

CodeSqueezer

取り扱い説明書

Ver. 1.08

シグナル・プロセス・ロジック

もくじ

1	はじめに	1
2	インストール	1
2.1	システム要件	1
2.2	インストールの手順	1
2.3	ライセンス条件の確認	1
3	操作方法	2
3.1	メインウインドウ	2
3.2	機能ボタンと表示の切り替え	2
3.3	プルダウンメニュー	4
3.4	サブウインドウ	5
4	数式記述	7
4.1	代入文	7
4.2	演算子とその優先順位	8
4.3	手続き呼び出し	9
4.4	標準手続き	9
5	mhdl 言語仕様	10
5.1	mhdl の思想	10
5.2	プログラムの要素	10
5.3	手続き定義文	11
5.4	名前の有効範囲	11
5.5	コンパイラ指示子	11
6	数値型決定の仕組み	12
6.1	基本思想	12
6.2	リンクの構造	12
6.3	コーダーの働き	13
7	おわりに	13

1. はじめに

このソフトウェアは、演算式を Verilog HDL に自動変換するもので、特に FPGA で利用するための数値処理論理の開発を念頭に製作されています。

CodeSqueezer には、デモ版、評価版、製品版の 3 種類があります。デモ版と評価版の違いは検証作業の進み具合のみであり、検証が完了した段階でデモ版を評価版に格上げしています。

デモ版、評価版、製品版の機能はいずれも同等です。デモ版および評価版は無許諾・無償で使用することができますが、生成したコードの利用は本製品の評価目的に限定され、これをお客様の商用製品に組み込むことは認められておりません。また、デモ版と評価版には、使用可能期間があります。現在の最新版 Ver.1.06 の使用期限は 2012/4/30 です。

製品版は、プロテクトキーを挿入した PC でのみ使用可能です。製品版を用いて作成した論理は、無制限にお客様の製品に組み込むことが認められます。製品版の価格は 39,800 円です。ご利用をお待ちしております。

2. インストール

2.1. システム要件

このソフトウェアは、Windows XP、Windows Vista、Windows 7 の 32 bit 環境で動作します。メインメモリーの容量などに特段の制約はありませんが、製品版をご購入される前に、使用を予定されている PC にデモ版もしくは評価版をインストールして正常動作を確認されることをお勧めします。

2.2. インストールの手順

CSEval108.msi ファイルをダウンロードします。このファイルは弊社ホームページ (<http://signal-process-logic.com>) から“CodeSqueezer 評価版のページ”を開き、“最新版をダウンロードする”を選択することで自動的にダウンロードされます。

次いで、“CSEval108.msi”をダブルクリックしてインストールを開始します。

インストールの間、インストーラはさまざまな問い合わせを行います。画面の表示に従い「許可する」「次へ」「インストール開始」等を選択することでインストール工程を先に進めることができます。

このソフトウェアはマイクロソフト社の“.NET Framework”を使用しています。これがお使いの PC に含まれていない場合は、インストールの過程で自動的にマイクロソフト社のサイトに接続され、.NET Framework がインストールされます。

2.3. ライセンス条件の確認

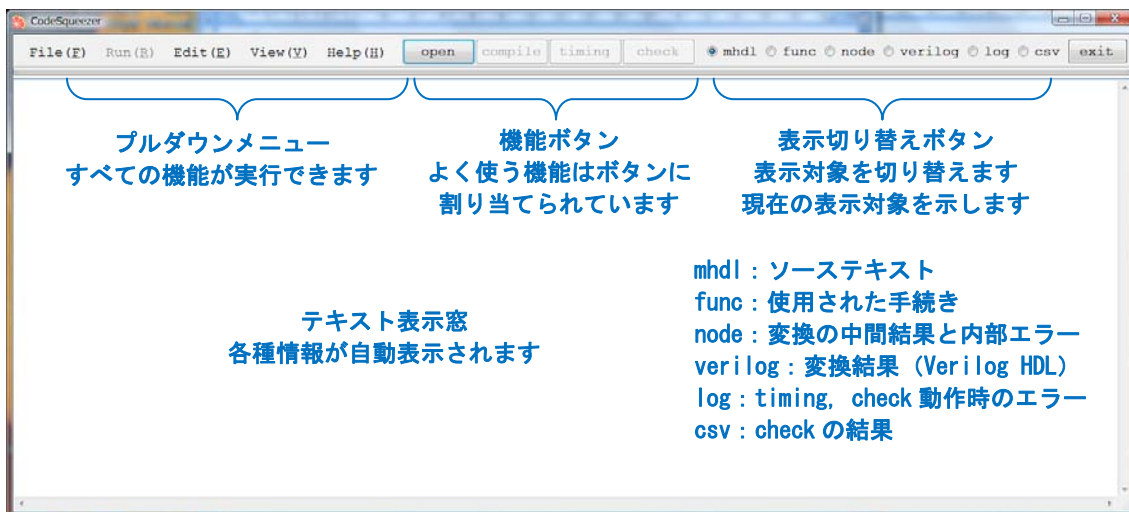
本製品を最初に起動した際には、ライセンス条件の確認画面が表示されます。内容をご確認いただき、“I agree” ボタンを押していただくことにより、本ソフトウェアのすべて

