パイプライン処理のための演算仕様記述言語 mhdl とその処理系

シグナル・プロセス・ロジック株式会社

瀬尾雄三

要旨

数値演算のパイプライン処理では、信号線や演算器の仕様を個別に設定することができる．我々は有効桁に着目してこれらの仕様を最適化する手法の開発を進めているが、今回はこれに用いる演算仕様記述言語 mhdl について報告する．小規模な利用者が多い FPGA 向けの論理開発に配慮して、mhdl の言語規約は必要な機能に限定して容易な習得を可能とした．また、今日のソフト開発環境に鑑みて改行と字下げにも意味を与えてプログラム構造の把握を容易にしたほか、冗長な記述を可能な限り排除して可読性向上を図った．mhdl で記述されたソースプログラムは手続き型言語としても解釈可能であり、幅広い応用が期待される．

Arithmetic pipeline process description language “mhdl” and the compiler

Signal Process Logic Inc.

Yuzo Seo

Abstract

In arithmetic pipeline processor, the signal wire and the computing unit can be individually specified. For such system, we are developing an optimizing method focusing effective bits. This report treats the language “mhdl” to describe the target pipeline process for the method. The language has limited rule for easy learning. In consideration of the editing environment of today, we put grammatical role on line-feed and the indent for better readability. Also, redundant descriptions were eliminated to simplify the source code. The “mhdl” source code can be interpreted also as a procedure-oriented source code, so the applications are widely expected.

1. はじめに

レジスタを介して演算器を直列に接続して構成されるパイプライン処理は、CPU の高速化技法以外にも、ディスクドライブの信号処理やビデオプロセッサなど、高速処理が要求される分野で古くから利用されてきた．最近では FPGA の登場により、高速演算処理を必要とする幅広い領域にその利用分野を拡大している．

パイプライン処理に用いられる多数の演算器は個別に仕様を定めることができ、浮動小数点数を用いれば、有効桁数に基づく簡素なアルゴリズムでこれらを最適化できる[1]．我々は、この手法を応用した数値演算論理設計支援システムの開発を進めているが、本報ではこれに仕様を与えるための、演算仕様記述言語 “mhdl” について報告する．

mhdl は数値演算部分の論理設計支援に特化した言語であり、一般的な数式の形式で仕様を与える. 処理系はこれを受けて、入力値に所望の演算処理を施して出力するモジュールを、Verilog HDL のソースコードとして出力する.

mhdl の仕様制定に際しては、小規模用途の多い FPGA 分野に配慮して、プログラミングの容易さを追求した. 言語規約は必要最小限にとどめるとともに、記述の簡素化を図った.

2. mhdl の特徴と背景

2.1. 数値型の扱い

mhdl は数値型を規定しない. コンパイラは、コンパイル時点で与えられる入力数値型に基づき、有効桁のみを処理するように内部の数値型を自動決定する.

今日の機械計算の世界では、有効桁という概念はあまり意識されていない. しかしながら、理工学の分野で有効桁は重要な概念であり、機械計算において有効桁に配慮することは、本来あるべき姿といえよう.

有効桁を処理するためには、浮動小数点数を扱う必要がある. 浮動小数点数は、数値の表現範囲が広い半面、これを扱う論理が複雑になる. 本手法では、指数部、仮数部ともにビット単位で最適化しているが、固定小数点数と比較して論理規模が増大することは避けられない. この短所はあるものの、安全性と見通しのよさを優先して、浮動小数点数を採用している.

2.2. データフロー

mhdl は、データフローのみを記述するため、状態の保持やフィードバックは扱うことができない. これらの機能が必要な場合は、別途準備された手続きを用いるか、生成した論理の外部で対処する.

フィードバックの禁止は“参照される変数はそれ以前に定義されなければならない”という規定により実現している. 変数の定義は、代入文左辺もしくは手続き宣言文の入力部に変数名を記述することで行われ、定義されることで変数に対応する信号線の源が決まる.

変数定義が使用に先立つことから、処理はソースコードの記載順に従って一様に進む. このため、mhdl で記述されたプログラムは、逐次処理手続きと解釈することもできる.

3. mhdl の言語仕様

3.1. プログラムの要素

ソースコードの記述に用いられる文字は、英字（大小文字を区別し“_”を含む）、数字、記号に大別される. 記号は、演算子、括弧、空白、“,”、“.”のいずれかである. 記号は、ソースコードを要素に区切る効果をあわせもつ.

名前は、英字に始まる英数字（英字と数字）の並びで、変数および手続きを識別する.

文は行末で区切られる. ただし、演算子、“(”もしくは“,”で終わる行は次行に継続するとみなされる. 文の行頭の空白数（レベル）は文の制御範囲を規定し、これよりレベルの深い文が続く限りをその文の制御範囲とする.

