (1~2)+0
        input [1:0] f2, // (2~3)+0
        input clock,
        output signed [3:0] dd, // (-8~4)+0 @(0, 2)
        output signed [4:0] x,  // (-16~15)+0 @(6, 1)
        output [2:0] x_exp,     // (0~7)+-3 @(6, 0)
        output reg x_nan,       // (0~1)+0 @(6, 0)
        output signed [4:0] y,  // (-16~15)+0 @(6, 1)
        output [2:0] y_exp,     // (0~7)+-3 @(6, 0)
        output reg y_nan);      // (0~1)+0 @(6, 0)

        div_1 u1(.in1(_tmp_5), .in2(d), .clock(clock), .out(x), .out_exp(x_exp), .out_nan(x_nan____));
        div_2 u2(.in1(_tmp_9), .in2(d), .clock(clock), .out(y), .out_exp(y_exp), .out_nan(y_nan____));
        wire  x_nan____;        // (0~1)+0 @(1, 1)
        wire  y_nan____;        // (0~1)+0 @(1, 1)
        wire [3:0] _tmp_0 = a1 * b2;    //(4~10)+0 @(0, 1)
        wire [3:0] _tmp_1 = b1 * a2;    //(6~12)+0 @(0, 1)
        wire signed [3:0] d = _tmp_0 - _tmp_1;  //(-8~4)+0 @(0, 2)
        assign dd = d;  //(-8~4)+0 @(0, 2)
        wire [3:0] _tmp_3 = f1 * b2;    //(4~10)+0 @(0, 1)
        wire [3:0] _tmp_4 = b1 * f2;    //(6~12)+0 @(0, 1)
        wire signed [3:0] _tmp_5 = _tmp_3 - _tmp_4;     //(-8~4)+0 @(0, 2)
```

クロック信号入力ポートが形成されています

タイミング情報がコメント部に追加されます

@(0, 2) は、
- クロック周期の遅れ（レイテンシ）が0
- クロックに対する遅れ（論理遅延）が2

であることを意味します（時間は任意単位）

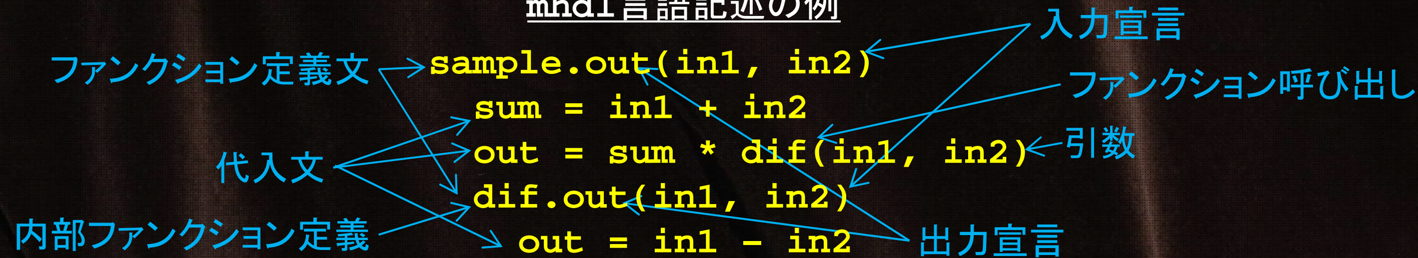
必要個所にはレジスタが挿入されます

クロック同期コードも記述されます

# mhdl言語の概要

ソースファイルは **mhdl** 言語仕様に従って記述します
**mhdl** の文には代入文とファンクション定義文とがあります

mhdl言語記述の例




段差付けによりファンクション定義文の範囲を示します
**mhdl** 言語は、信号線(リンク)を抽象化して扱います
信号線の型はコンパイル時に与えます