の機能が利用可能となります。この条件にご同意いただけない場合は“I disagree”ボタンを押して画面を閉じてください。

3. 操作方法

3.1. メインウインドウ

CodeSqueezer を起動するとメインウインドウ（下図）が開きます。



3.2. 機能ボタンと表示の切り替え

3.2.1. open ボタン

ソースファイルを開く際に使用します。ソースファイルの拡張子は“.mhdl”です。

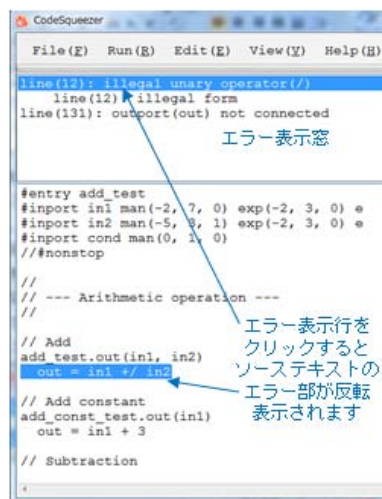
open ボタンを押すと“開く”ダイアログウインドウが開きます。目的のファイルを選択して“開く”ボタンを押すことで、ソーステキストが取り込まれ、テキスト表示窓に表示されます。

ソーステキストには、数式または mhdl 言語で記述された手続きを与えます。数式の記述に関しては 4 節、mhdl 言語仕様に関しては 5 節をご参照ください。

3.2.2. compile ボタン

このボタンを押すとコンパイルが開始されます。ソーステキストでコンパイル条件を指定していない場合は、コンパイルに先立って、必要な各種情報を設定するための“Entry Editor”画面が開きます。コンパイル条件の指定方法については 5.5 節で、Entry Editor の詳細につきましては 3.4.1 節で解説します。

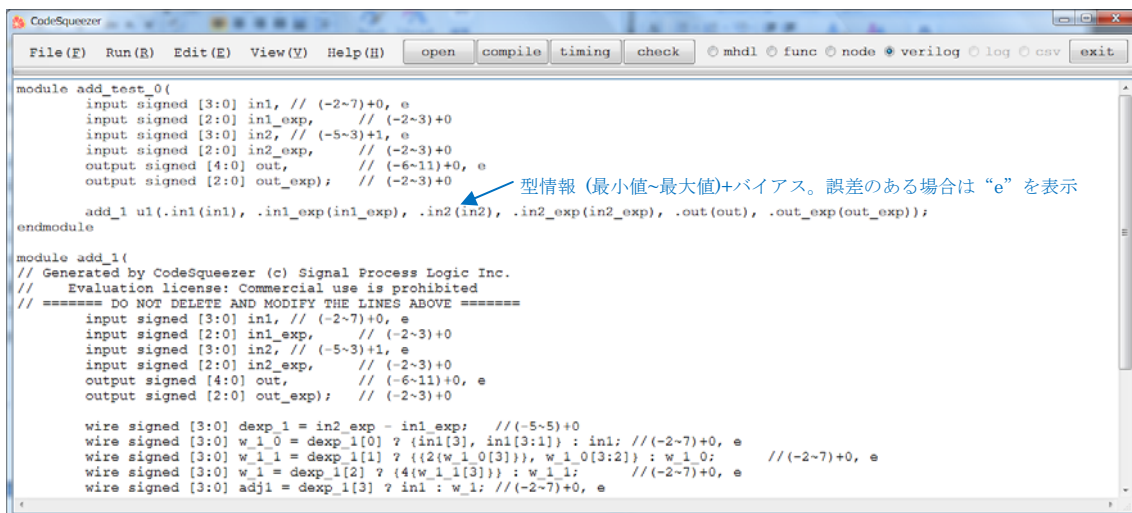
コンパイルエラーがある場合は、テキスト表示窓の上部にエラー表示窓が現れます。エラー表示行をダブルクリックするとソーステキストの該当部分が強調表示さ



れます。

テキスト表示窓では各種編集操作が可能です。エラー箇所を修正して compile ボタンを押すことでコンパイルし直すことができます。また、ソーステキストが表示されている状態で Ctrl-S を押すと、表示されている内容が元のファイルにセーブされます。

