浮動小数点処理を含む
論理設計支援システム

シグナル・プロセス・ロジック株式会社

瀬尾雄三

シグナル・プロセス・ロジック㈱のご紹介

- 設立 2009年6月、営業開始 2009年8月
- 主要業務：ソフトウエア“***CodeSqueezer***”の制作・販売

CodeSqueezer シリーズ

- 演算を多く含む信号処理用のFPGA論理設計支援ソフト
- 小規模な利用者が多いというFPGAの特性に配慮して、
  “安価で手軽に使える”ことを特徴としている
- ***CodeSqueezer Basic***
  - 固定小数点数の四則演算モジュールを自動形成
  - 2010年2月販売開始
- ***CodeSqueezer Floating***
  - 浮動小数点数を含む数値演算論理を自動形成
  - 2011年1月販売開始予定 ⇒ 今回報告

FPGAによる浮動小数点処理

- 浮動小数点数処理はCPUなどのデジタル演算では一般的だが、FPGAでは固定小数点処理が多く行われている
- IEEE754標準の浮動小数点表現をFPGAで利用する場合、
  - 型の種類が少なく無駄な信号線が配設される
  - hidden bitを採用しているため0付近での別処理が必要
  - 異常の表示を扱うための処理が複雑

  などの問題がある

IEEE754標準の浮動小数点表現は、単一の演算器を用いるノイマン型計算機に最適化されている
並列信号処理を行うFPGAには、IEEE754標準とは異なる浮動小数点表現が最適なのではないか？

並列論理に浮動小数点処理を含めることの得失

◎　ダイナミックレンジが広い

×　一般にバレルシフタが必要となり論理が複雑化する

⇒　最小の論理規模で浮動小数点処理を実現する工夫が必要

◎　有効桁に着目した数値型の自動決定が可能

並列論理に最適化した浮動小数点表現方式

- 指数部、仮数部のビット幅を任意の幅に設定できること
- hidden bitは使用せず、負数を2の補数表現とすること
- 異常表示のためには専用の信号線を設けること

⇒　論理の簡素化

数値型自動決定アルゴリズムへの対応

今回採用した数値表現方式

- 任意数値型(Link)をハードウエア信号線(Wire)の組で表現する
- 誤差をもたない整数(int)と誤差をもつ実数(real)の二種類を準備

class Link:任意型を表現する信号線の型

内容	変数名	型	int	real	値
誤差なしフラグ	certain	bool	true	false	-
仮数部	man	Wire	○	○	任意
指数部	exp	Wire	×	△	任意
異常表示	of	Wire	△	△	1 | 0
	uf	Wire	△	△	1 | 0

○:必須、△任意、×:指定不可

値 = man x 2^{exp}

class Wire:
Verilogのwireに対応する数値型

内容	変数名	型
符号反転フラグ	neg	bool
定数項	bias	long long
最小値	min	long long
最大値	max	long long

Wire値 = wire値 x (neg ? -1 : +1) + bias

min ≦ 下線部 ≦ max

符号の有無とビット幅はmin, maxで決める

of	uf	状態
0	0	正常
1	0	オーバーフロー
0	1	アンダーフロー
1	1	数値化不能 (NaN)

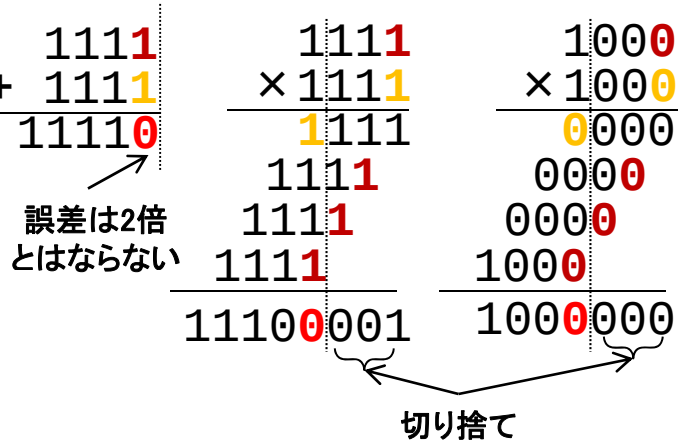
数値型の自動決定

- int：演算の結果得られる全てのビットを後段に伝達
  - 演算結果の最大値と最小値で型を管理する
- real：意味ある情報のみを後段に伝達
  - 有効桁数ではなく誤差に注目する

