CodeSqueezer

取り扱い説明書

シグナル・プロセス・ロジック

もくじ

1. はじめに

このソフトウエアは、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 製品版は同梱のプロテクトキーを挿入した PC でのみ使用可能です。本ソフトウエアで作成した論理は、無制限にお客様の製品に組み込むことができます。

2. インストール

2.1. システム要件

このソフトウエアは、Windows XP、Windows Vista、Windows 7 の 32 bit 環境で動作します。ディスプレーには、本ソフトウエアのメインウインドウ（幅 1280×高さ 600）が表示できる SVGA 以上の解像度が必要です。その他に関しては、特段の制約はありません。

2.2. インストールの手順

“CSBiz*xxx*.msi”はウインドウズインストーラに対応したインストールパッケージです。ここで、“*xxx*”はバージョンを表す数値です。また、最新のインストールパッケージは、弊社ホームページ（http://signal-process-logic.com）から“CodeSqueezer 製品**版のページ**”を開き、“**最新版をダウンロードする**”を選択していただくことで入手可能です。同梱のプロテクトキーは、将来のバージョンにおいても有効に機能します。

次いで、“CSBiz*xxx*.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が、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。

このソフトウエ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”ボタンを押してください。

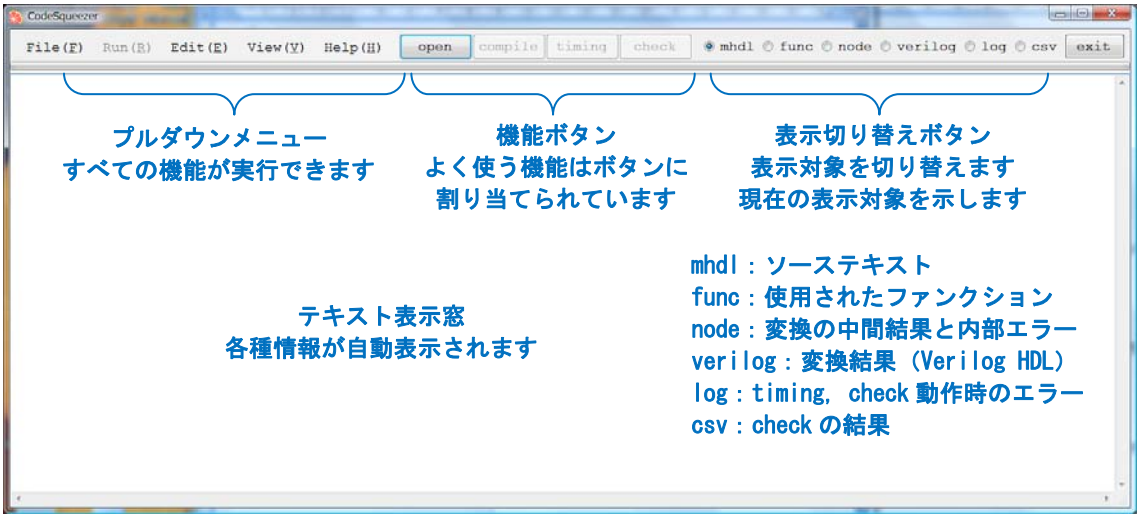
本製品が生成するコードには弊社の著作物が含まれておりますが、上記ライセンス条件をご確認いただくことで、お客様（プロテクトキーの正当な所有者）にはこれらを含む製品を無制限に製作・販売することが許諾されます。

なお、生成コードがこの許諾の元に作成されていることは、モジュールのコメント部に付加される許諾コードで識別されるようにしております。モジュールのコメント部は削除しないようお願いします。