コンパイルエラーがない場合は、テキスト表示窓に Verilog HDL に変換された結果が表示されます。この段階では、クロック信号やレジスタを含まない“非同期コード”に変換されています。



コンパイル結果には、コメントとして型情報が追加されます。

3.2.3. timing ボタン

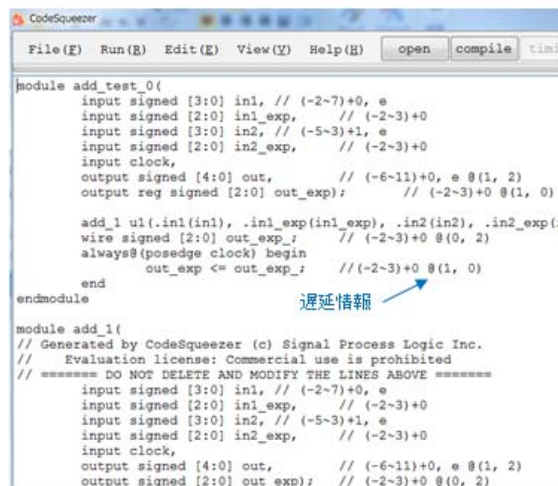
timing ボタンを押すと、必要な個所に自動的にレジスタが挿入され、クロック同期コードを形成します。また、その結果を反映した Verilog HDL のコードをテキスト表示窓に表示します。

コメントの末尾に遅延情報が付加されます。遅延情報は@マークに続けて、クロックサイクルの遅れ、およびクロック立ち上がりから信号が確定するまでの時間をコンマで区切ってかっこ内に表示します。

timing の処理は、それぞれの演算における論理遅延時間と、許容される最大の論理遅延時間（通常はクロック周期）に基づいて行われます。これらの条件は、プルダウンメニュー“Edit”の“Environ”で設定します。設定方法の詳細は、3.4.3 項をご参照ください。

3.2.4. check ボタン

check ボタンを押すと、論理検証を行い、



結果を表示するとともに csv ファイルに書き出します。テストベクターは入力信号範囲に基づいて自動生成されます。テストベクターの生成方法は Entry Editor で制御することができます (3.4.1 節)。テストベクターが設定された最大値より多い場合は、最大数程度に収まるようランダムに間引きされます。テストベクターの最大数は、プルダウンメニュー “Edit” の “Environ” で設定します。

3.2.5. 表示切り替えボタン

テキスト表示窓に表示される内容は、メインウインドウ右上部のラジオボタンによって切り替えることができます。表示内容はコマンドが実行されたときに自動的に切り替わりますが、このときラジオボタンの選択マークも自動的に変化し、テキスト表示窓に何が表示されているかを通知します。

ラジオボタンのラベルと表示内容との関係は以下の通りです。

- mhdl : ソースコード
- func : 使用された手続き
- node : 変換の中間結果と内部エラー
- verilog : 変換結果 (Verilog HDL)
- log : timing, check 動作時のエラー
- csv : check の結果