概念

不確定性指数 (UI) ≡ 信号の誤差/桁の重み

- UIが管理値 (lmt) 以下の桁を後段に伝達する
  最下位ビットのUI：0.5 lmt ≦ UI ＜ lmt
- 安全サイドで処理
  入力最下位のUIは最小値（0.5 lmt）と仮定
  最下位ビットのUIが2倍以上となれば切捨て

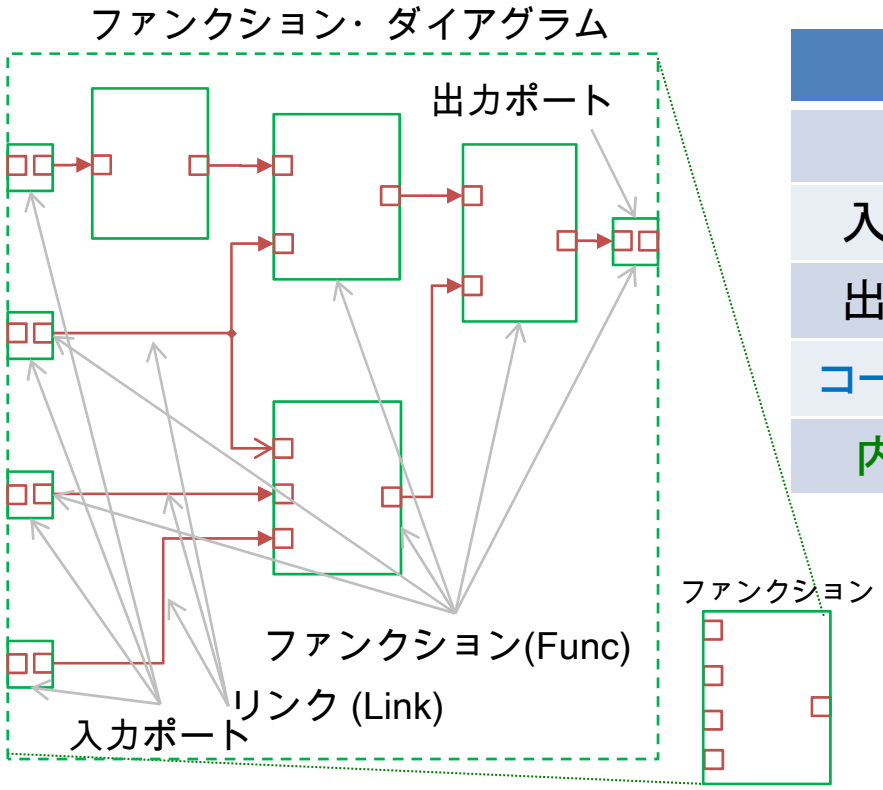


加減算結果は、全てのビットを後段に伝達する
乗算は、[入力の最大有効桁数 − 1桁] を切り捨てる
適宜切捨て処理を挿入して誤差の増加を抑制する

ファンクション・ダイアグラム

抽象的な信号処理論理仕様の記述

- 単純な演算機能（ファンクション）を抽象的信号線（リンク）で接続
- ファンクション・ダイアグラムもファンクションとして定義される



class Func

内容	変数名	型
名称	name	String
入力ポート	inports	List<Port>
出力ポート	outports	List<Port>
コード化関数	coder	Coder
内部構成	funcs	List<Func>