3.2. 手続き

プログラムは手続きにより構成される. 手続きの記述は手続き宣言文によって開始される. 手続き宣言文の形式は以下のとおりである.

```
name.(out1, out2,…) (in1, in2,…)
```

ここで、*name* は手続き名、*out1*, *out2*,…は出力変数名、*in1*, *in2*,…は入力変数名である. 出力変数名はピリオッドに続けて記述する. 出力変数が単一の場合は出力変数名を囲む括弧は不要である. 出力変数名を省略した場合は手続き名と同じ名前の出力変数がとられる. 入力変数名とこれを囲む括弧は省略できない.

手続きの定義は、その宣言文よりも深いレベルの文が続く間に行われ、任意の数の代入文と手続き定義が記述される. 手続きはその包含関係により木構造に配置される.

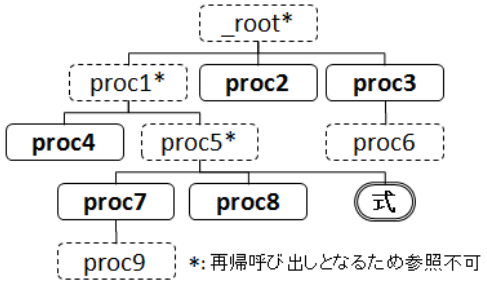

図1. 式から参照できる手続き（太字）

手続き名は、図 1 に示すように、その宣言文が置かれた手続きの範囲内で通用する. 変数名は宣言された手続きでのみ通用する. なお、手続きの再帰呼び出しはできない.

3.3. 最上位手続き

宣言文のレベルが最小の手続きを最上位手続きと呼ぶ. コンパイルは最上位手続きを対象に行う. 複数の最上位手続きがある場合は、コンパイルの際にいずれかを選択する. この選択は、コンパイラディレクティブを用いてソースコード上で行うこともできる.

他の手続きに含まれない手続きは、ダミー手続き“_root”の下に配置され、通常は_root 直下の手続きが最上位手続きとなる.

他の手続きに含まれない代入文が記述された場合は_root を最上位手続きとする. この場合、_root の手続き宣言文は自動的に形成され、代入文の左辺にのみ現れる変数が出力に、右辺にのみ現れる変数が入力に割り当てられる. この規約により、数式のみの記述も正当な mhdl ソースコードとみなされる.

3.4. 式と代入文

演算子とその優先順位を表 1 に示す. mhdl には型がないためビット演算子はなく、入出力を分離する必要から複合代入文も存在しない.

代入文は代入演算子“=”で接続された式である. 代入演算子に限り、複数の値を扱うことができる. この機能は、複数の出力をもつ手続きを使用するためのものである.

複数の要素はリストとすることにより一体

優先度	演算子	演算内容	項数	演算順序
高	^	冪乗	任意	右→左
	*, /	乗算、除算		不定
	+, -	加算、減算		
	<, <=, == !=, >=, >	比較	2	-
	&	論理積	任意	不定
	!	排他論理和		
	|	論理和		
低	?, :	条件式	3	-
	=	代入	任意	右→左

表1. 演算子

で扱われる. リストは、その要素を“,”で区切って括弧内に並べたものであり、手続き呼び出しの際の引数の記述や手続き宣言文の入出力変数記述にもリストの記法が用いられる.

3.5. 定数の記述

定数は数字の並びであり、先頭以外に一つの小数点を含むことができる. また、指数部指定文字“E”に続く数字の並びにより指数部を与えることができる. 指数部の数字には符号を付してもよい. 定数値は十進法で解釈される.

変数の数値型はコンパイラが自動決定するが、定数の数値型はソースコードから識別される必要がある. 特に、誤差の有無は数値だけからは識別できず、特別な規約が必要となる.

ソースコードが十進法で記述されるのに対して論理回路は二進法で実装されるため、小数部をもつ定数を誤差なく扱うことはできない. そこで、小数点もしくは指数部がある定数は誤差をもつ定数であるとみなし、そのいずれももたない定数を誤差のない定数であるとみなす. 文字“E”のみを定数の後に記述すれば、“E0”と解釈され、誤差をもつ定数とみなされる.