3.3. プルダウンメニュー

本ツールのすべての機能は、メインウインドウ左上のプルダウンメニューから実行することができます。メニューの割り当ては、標準的なプルダウンメニューに準じています。

3.3.1. File プルダウンメニュー

New: 新しいソーステキストを作成します。テキスト表示窓に直接書き込みます。

Open: 既存のソースファイルを開きます。機能ボタン “open” と同じです。

Save: 現在のソースコードを読み込んだファイルにセーブします。

Save As: 現在のソースコードを新しいファイルにセーブします。

Exit: プログラムを終了します。“exit” ボタンと同じです。

3.3.2. Run プルダウンメニュー

Compile: コンパイルします。機能ボタン “compile” と同じです。

Timing: タイミング処理をします。機能ボタン “timing” と同じです。

Check: チェッカーを起動します。機能ボタン “check” と同じです。

3.3.3. Edit プルダウンメニュー

Cut: 選択範囲を切り取りコピーバッファに取り込みます。Ctrl-X と同じです。

Copy: 選択範囲を複製してコピーバッファに取り込みます。Ctrl-C と同じです。

Paste: コピーバッファをカーソル位置にコピーします。Ctrl-Vと同じです。

Select All: テキスト全体を選択範囲とします。Ctrl-Aと同じです。

Environ: 変換条件を設定するための Environ Editor がポップアップします。この操作方法は 3.4.2 節で解説します。

3.3.4. View プルダウンメニュー

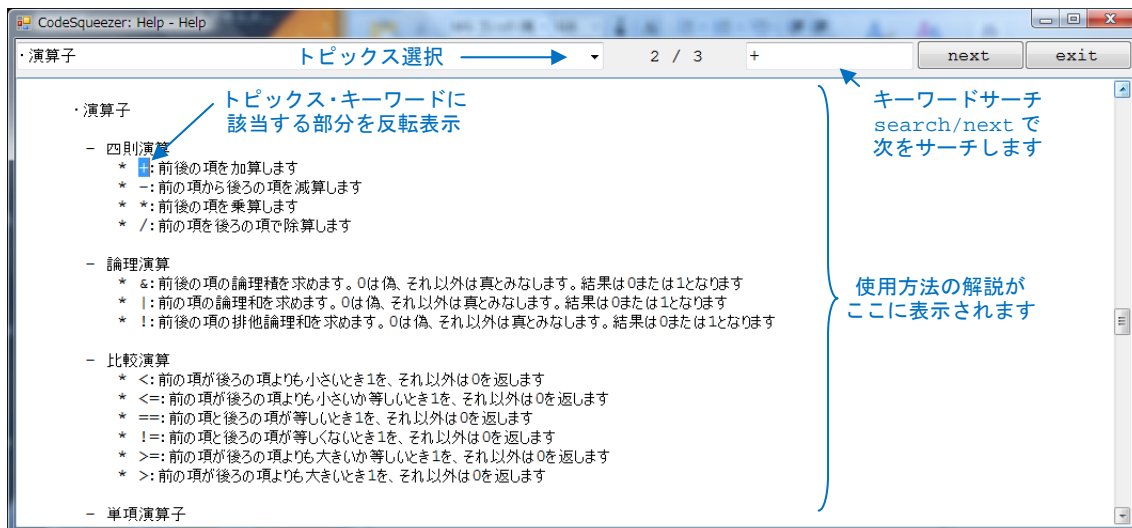
このメニュー項目は、表示切り替えボタンと同じ機能を持ちます。

3.3.5. Help プルダウンメニュー

License: ライセンス条件を表示します。

Help: 使用方法を解説するヘルプウインドウ (下図) が開きます。

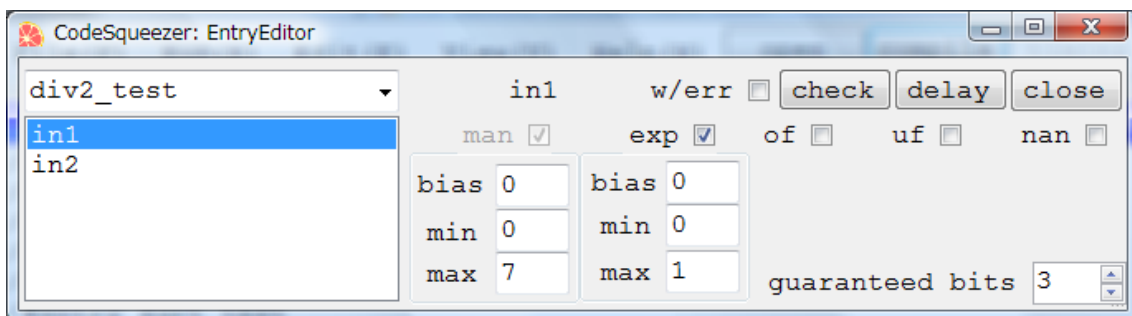
Version: バージョン情報を表示します。



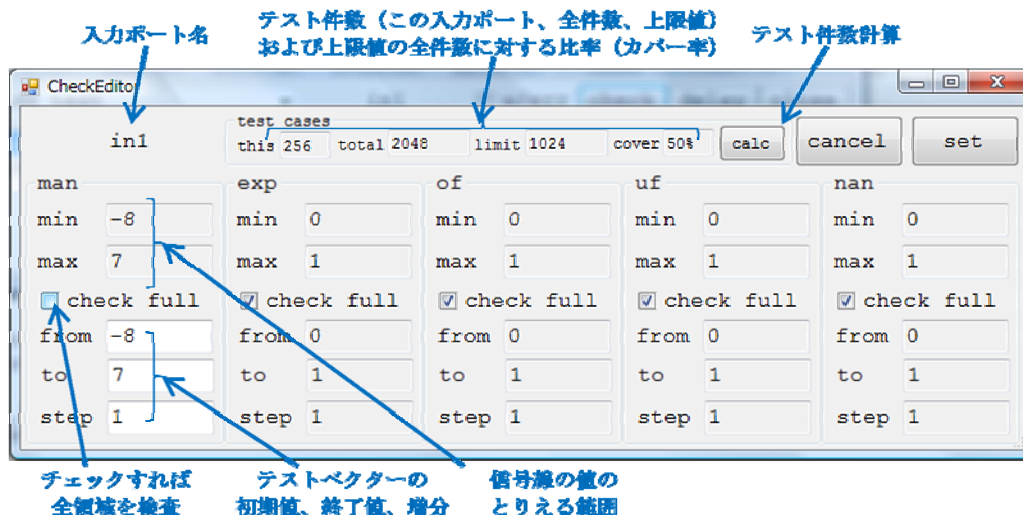
3.4. サブウインドウ

3.4.1. Entry Editor

コンパイラ指示子 “#nonstop” がない場合、compile ボタンを押すと EntryEditor 画面が開きます。この画面では、コンパイルを行う最上位の手続きと、外部入力信号の属性を設定します。これらの条件はコンパイラ指示子によりソース中に記述することもできます。