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。各部の操作方法につき、以下解説します。

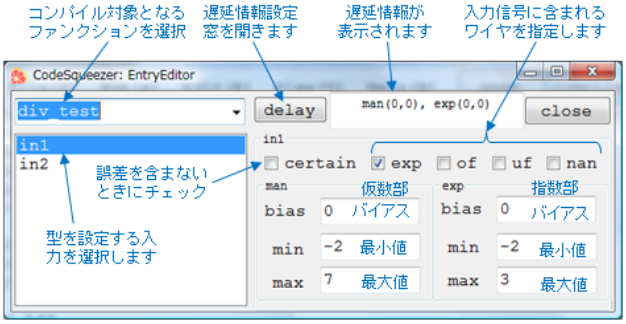


3.2. 機能ボタン

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。

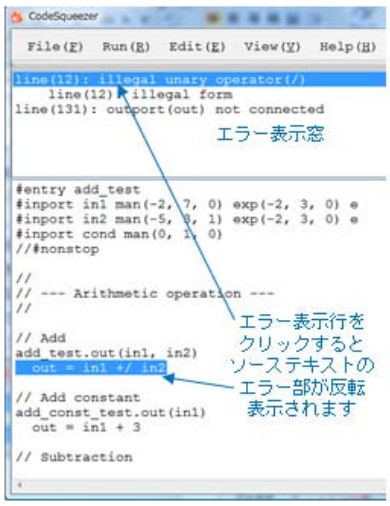
open ボタンを押すと“開く”ダイアログウインドウが開きます。目的のファイルを選択して“開く”ボタンを押すことで、ソーステキストが取り込まれ、テキスト表示窓に表示されます。



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。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。この画面での設定は右上の図の通りです。なお、これらの情報はコンパイラ指示子を用いてソースファイル中に記述することもできます。

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

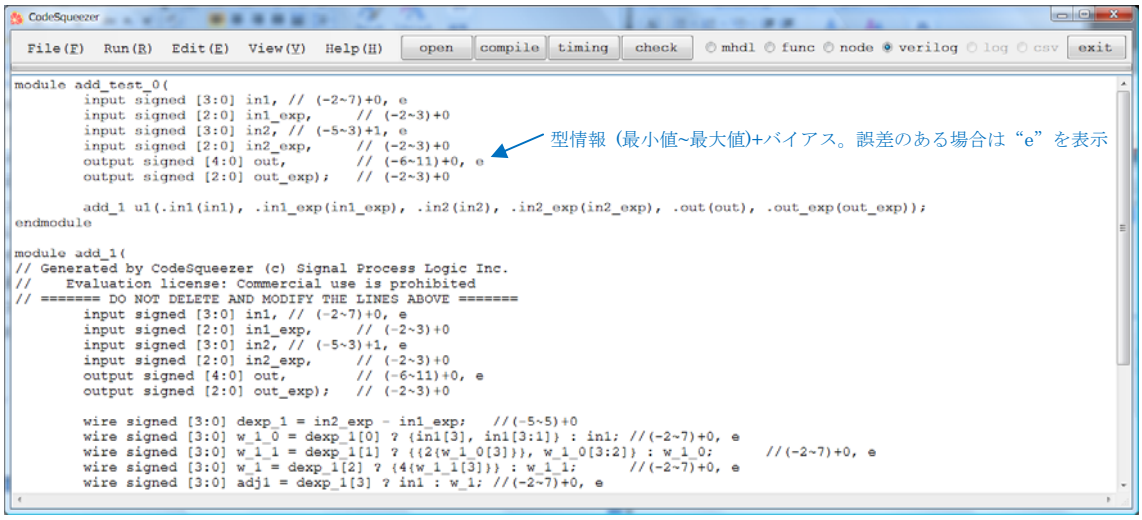


リックするとソーステキストの該当部分が強調表示されます。

テキスト表示窓では各種編集操作が可能です。エラー個所を修正して 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。

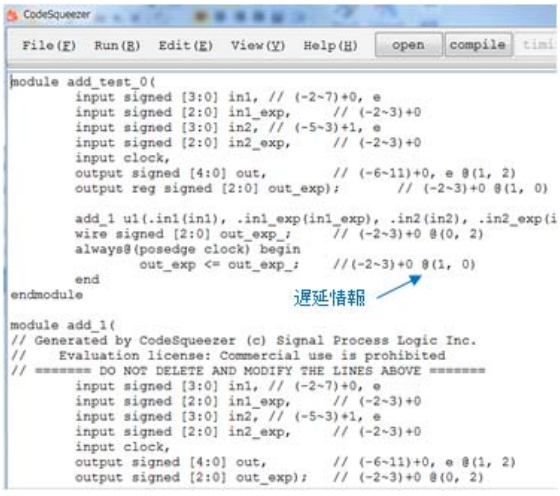


3.2.3. timing ボタン

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も、プルダウンメニュー“Edit”の“Environ”で設定します。

csv ファイルはカレントディレクトリに書き込まれます。インストールディレクトリがカレントディレクトリとなっている場合、書き込み禁止エラーとなる場合があります。このような場合は、ソースファイルをユーザディレクトリにコピーし、これを open することでカレントディレクトリをユーザディレクトリに変更してください。