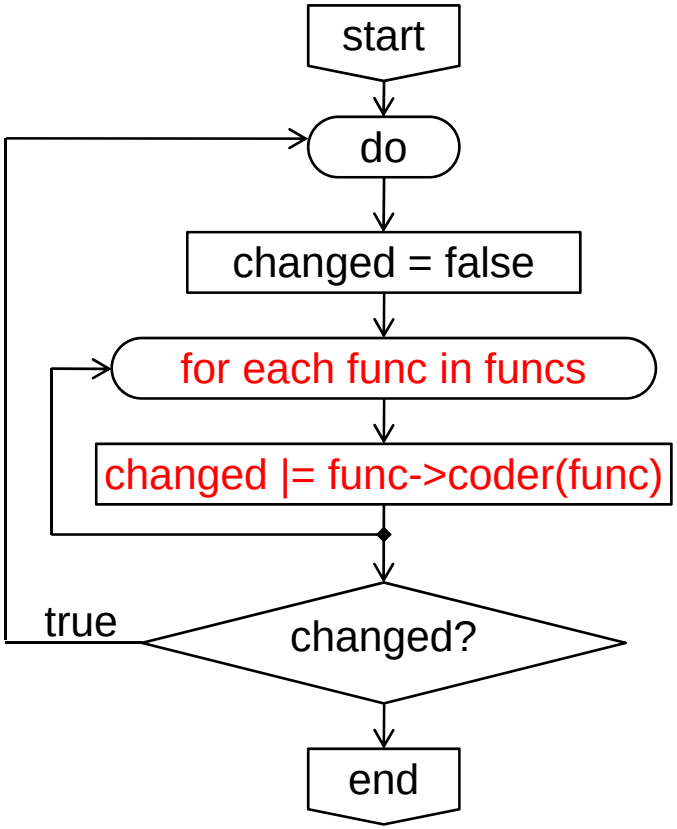
class Port

内容	変数名	型
名称	name	String
接続リンク	link	Link
対応するファンクション	func	Func

コード化関数はファンクションをVerilogHDLに変換
右表緑字部分はファンクション・ダイアグラムを定義

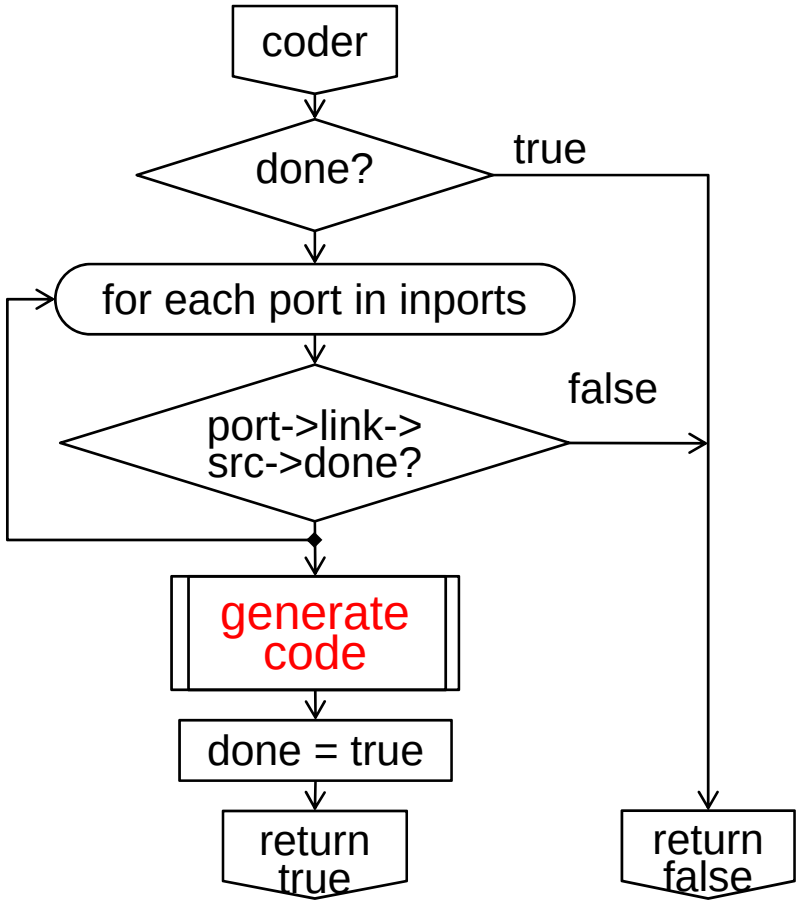
ファンクションのコード化手続き

ファンクション・ダイアグラムの処理



ファンクション・ダイアグラムに含まれる
各ファンクションのコード化関数を、
変化がある限り呼び出し続ける

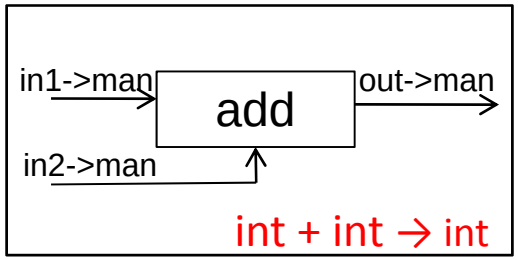