“check” ボタンは論理検証におけるテストベクター生成機能を制御するためのもので、このボタンを押すことで開くウィンドウ（下図）により選択された入力に対するテストベクターの初期値、最終値、増分を設定します。



calc ボタンを押すと、この入力ポートに対してテストを行う状態の数と、入力ポート全体の状態数から計算されるテストベクターの数、およびテストベクターの上限とその計算値に対する比率（カバー率）が再計算されて表示されます。

カバー率が 100%に満たない場合は、テストベクター数の上限により、指定された入力ポートの状態の全ての組合せに対するテストは行われず、適宜間引きされたケースに対してテストが行われます。

3.4.2. Environ Editor

Environ Editor は、コード形成に際して使用される種々の条件を設定するためのウィンドウで、プルダウンメニューの Edit から Environ を選択することにより開きます。

このウィンドウは、機能ボタンと同名のタブによりページを切り替えて、各機能ボタンに対応する処理条件を設定します。操作方法は次の通りです。

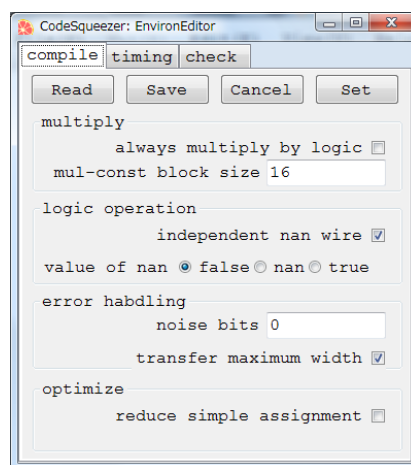
Read ボタン：ファイルから変換条件を読み取ります。

Save ボタン：現在の変換条件（表示内容）をファイルに書き込みます。

Cancel ボタン：現在のウィンドウに加えられた変更を取り消して Environ Editor を終了します。

Set ボタン：表示された各条件をセットして Environ Editor を終了します。

limit logic delay：最大許容遅れ時間を任意単位（他の遅れ時間と同じ単位としま



す) で設定します。

`add-sub logic delay` : 加減算の遅れ時間。

`always multiply by logic` : 乗算論理のための論理回路を形成します。この指定がない場合は、Verilog HDL の乗算演算子を使用して、乗算論理の形成を Verilog HDL の処理系に委ねます (多くの処理系ではハードウェア乗算器を割り当てます)。

`multiply logic delay` : Verilog HDL の乗算演算子を用いた演算で生じる遅れ時間。

`multiply const optimize` : 定数乗算最適化のブロックサイズです。定数乗算は経路探索の手法を用いてシフト演算と加減算の組合せに変換しますが、探索を行うブロックサイズが大きいくほど、多桁の定数を乗算する際の演算時間が増大します。定数乗算を含むソースコードを処理する際に変換時間が長すぎる場合は、この値を減少してみてください。

`and-or logic delay` : 論理演算の遅れ時間。

`if-then-else logic delay` : 条件式演算の遅れ時間。

`noise bits` : 誤差のある信号線で誤差 (ノイズ) とみなされる仮数の下位ビット数。

`independent nan wire` : 数値化不能異常フラグ (nan) に独立信号線を用いることを指定します。nan 信号線を用いない場合は、オーバーフローフラグとアンダーフローフラグを共に立てることで nan を表します。

`value of nan` : 論理演算結果が数値化不能である場合 (nan が入力された場合に生じる) の出力を真とするか、偽とするか、nan とするかを設定します。

`maximum checker cases` : チェッカーで発生させるテストベクターの最大数。

`reduce simple assignment` : Verilog コード形成の際に単純な代入文を省略します

`transfer maximum width` : 信号線幅が許す限りのビット幅で後段に伝達します

4. 数式記述

4.1. 代入文

代入文は“**信号線名 = 式**”の形で記述します。

信号線名は、英字 (大文字と小文字を識別します) に始まる英数字の並びです。アンダースコア “_” は英字とみなされ、信号線名にも使用することができますが、特に先頭もしくは末尾を “_” とする名前はシステムが割り当てている場合がありますので、使用しない方が安全です (支障の有無は Verilog HDL のソースコードで確認することも可能です)。