# 代入文のみの記述

ファンクションの外部に代入文が記述された場合には
ファンクション“_root”が全てを含むと想定されます

“_root”のファンクション定義文は自動生成されます
“_root”の入出力は以下の規則により生成されます

入力：代入文右辺に現れるリンクのうち、左辺に現れないもの
出力：代入文左辺に現れるリンクのうち、右辺に現れないもの

# mhdl の演算子

## 二項演算子

*, /, +, -: 加減乗除の演算子です
<, <=, !=, ==, >=, >: 比較演算子です
&, !, |: 論理積、排他論理和、論理和の演算子です

## 単項演算子

-: 符号を反転します
!: 論理値を反転します

## その他の演算子

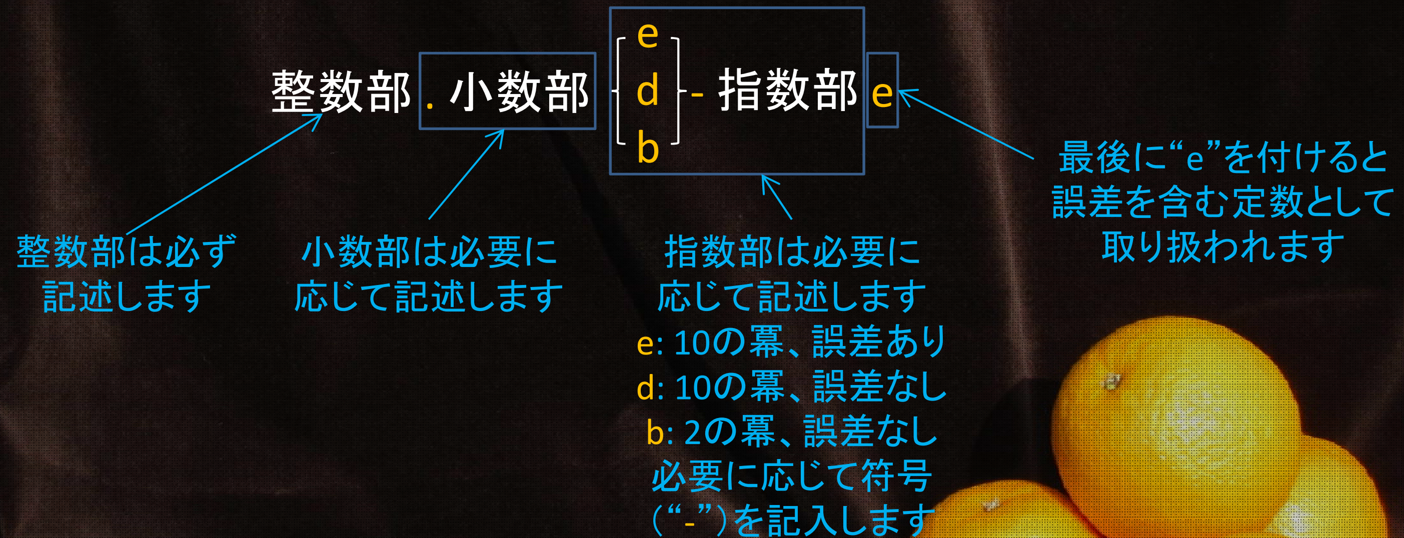
a ? b : c: aが真ならb、偽ならcをとります
(, ): 括弧内が優先して演算されます

### 演算子とその優先順位

| 優先度 | 演算子 | 演算内容 | 項数 |
|---|---|---|---|
| 高い | ＊、/ | 乗除算 | 任意 |
| ↑ | ＋、− | 加減算 | 任意 |
| | ＜、＜＝、！＝、＝＝、＞、＞＝ | 比較演算 | 2項限定 |
| | & | 論理積 | 任意 |
| | ! | 排他論理和 | 任意 |
| ↓ | \| | 論理和 | 任意 |
| 低い | ？、: | 条件式 | 3項限定 |

# mhdl の定数記述

mhdlではリンク名が記述される部分に定数を記述することも可能です



# 標準ファンクション

- 信号の加工
  - reduce(in) : inの有効桁数を一つ減じたものを出力します
  - to_int(in) : inを整数に変換します。オーバーフローしない充分なビット幅がとられます
  - to_fix(in1, in2) : in1をin2の型と同じ固定小数点数に変換します。in2の指数部は定数でなければなりません
    整数のビット幅を制限する場合にも to_fix をご利用ください。オーバーフローが生じる場合はフラグがセットされます
  - clip(in) : inがオーバーフローした場合、最大値または最小値を返します。数値化不能異常の場合は値を0に固定します
    異常フラグは解除されます
- 異常の検出
  - of(in)／uf(in)／nan(in) : inがオーバーフロー／アンダーフロー／数値化不能異常であるとき1を返します
- 数値の論理化
  - positive(in)／negative(in) : inが正／負である場合に1となります。検出限界以下の場合は0を返します
  - nonzero(in)／undetectable(in) : inが0でない場合／0または検出限界以下である場合に1となります
- 検出限界の設定
  - 検出限界は誤差（ノイズなど）とみなされるビット数（noise bits）で指定します
  - noise bitsの設定は、プルダウンメニューの Edit → Environ でポップアップする Environ Editorで行います
  - 出荷状態では noise bits は0となっています（誤差は仮数部の1未満であるとみなします）

# 信号線の数値表現方法（リンクとワイヤの対応）

信号線（リンク）は5種類のワイヤ（Verilog HDLのwireに対応）で構成されます