コード化関数の処理

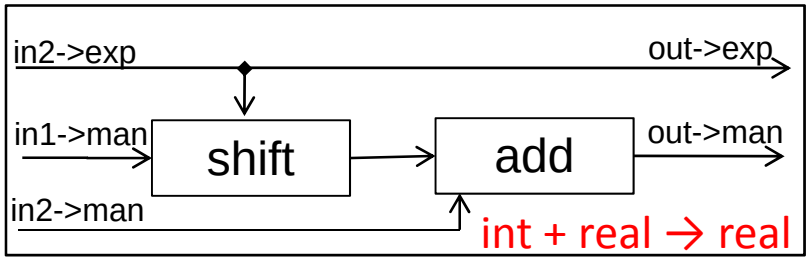


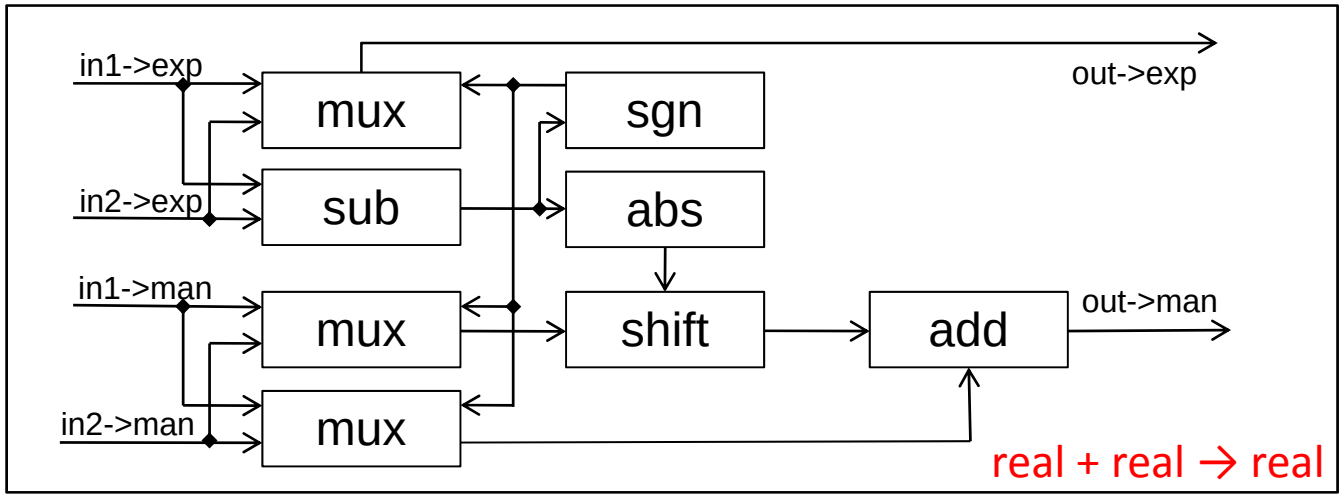
コード化が未了で、入力ポートに接続された
全リンクの型が決定していたらコードを形成

コード化関数：加算ファンクションのcoder

- 入力の int/real の組合せに応じて処理を切り替える
- ワイヤに対するコード化関数を順次呼び出す
- ワイヤのコード化関数は、
  - ノード（Verilog HDLの代入文に相当）を形成し、
  - 出力ワイヤの型属性（neg, bias, min, max）を決める