式は“**項**”もしくは“**演算子で結ばれた項の並び**”で、項は以下のいずれかです。

- ・ 信号線名
- ・ 定数
- ・ 項の前に単項演算子が置かれたもの
- ・ より優先順位の高い演算子で結ばれた項の並び
- ・ かつこ “()” に囲まれた式
- ・ 手続き呼び出し

代入文右辺の式中に現れる信号線は、その代入文よりも上で代入文左辺の信号線もしくは入力信号線として現れていなければなりません。この制約により信号はソースリストの上から下へ流れることが強制され、プログラムの可読性を改善します。

定数は十進数で表現します。定数には小数点に続く小数部と指数部を含めることが可能です。指数部は、指数の底を表す記号“e”、“b”、“d”のいずれかに、十進数を続けて構成されます。指数部の十進数には符号(“-”または“+”)を付けることができます。底の記号“e”および“d”は10の冪であることを、“b”は2の冪であることを表します。

定数を構成する文字列が“.”または“e”を含む場合には誤差を含むものとして扱われます。数値の末尾に“e”を付け加えて誤差を含むことを指定することもできます。

単項演算子は、符号を反転させる負号“-”または論理を反転させる否定“!”のいずれかです。複数の単項演算子がある場合は右側から演算されます。

代入文は“a = b = c”のように二つ以上の代入演算子を記述することも可能です。このように書かれた場合は“b = c”に続いて“a = b”が記述されたと解釈されます。

4.2. 演算子とその優先順位

演算子とその優先順位は下表のとおりです。

優先順位	演算子	演算内容	項数
高い ↑ ↓ 低い	^	冪乗	任意
	*, /	乗算、除算	任意
	+, -	加算、減算	任意
	<, <=, == !=, >=, >	比較	2項限定
	&	論理積	任意
	!	排他論理和	任意
		論理和	任意
	?, :	条件式	3項限定

比較演算および論理演算は論理値を返します。論理値は真の場合に数値1、偽の場合に数値0をとります。数値に対して論理演算が行われる場合などで数値を論理値に変換する際には、0である場合を偽、0以外である場合に真であるとみなされます。

noise bits (3.4.3 節参照) がゼロでない場合、誤差のある信号線値の値が noise bits で表現される範囲内である場合には、この信号線の論理値は偽とみなされます。

排他論理和には否定演算子と同じ“!”を使用します。これは、排他論理和は前後の論理値が等しくない場合に真を返すためであり、数値が等しくないことを表す演算子“!=”に準じています。排他論理和は任意の項数に対してとることが可能であり、こうした場合には、真である項の数が奇数である場合に真を、偶数である場合に偽を返します。

条件式は“a ? b : c”の形で記述します。aは論理値として扱われ、式の値はaが真である場合にb、偽である場合にcをとります。

4.3. 手続き呼び出し

手続き呼び出しは“手続き名(引数リスト)”の形で記述されます。

手続きには、“ユーザ定義手続き”と“標準手続き”があります。標準手続きは、ツールとともに提供されているもので、次節に記述いたします。ユーザ定義手続きはユーザが定義する手続きで、定義方法は5.3節で解説いたします。

引数リストはコンマ“,”で区切られた項の並びとして記述します。

式中に記述できる手続きは単一の信号線を返す手続きに限られます。複数の信号を返す手続きは代入文右辺にのみ記述することが可能で、その形式は次の通りです。手続きの出力は左辺に記述された各信号線に順次返されます。

(信号線名, 信号線名, ...) = 手続き名(引数リスト)

4.4. 標準手続き

4.4.1. 演算手続き

システムは演算子を手続き呼び出しに変換しています。これらの手続きは標準手続きとして提供されており、どのような手続きに変換されたかは表示を `func` に切り替えることで知ることができます。これらの手続きは、ソースコード中に記述して呼び出すことも可能です。

除算において、商と余りが必要な場合には標準手続き“`div2`”を使用します。`div2`の呼び出し形式は以下の通りです。

(商, 余り) = `div2`(被除数, 除数)

`div2` は、被除数と除数が共に誤差を含まない場合、相互の仮数部に対して整数除算を行い、商と余りを出力します。それぞれの指数部は“`商 × 除数 + 余り = 被除数`”となるように設定されます。被除数と除数のいずれかが誤差を含む場合には、有効桁に対する除算を行ってその結果を商として出力するとともに、余りはゼロを返します。

`sqrt` は、引数の平方根を返します。

余りを与えない除算と平方根演算の結果はいずれも、入力に誤差が含まれない場合にも、誤差をもちます。このとき、誤差なし変数は `guaranteed_bits` の有効桁数をもつとみなされ、その最大値を結果の有効桁数とします。`guaranteed_bits` は、デフォルトでは符号を除くビット幅に設定されますが、`EntryEditor` で任意の値に設定することもできます。