指数部が正の場合、および指数の底が 2 である場合には、指数部があっても誤差なく二進化できる. 誤差なし定数を指数形式で記述するには指数部指定文字に“D”もしくは“B”を用いる. “D”は、“E”と同様に底を 10 とする指

数部を指定し、指数部には正の値のみが許される．“B”は底を 2 とする指数部を指定し、指数部の値は正であっても負であってもよい．なお、指数部指定文字は、大文字でも小文字でも同等である．

4. 処理系

4.1. 処理系の概要

mhdl ソースコードは図 2 に示す手順により Verilog HDL コードに変換される．

パーサはソースコードを構文解析して、手続き宣言を木構造に配置するとともに、手続きに含まれる代入文を演算順位に従う式の木構造に配置する．次に、演算子を手続き配置リストに変換する．これらの処理は、一般的な手続き型言語のコンパイル手順と同様である [2, 3] ．

これにより生成された手続き配置リストは、手続き型プログラミングの場合は実行コードに対応しているが、パイプライン処理の場合には配置される実体には対応しておらず、配置方法を規定するテンプレートとして機能する．

次にコンパイラは、コンパイルの際に指定される最上位手続きから、その手続き配置リストをファンクションダイアグラムに変換する．ファンクションダイアグラムは、単純な演算機能（ファンクション）を信号線で接続したもので、表現は抽象的だが実体に対応している．

ファンクションダイアグラムは、ファンクション毎に準備された“コーダ”と呼ばれる変換関数によりノードグラフに変換される．ノードグラフは、Verilog HDL の代入文に対応するグラフで、整数演算を行うノードと整数を伝達するワイヤで構成される．

ノードグラフは、タイミング解析により必要な個所にレジスタが挿入された後、規則的な置き換えにより Verilog HDL に変換される．

ファンクションダイアグラム以降の処理内容に関しては前報[1]で詳しく扱ったので、本報

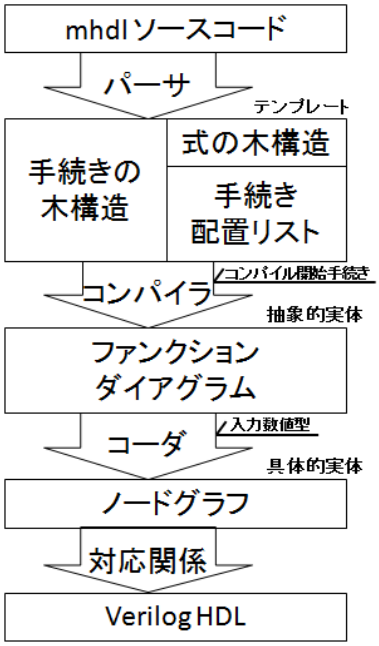

図2. コンパイルの流れ

では mhdl ソースコードをファンクションダイアグラムに変換する部分を中心に説明する．

4.2. 手続きクラス

手続きクラス“Proc”はプログラムの論理構造に従って各要素を配置するためのデータ構造で、その主要な要素は次のとおりである．

- **手続き名、入出力変数名リスト**：手続き宣言文の記述に対応
- **上位手続きと内部手続きへのポインタ**：手続きの包含関係に対応
- **式のリスト**：代入文に対応
- **手続き配置リスト**：式を変換して形成

パーサは手続き宣言文を読み込むと Proc オブジェクトを生成し、手続き名と入出力変数名のリストをセットする．また、この宣言文を含む手続きとの相互参照を可能とするよう、上位手続きへのポインタをセットするとともに、上位手続きの内部手続きリストに、生成した手続きへのポインタを追加する．

4.3. 式の木構造と項

代入文は、一般のコンパイラ同様、演算子の優先順位に従って式の木構造に変換される．

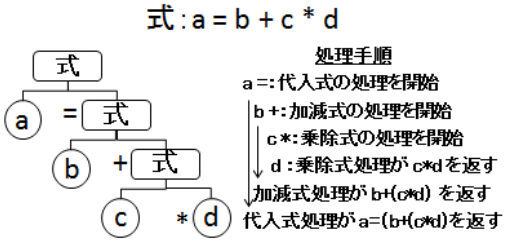

図3. 式の木構造化

式は同じ優先度の演算子で接続された項の並びであり、項は以下のいずれかである.

- 変数または定数
- 手続き呼び出し
- 単項演算子をもつ項
- 式

項は、変数または定数の場合は文字列として、その他の場合は対応するオブジェクトへのポインタとして表現される.

優先度の高い演算子で結ばれた式、または括弧に囲まれた式が現れた場合、これを再帰的に解析し、得られた式へのポインタを項とする.

図 3 は、“a = b + c * d”に対するパーサの動作を示したものである. パーサは、項とその次の演算子を読み込み、演算子の優先度に応じて式処理関数を再帰的に呼び出すことで、式を木構造に変換している.

式を木構造に整理した後、単項演算子の簡素化や定数の集約などの最適化を行う.

4.4. 手続き配置リスト

コンパイラは、式の木構造を手続き配置リストに変換する. 手続き配置リストは、手続き名と、入出力に接続される変数名からなる.

式とこれを変換して得られる手続き配置リストの一例を表 2 に示す. この例では、4 項からなる加減式が 3 つの手続き配置に変換される. 手続き名_add は処理系が提供する加算の標準手続きである. 同様の標準手続きは他の演算子に対しても提供されている.

