

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

mhdl 開発の背景

CodeSqueezer: ボタンを押すだけで数値演算論理を形成

パイプライン処理演算論理 (black box)

$$y = a * x^2 + b * x + c$$

演算仕様を言語“mhdl”で記述

ツールの想定用途と要求

- FPGAを用いた小ロット製品の設計
- ツールへの要求
 - + 安価で生産性の高いツールであること
 - + 生成される論理の性能は優先しない

ASIC vs FPGA

コピー数順に配列したデザイン

手続きの木構造と手続き名の通用範囲

手続き宣言の包含関係により、手続きは木構造を形成する

手続き名は、その宣言が置かれた手続きの範囲内で通用する

みえない root: 処理系が準備

- root直下に式を記述した場合 rootが最上位手続きとなる (手続き宣言文は自動形成)
- 最上位手続き
- コンパイル開始手続きとなりえる式を含むのはproc5, proc1, root
- この下で宣言された手続きが参照可能
- 再呼び出しはできない
- 式を含む手続き(*)は参照不可

生産性向上のための課題と解決策

演算アルゴリズム

- IPの選択、パラメータ設定は複雑
- ⇒ 演算子を用いた数式記述“mhdl”

数値型の選択

- ビット幅、符号の有無を個々に設定
- 固定小数点数の取り扱いは複雑
- ⇒ 有効桁に着目した数値型の自動最適化

タイミングの解決

- レジスタの自動挿入

mhdlの言語仕様: 代入文と式

演算子: 右図

代入文: 代入演算子をもつ式

- 代入に限りリストを扱う
- 例: $(x, y) = \text{div2}(a, b)$
- 記述順序
- 参照される変数はそれ以前に定義される必要がある (フィードバックの禁止)
- 定義: 代入文左辺または手続き定義文の入力宣言部に変数名を記述すること

定数の記述: 右図

演算子	演算内容	演算結果
+	加算	不定
-	減算	不定
*	乗算	不定
/	除算	不定
%	剰余	不定
>	比較	不定
<	比較	不定
>=	比較	不定
<=	比較	不定
!	論理否定	不定
!	論理否定	不定
?	条件式	不定
=	代入	不定

指数部数値: 負号も可
E, D: 底10の指数部
B: 底2の指数部
e, d, b を用いてもよい

有効桁に着目した数値型の自動最適化

有効桁は現在の計算機では重視されていない

パイプライン処理: 個別最適化が効果的

- 信号ごとに専用の演算器と信号線
- 論理資源の消費削減にも有用
- 浮動小数点数を扱う必要がある

特徴

- 判断基準が明確: 機械的処理が容易
- 固定小数点型に比べ論理規模は増大
- ⇒ 生産性重視とのコンセプトに合致

mhdlの処理系: 概要

mhdl ソースコード

パース

式の木構造

手続き配置リスト

コンパイラ

ファンクションダイアグラム

ノードグラフ

コード生成

Verilog HDL

mhdlの言語仕様: 基本思想

高生産性: 容易なプログラミング

- 高い抽象性、階層化、型なし言語
- 簡素で読みやすいソースコード
- + 改行と字下げに意味をもたせ; {} を排除
- + 複数の値を返す手続きを簡単に記述する: リスト

処理系とパイプライン処理に由来する制約

- 入出力が同じ演算子、数値型に制約のある演算子は使用不可
- フィードバックの禁止: タイミング制約を自動解決するため動作不定になる
- ⇒ Cとは異なる言語仕様

mhdlの処理系: 手続きの処理

手続きクラス (class Proc): 手続き定義毎に生成

- 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, 下位手続きへのポインタ: 手続き包含関係に対応
- 式のリスト: 括弧と演算子に基づいた木構造に形成
- 手続き配置リスト: 式のリストを変換して形成

式の木構造

- 式: 同じ優先度の演算子で接続された項の並び
- 項: 変数または定数、手続き呼び出し、単項演算子に先導された項、式、のいずれか

mhdlの言語仕様: プログラムの要素

使用される文字

- 英字: 大小文字を区別し, "." を含む
- 数字: 0-9
- 記号: 演算子、括弧、空白、"/", "*", ":", "}"

名前: 英字に始まる英数字列。個々の変数、手続きを識別

コメント: "//" から行末まで、および "/" と "*" の間

文

- 行末は文を区切る。ただし、演算子 "("、"["、"{" で終わる行は次行に継続
- 文のレベル: 行頭の空白の数
- + 文の制御範囲: その文よりもレベルの深い文が続く限り
- 文の種類: 手続き定義文、代入文 (現在のところこの2種だけ)

ファンクションダイアグラム (FD) の形成

手続きクラスからFDの形成: 右図

定義済み変数 (= リンク) の管理

- リストを初期化し入力変数を登録
- 代入文左辺を定義済みリストに追加
- 右辺変数は定義済みリストから検索

FDからVerilog HDLコードの形成

- 各コード化関数 "coder" によりノードグラフに変換
- 入力型を参照してコードを形成、出力の型を決定
- ノードグラフに対してタイミング処理を行い、次いでVerilog HDLに変換する
- 詳細は、前報 (DAシンポジウム論文集2010, p3-8) 参照

mhdlの言語仕様: 手続き

手続き宣言文: 手続き定義を開始する

形式: `name.(out1, out2,...)(in1, in2,...)`

- + name: 手続き名
- + (out1, out2,...): 出力変数名。ピリオド "." に続けて記述
 - 単一の場合は括弧は不要
 - 全て省略された場合は手続き名が出力変数名となる
- + (in1, in2,...): 入力変数名
 - 単一の場合も括弧は省略できない

手続き定義本体

- 手続き宣言文よりもレベルの深い文が続く間に記述
- 任意の数の代入文と手続き定義が含まれる

まとめと今後の課題

パイプライン演算処理を記述する新言語“mhdl”を提案した

特徴は、プログラミングの容易さ

- 有効桁に着目した数値型の自動最適化
- 改行と字下げに意味をもたせ、論理構造の把握を容易にした

今後の課題

- 信号処理分野での論理設計支援ツールとしての機能拡充
 - + 標準手続きの拡充: 関数、時系列処理、...
 - + ユーザー側での標準手続き開発環境を提供: Function Developer's Kit
- 一般的な数値演算分野を狙った言語仕様の拡張