3.3. 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：使用されたファンクション
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

3.4. 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させることができます。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。

3.4.1. File プルダウンメニュー

New: 新しいソースコードを作成します。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。

3.4.2. 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。

3.4.3. Edit プルダウンメニュー

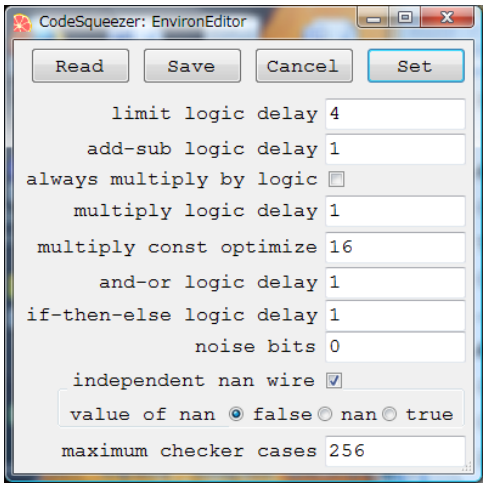
Cut: 選択範囲を切り取りコピーバッファに取り込みます。Ctrl-X と同じです。

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。この操作方法は次節で解説します。



3.4.4. Environ Editor

Environ Editor の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：現在のウイ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：ウインドウ表示に従って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エア乗算器を割り当てます）。

multiply logic delay：Verilog HDL の乗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。

3.4.5. View プルダウンメニュー

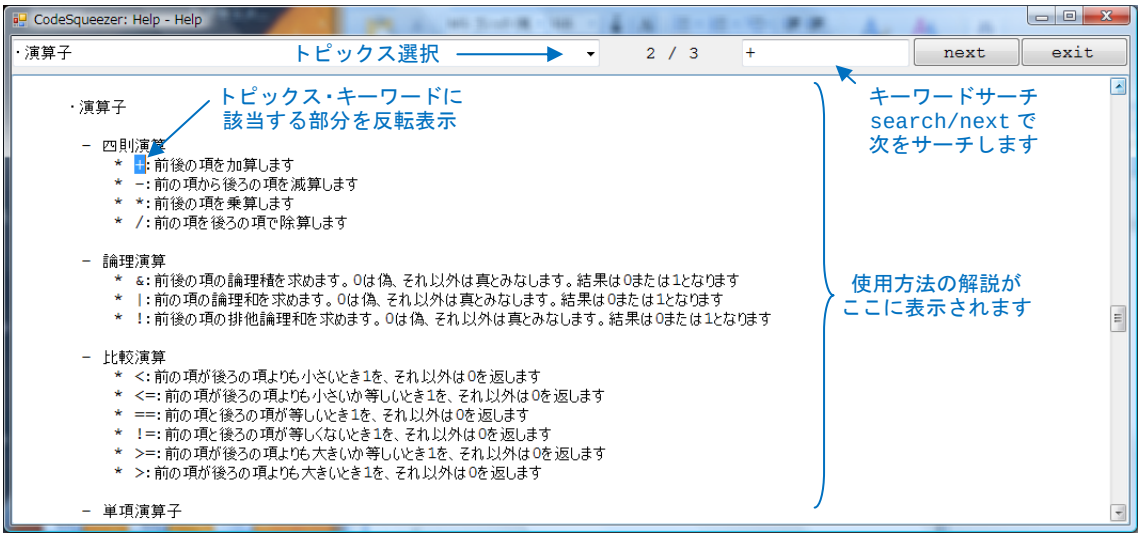
このメニュー項目は、表示切り替えボタン（3.3 節参照）と同じ機能をもちます。

3.4.6. 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。



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
- かっこ“()”に囲まれた式

・ ファンクション呼び出し

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。この制約は、信号がプログラムの記述に従って（ソースリストの上から下へ）流れることを強制するもので、プログラムの読み易さを確保するとともに、誤りの混入を排除するために設けられています。

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。底の記号に“e”を用いた場合、この定数は誤差を含むことを意味します。定数が誤差を含むことは、定数の末尾に“e”を付け加えることでも指定できます。