4.4.2. 有効桁の削減

`CodeSqueezer` は、誤差を含む信号線に対して有効桁のみを出力するコードを形成しますが、有効な情報が失われないよう、少しでも有効な情報を含む桁はすべて出力します。この結果、演算を繰り返すに従って下位の桁に含まれる有効な情報は次第に減少します。

ユーザの判断による無効桁の除去を可能とするため、標準手続き `reduce` が準備されています。この関数を `reduce(in)` の形で呼び出した場合には `in` の有効桁数を一つ減じた出

力が得られます。reduce の具体的な処理は、仮数部の最下位桁を削除するとともに指数部を 1 増加させることで行っています。

4.4.3. 出力の加工

内部信号線と出力信号線の型は、入力信号線の型に従って自動的に決定されます。しかしながら、出力の型は出力に接続されるデバイスに適合する型とする必要があります。このような際には型を変換する標準手続きを使用します。

型変換の標準手続きとして準備されているものは以下の通りです。

`to_int(in)` : `in` を整数に変換します。

`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。

`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。

整数のビット幅を制限する場合には `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

4.4.4. 異常と符号の検出

異常および符号を検出するための標準手続きとして以下のものがあります。

`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

`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。

5. mhdl 言語仕様

5.1. mhdl の思想

mhdl は演算処理の仕様を抽象的に記述するための言語であり、最終的にハードウェア記述言語 (HDL) に変換されることを意識して仕様が定められています。

mhdl は、抽象的な演算機能を有する“手続き”の入出力ポート間を抽象的な信号線である“リンク”で接続した形で演算処理仕様を規定します。mhdl には数式を記述することも可能です。コンパイルの段階で数式は手続き呼び出しに変換されて処理されます。

抽象記述を徹底するため、ハードウェアに依存した機能 (演算ハードウェアが二進法に基づいて構成されていることも含めて) は mhdl の仕様からは排除されています。

5.2. プログラムの要素

mhdl のソースコードは改行記号で区切られた文字列として与えられます。

“/”以降行末まで、および“/*”と“*/”の間はコメントとみなされ、空白と同等に扱われます。改行記号の前の空白は無視されます。

改行記号は文の区切りとして扱われますが、閉じかっこ以外の記号で終わる行は次行に継続するものとみなされ、改行記号と次行先頭の空白は単一の空白に変換されます。記号は、演算子を構成する演算記号“*/+-<>!=&|?:()”および区切り記号であるコンマ“,”と空白のいずれかです。これらのうち、空白もしくは閉じかっこ“)”以外の記号で終わる行が継続行となります。

行頭の空白の数をその行のレベルと呼びます。レベルの高低に関しては、空白が多いほど低く、空白が少ないほど高いものとみなします（レベルは字下げの“深さ”に相当します）。

文には代入文と手続き定義文があります。代入文に関しては4.1節をご参照ください。

5.3. 手続き定義文

手続き定義文の一般的形式は以下の通りです。

手続き名 . 出力記述 (入力信号線のリスト)

手続き名は信号線名と同様、英字に始まる任意長さの英数字の並びとして記述されます。出力記述は、出力信号線が単一である場合には信号線名を、複数の信号線を出力する場合はこれらの信号線の名称をかっこ“()”の中にコンマ“, ”で区切って並べます。

入力信号線のリストは括弧の中に入力信号線名をコンマで区切って並べたもので、出力記述と異なり、信号線が単一の場合もかっこは省略できません。

手続き定義は、代入文と内部手続き定義が含まれます。手続き定義の範囲は、手続き定義文よりもレベルの低い文が続く限りです。

手続き定義に含まれない代入文が記述された場合、見えない手続き“_root”がすべてを含むものとみなされます。代入文の右辺にのみ現れる信号線は入力信号線、左辺にのみ現れる信号線は出力信号線となります。

5.4. 名前の有効範囲

手続きの内部で宣言された手続き名は、宣言を行った手続きの内部でのみ通用します。他の手続きに含まれない手続きのみがコンパイルの際の最上位手続きとなり得ます。

信号線名の有効範囲は、その信号線が宣言された手続きの内部に限定されます。

5.5. 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 手続き名 : コンパイルを行う最上位の手続きを指定します