式を手続き配置に変換する関数は式の優先順位ごとに準備されている. この変換関数は、

式 : a + b + c + d

手続き名	_add	_add	_add
入力	a, b	c, d	_var1, _var2
出力	_var1	_var2	_var3

表2. 式と手続き配置リスト

加減式のように項の数が任意である式に対しては演算を二分木の形に配置することで遅れ時間を最短とし、乗除式に対しては乗算を先に行って最後に一度だけ除算を行う形とすることで論理資源消費を抑える.

4.5. ファンクションダイアグラムの形成

コンパイラは、コンパイル時に与えられるコンパイル開始手続きから、手続き配置リストをファンクションダイアグラムに変換する.

ファンクションダイアグラムはそれ自体が一つのファンクションであり、その入出力ポートには外部リンクと内部リンクが接続される. コンパイル開始手続きから形成されるファンクションの外部リンクは外部入出力となる.

ファンクションダイアグラムであるファンクションを形成する際、入出力ポートにはこれと同名の内部リンクが接続される. 入力ポートの内部リンクは定義済みリンクリストに登録され、出力ポートの内部リンクは出力リンクリストに登録される.

次に、手続き配置リストに従い手続きを配置する. これにはまず、配置する手続き名を、手続き宣言の木構造および標準ファンクションリストから検索し、手続き宣言の木構造に同名の手続きが見出された場合にはその手続きを再帰的にファンクションダイアグラムに変換して配置し、標準ファンクションリストに見出された場合には該当する標準ファンクションを配置する.

こうして配置されファンクションの入出力には、手続き配置リストに記述された入出力変数と同名のリンクが接続される. このとき、入

力ポートには定義済みリンクリストのリンクが接続される．出力ポートには、出力リンクリストにあればこれを、ない場合は新たなリンクを形成して接続し、いずれの場合もこれを定義済みリンクリストに追加する．

4.6. Verilog HDL への変換

ファンクションダイアグラム以降の処理は前報[1]に詳しい．以下はその概要を紹介する．

ファンクションには“コーダ”と呼ばれるコード化関数が付随している．コーダは入力ポートに接続されたリンクの型が決定した段階で、これに対応する演算論理を形成する．

ファンクションダイアグラムのコーダは、その内部に配置されたファンクションのコーダを、変化が生じている限り呼び続ける．

コーダは必要に応じてモジュールを形成し、ノードを追加する．ノードは Verilog HDL の代入文に対応するもので、左辺のワイヤ、右辺のワイヤのリスト、および演算の種別からなる．

ノードグラフに対して、タイミング制約を満足するようレジスタが挿入される．最後に、レジスタの有無とポートか否かに応じて、Verilog HDL のワイヤ宣言文、アサイン文、レジスタ宣言と同期代入文の組合せに適宜変換される．

5. まとめ

本報では、パイプライン処理による演算論理を記述するための新言語 mhdl を提案し、その言語規約と処理系について解説した．

mhdl の言語仕様は小規模用途の多い FPGA 分野に配慮して、プログラミングの容易さを重視して定めている．このために mhdl の処理系は、有効桁という概念を用いて入力数値型から内部の信号線と演算器を最適化し、タイミング制約も自動的に解決している．

このような処理系を前提とすることから、mhdl では数値型に依存する演算子は利用できない．また、mhdl による記述はデータフローに限定され、フィードバックや複合代入文も記述できず制御文も存在しない．

フィードバックを禁止するため“参照される変数はそれ以前に定義されること”との言語規約を設けた．これにより、mhdl で記述されたソースコードは、パイプライン処理としても、逐次処理としても解釈することができる．

mhdl は、論理構造を表す字下げを言語規約に取り込んだ．こうすることで、読みやすいソースコードを簡潔に記述することができる．

mhdl の処理系は、論理設計支援システムとして製品化され、評価版も公開中である[4]．今後は、より効率的に演算論理仕様が記述できるよう、標準手続きの拡充などの改良を続けていく計画である．

FPGA の高機能化は進行中であり、スループットが要求される数値演算の分野でもパイプライン処理の利用が拡大する可能性が高い．

一般の数値演算を記述するためには、制御文や状態の保持も必要であり、逐次処理で実装される機能も記述できる言語仕様とする必要があろう．mhdl を用いて一般的な数値演算を記述する手法と、これを可能とするための mhdl の言語仕様拡張は、今後の大きな検討課題と考えている．

引用文献

[1] 瀬尾雄三: 浮動小数点処理を含む論理設計支援システム，情報処理学会シンポジウムシリーズ, Vol.2010, No.7, pp.3-8, 2010

[2] Aho, A.F, et.al.：Principles of Compiler Design, Addison-Wesley series in computer science and information processing, 1977

[3] 中西正和，他：やさしいコンパイラの作り方，共立出版，1980

[4] http://signal-process-logic.com