- **man**： 仮数部。必須のワイヤで、整数を表します
- **exp**： 指数部。このワイヤがある場合は、仮数部の値に$2^{exp}$が乗じられます
- **of**： オーバーフローフラグ （フラグの値は0または1のみをとります）
- **uf**： アンダーフローフラグ
- **nan**： 数値化不能フラグ （of&ufで代用することもできます）

ツール上ではワイヤ属性を“最小値、最大値、定数”の組合せで表現します
- 最小値～最大値を表現するに必要十分なビット幅のワイヤを形成します
- 最小値が負の場合は符号付き整数、それ以外は符号なし整数となります
- 定数部分はツール内で処理され、Verilog HDLコードは形成しません

リンクには誤差の有無を表す属性が与えられています
- 誤差のある信号に対しては、有効桁のみを演算するコードを形成します

# コンパイラ指示子

“#”で始まる行はコンパイラに対する指示で、以下の種類があります

#entry ファンクション名：コンパイルを行う最上位のファンクションを指定します
#inport 型指定子：入力の型を指定します。
“型指定子”には以下の記述を必要に応じて記述します
man(最小値， 最大値， 定数部)：仮数部。必ず記述します
exp(最小値， 最大値， 定数部)：指数部。指数部があれば記述します
of：オーバーフローフラグがある場合に記述します
uf：アンダーフローフラグがある場合に記述します
nan：数値化不能誤差フラグがある場合に記述します
e：信号線の数値が誤差を含む場合に記述します
#nonstop：これがある場合は“compile”ボタンですぐにコンパイルを開始します
この指定がない場合はコンパイル開始前にエントリーエディタが開きます

# エントリーエディタ

コンパイル開始の際にエントリーエディタが起動します（#nonstopディレクティブがない場合）
エントリーエディタでは、コンパイルするファンクションと入力信号型を指定します

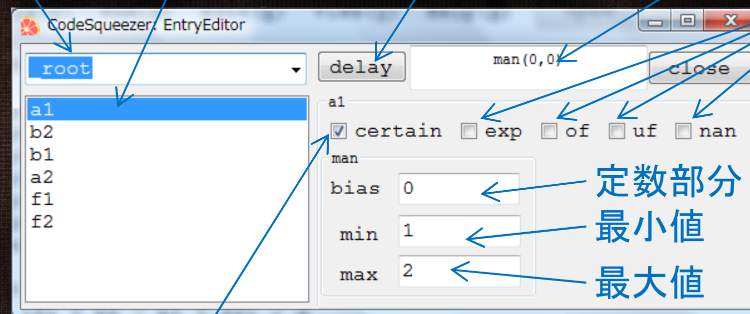
コンパイルする
ファンクションを
選択します

型を設定する
入力ポートを
選択します

このボタンで
遅延条件が
設定できます

各信号線の
遅延条件が
表示されます

必要な信号線に
チェックを入れます




定数部分

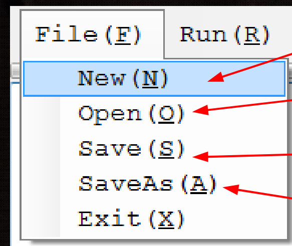
最小値

最大値

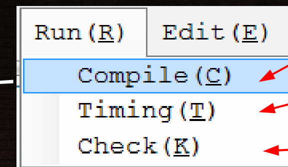
誤差を含まない
ことを指定します

# プルダウンメニュー

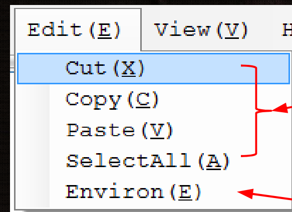
プルダウンメニューで、すべての操作を行うことができます




新規にソースを作成します
既存のソースファイルを開きます
ファイルに上書き保存します
別のファイルに保存します

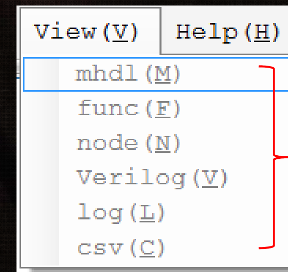



コンパイルします
タイミング制約を解決します
チェッカーを起動します

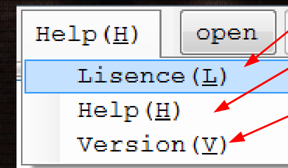



テキストの編集コマンドです
ショートカットキーも使用できます

各種処理条件を設定します
次頁参照




表示を切り替えます




ライセンス条件を表示します