#inport 型指定子 : 入力 of の型を指定します。型指定子” of uf nan e

man(最小値, 最大値, 定数部) exp(最小値, 最大値, 定数部) of uf nan e

ここで、man は必ず指定し、exp, of, uf, nan はこれらの信号線が存在する場合に指定します。最後の“e”は、信号線が誤差を含む場合に指定します。

#nonstop: これが指定された場合、必要がなければ Entry Editor をスキップします。

6. 数値型決定の仕組み

6.1. 基本思想

mhdl ソースコードの段階では数値は型をもたず、数値型はコンパイルの段階で決定されます。型決定のため、入力信号仕様だけはユーザが与える必要があります。システムはこの情報に基づき、信号の伝達経路に従って順次、手続きの構成を決定し、信号線の型を決めてゆきます。

生成論理が消費する論理資源を削減するため、無効な情報に対する演算論理は生成しません。各演算器出力の有効桁を決定するため、それぞれの信号線は誤差を含む情報であるのか、誤差を含まない情報であるのかを識別するフラグが与えられています。入力信号が誤差を含む場合には、有効桁のみを出力する論理が自動的に構成されます。

6.2. リンクの構造

抽象的信号線を“リンク”と呼びます。リンクは複数の“ワイヤ”で構成されます。ワイヤは Verilog HDL の“wire”に対応する概念で、複数のビットから構成され整数値を表現することができます。

リンク名を name とするとき、これに含まれるワイヤの名称と意味は以下の通りです。

name : 仮数部です。すべてのリンクはこのワイヤをもちます。

name_exp : 指数部です。これがある場合、仮数部の値に $2^{\text{指数部}}$ が乗じられます。

name_of : オーバーフローフラグで、0 または 1 の値のみをとります。

name_uf : アンダーフローフラグで、0 または 1 の値のみをとります。

name_nan : 数値化不能異常フラグで、0 または 1 の値のみをとります。

ワイヤは属性として“最大値”、“最小値”および“バイアス”をもちます。ワイヤを Verilog HDL の“wire”文に展開する際、最大値から最小値までを格納可能なビット幅が取られます。また、最小値が負である場合、このワイヤは符号付であるとして扱われます。ワイヤが定数である場合は、最大値と最小値を共にゼロとし、定数値をバイアスに与えます。

ワイヤの値は Verilog HDL における wire の値にバイアスを加算したものとみなされます。バイアス部分は Verilog HDL のコードには現れず、mhdl 処理系の内部でのみ意味をもちます。

このような仕様を採用している理由は、生成されるコードを最小にするため、および各種の数値を統一的に取り扱うことを可能とするためです。すなわち、定数を加算する演算はバイアスのみの操作で実現されるため、Verilog HDL のコードは不要となります。また、定数、変数、固定小数点数、浮動小数点数を同様に扱うことが可能となり、処理系が

簡素化されるという利点があります。

6.3. コーダーの働き

各手続きは、手続き固有の“コーダー”と呼ばれる関数により実現されています。システムは、コンパイル指令を受け取ると、コンパイル対象であるすべての手続きに対して順次コーダーを起動するよう要求を出します。

コーダーは、手続きのすべての入力信号線の属性定義が完了していることをチェックして、これらの属性定義に従って手続きの機能を中間形式（ノード）に変換し、出力信号線の属性定義をセットして、手続きの変換完了フラグを立てます。

いずれかの手続きで変換がおこなわれている間、システムは全ての手続きに対して繰り返しコーダー起動要求を出します。これにより、全ての手続きが順次ノードに変換されます。

ノードは Verilog HDL の代入文と一対一に対応するもので、ノードを Verilog HDL のソースコードに変換する際には、その左辺が入出力ポートに割り当てられているかどうか、および左辺はレジスタであるかワイヤであるかに応じて適切なソースコードに変換されます。これらの処理の詳細に関しては文献*をご参照ください。

*) “浮動小数点処理を含む論理設計支援システム” 情報処理学会 DA シンポジウム 2010 論文集 p3~8

7. おわりに

以上、CodeSqueezer の使用方法と内部処理の概要について説明いたしました。本文書の内容についてご疑問、ご不明の点がございましたら、遠慮なく弊社にお問い合わせください。デモ版、評価版をご利用のお客様に対しても可能な限りのサポートをいたします。弊社カスタマーサポートのメールアドレスは以下の通りです。

`cs@signal-process-logic.com`

文書、ソフト等につきましては随時バージョンアップを行ってまいります。最新情報は弊社のホームページをご参照ください。弊社ホームページの URL は下記の通りです。

`http://signal-process-logic.com`

今後ともお引き立てのほどよろしく願いいたします。

改変記録

2011.11.12 新規作成

2011.11.13 表現全般の手直し

2012. 3. 7 Ver. 1.06 に対応

2012. 5.30 Ver. 1.08 に対応

2012. 8. 6 Ver. 1.08Biz/Eval に対応

シグナル・プロセス・ロジック株式会社

<http://signal-process-logic.com>