小数部を含む十進数で表される定数は、正確な二進数に変換できないかもしれないことにご注意ください。小数部を含む十進定数は、誤差を含む定数であることを明示して記述することを推奨します。

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、定義により、右側から演算されます。

演算子とその優先順位に関しては次節を、ファンクション呼び出しに関しては次々節をご参照ください。

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 ↑	*, /	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
↓ 低い	?, :	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真偽不明となります。このような場合の論理演算は数値化不能（nan）異常値が入力されたものとして扱います。数値化不能である論理演算結果は、nan value（3.4.3節参照）に従います。

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. ファンクション呼び出し

ファンクション呼び出しは“**ファンクション名(引数リスト)**”の形で記述されます。

使用可能なファンクションには、“**ユーザ定義ファンクション**”と“**標準ファンクション**”があります。標準ファンクションは、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。ユーザ定義ファンクションはユーザが定義するファンクションで、定義方法は 5.3 節で解説いたします。なお、再帰呼び出しを行うことはできません。

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。

ファンクションの帰り値は単一であるとは限りませんが、式中に記述できるファンクションは単一の信号線を返すファンクションに限られます。

複数の信号を返すファンクションは演算式を含まない代入文にのみ記述することが可能で、その形式は次の通りです。ファンクション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, …) = ファンクション名(引数リスト)

4.4. 標準ファンクション

4.4.1. 演算ファンクション

システムは演算子をファンクション呼び出しに変換しています。これらのファンクションは標準ファンクションとして提供されており、どのようなファンクション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。

これらのファンクションを式中に記述して呼び出すこともできますが、可読性の観点から、ファンクション呼び出しとはせずに、演算子を用いて記述することをお勧めします。

除算において商と余りが必要な場合には標準ファンクション“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。

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、標準ファンクション 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ファンクションを使用します。

型変換の標準ファンクション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ファンクション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、最終的にハードウエア記述言語（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“**ファンクション**”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ファンクション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、ハードウエアに依存した機能（演算ハードウエ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、ブランクと同等に扱われます。改行記号の前のブランクは無視されます。

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ブランクは単一のブランクに変換されます。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“,”とブランクのいずれかです。

行頭のブランク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、ブランクが多いほど低く、ブランク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)。

文には代入文とファンクション定義文があります。代入文に関しては 4.1 節をご参照ください。

5.3. ファンクション定義文

ファンクション定義文の一般的形式は以下のいずれかです。

ファンクション名.出力記述(入力信号線のリスト)

ファンクション名（入力信号線のリスト)

ファンクション名は、信号線名と同様、英字に始まる英数字の並びです。

“出力記述”の部分には、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。出力信号線が単一である場合には信号線名の記述も省略できます。この場合は、ファンクション名と同名の出力信号線を宣言したものとみなされます。

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。

ファンクション定義は、代入文と内部ファンクション定義が含まれます。ファンクション定義の範囲は、ファンクション定義文よりもレベルの低い文が続く限りです。

ファンクション定義に含まれない代入文が記述された場合、見えないファンクション“_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

ファンクションの内部で宣言されたファンクション名は、宣言を行ったファンクションの内部（下位のファンクションを含む）で通用します。これに対して信号線名は、その信号線を宣言した（直上の）ファンクション内部でのみ有効です。

コンパイルは、他のファンクションに含まれないファンクションのみから開始可能です。

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 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”の一般形式は以下の通りです。

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、ファンクションの構成を決定し、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取り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

各ファンクションは、ファンクション固有の“**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ファンクション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、ファンクション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ファンクションの機能を中間形式（ノード）に変換し、出力信号線の属性定義をセットして、ファンクションの変換完了フラグを立てます。

いずれかのファンクションで変換がおこなわれている間、システムは全てのファンクション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、全てのファンクション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

本文書の内容ついてご疑問、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。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

ソフトウエア本体および各種文書は随時バージョンアップをいたします。これらの最新情報は、弊社のホームページ（http://signal-process-logic.com）に掲載いたしますので、適宜ご参照ください。

改変記録

2011.11.12 新規作成
2011.11.13 表現全般の手直し
2011.11.19 評価版に対応。出力宣言の省略について記述を追加
2011.11.20 製品版に対応
2011.11.22 ライセンス条件の説明を手直し

シグナル・プロセス・ロジック株式会社
http://signal-process-logic.com