ヘルプ画面を表示します
バージョン情報を表示します

# 処理条件の設定 （Edit→Environ）

条件をファイルから読み込みます

条件をファイルにセーブします

条件を変更せずに終了します

条件を変更して終了します

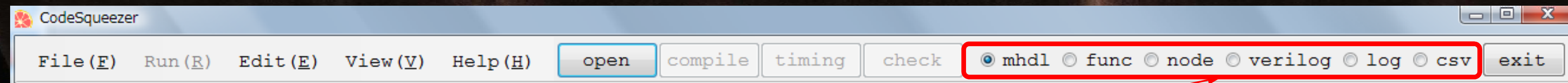
起動時にファイル“environ.env”から読み込みます

CodeSqueezer: EnvironEditor

Read　Save　Cancel　Set

| 項目 | 値 | 説明 |
| --- | --- | --- |
| limit logic delay | 4 | 最大許容論理遅延時間（遅延時間の単位は任意です） |
| add-sub logic delay | 1 | 加減算の論理遅延時間 |
| always multiply by logic | ☐ | 乗算論理を形成する（ハードウエア乗算器を使わない） |
| multiply logic delay | 1 | Verilog乗算（ハードウエア乗算器も使用）の論理遅延時間 |
| multiply const optimize | 16 | 定数乗算を最適化するブロックサイズ |
| and-or logic delay | 1 | 論理演算の論理遅延時間 |
| if-then-else logic delay | 1 | if 式演算の論理遅延時間 |
| noise bits | 0 | ノイズとみなす下位ビット数 |
| independent nan wire | ☑ | 数値化不能フラグ専用の信号線を使用する |
| value of nan | ◉ false ○ nan ○ true | 論理演算結果が数値化不能であった場合の値 |
| maximum checker cases | 256 | チェッカーの最大テストケース数 |

# 表示の選択とファイル出力




ラジオボタンで表示を選択することができます

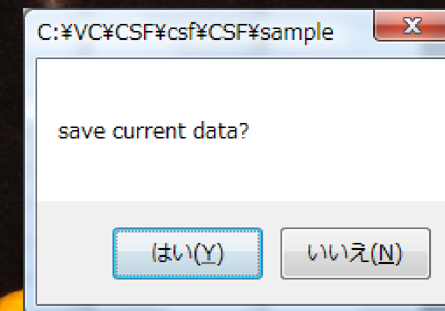
表示内容とファイルセーブの際の拡張子は下表の通りです

ファイルメニューにより表示内容をファイルに出力できます

| ラベル | 拡張子 | 内容 |
|---|---|---|
| mhdl | mhdl | ソーステキスト |
| func | func | コンパイルされるファンクション |
| node | node | コンパイル中間結果* |
| verilog | v | Verilog ソースコード |
| log | log | エラー情報 |
| csv | csv | チェッカーの出力 |

＊:コンパイル・エラーの一部もここに表示されます

終了やファイルオープンの際、問合せウィンドウがポップアップします（下図）「はい」を選択するとこれまでの結果がファイルに格納されます
チェック結果（csv ファイル）はチェック実行時に自動的に作成されます



# 追加情報

*CodeSqueezer* は 11/21 販売開始の予定です

最新情報につきましては弊社 HP*をご参照ください

*) http://signal-process-logic.com

お問い合わせ等は下記宛メールをお送りください

cs@signal-process-logic.com

定価 39,800 円を予定しています。ご期待ください

*シグナル・プロセス・ロジック株式会社*