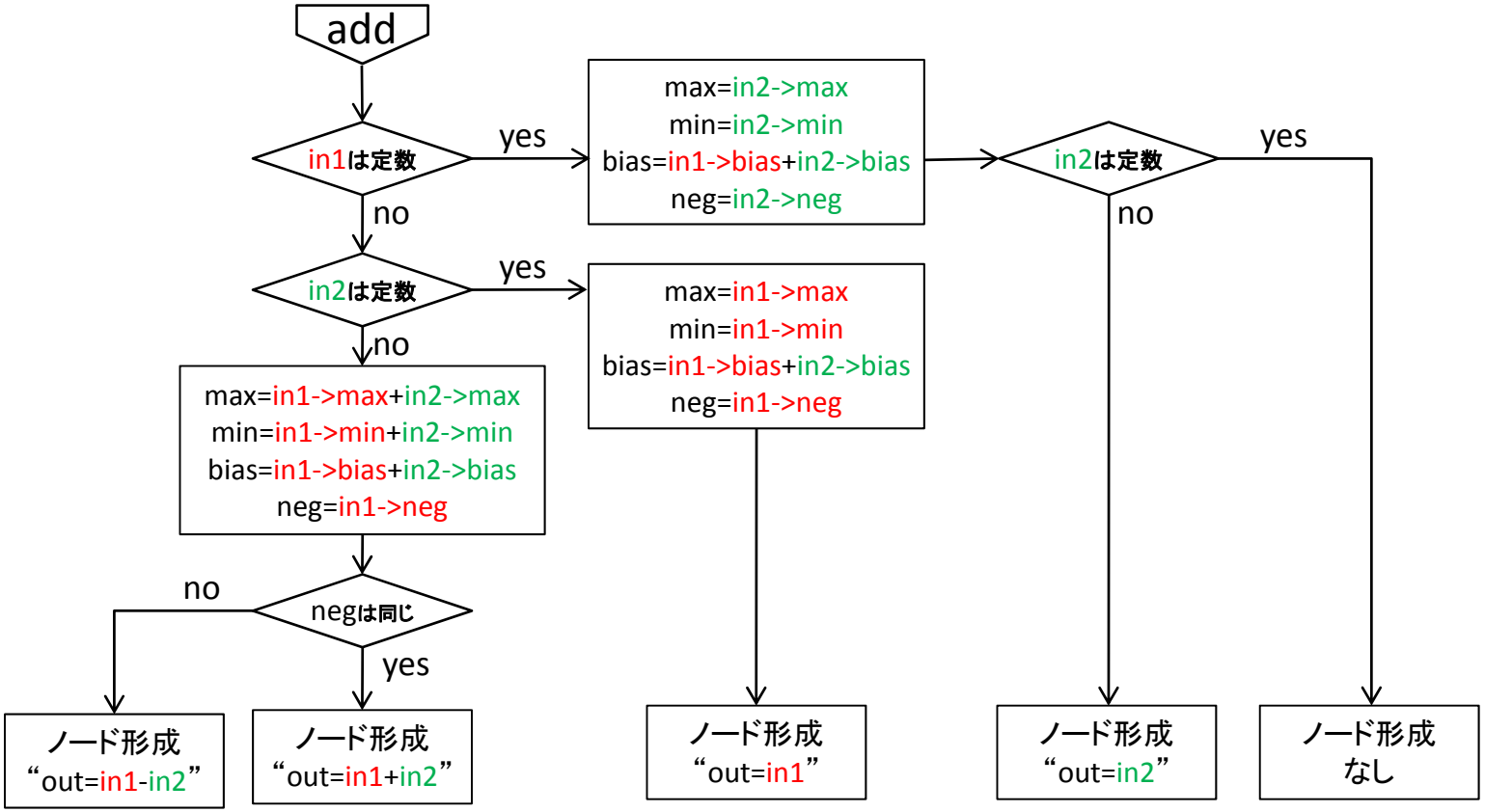




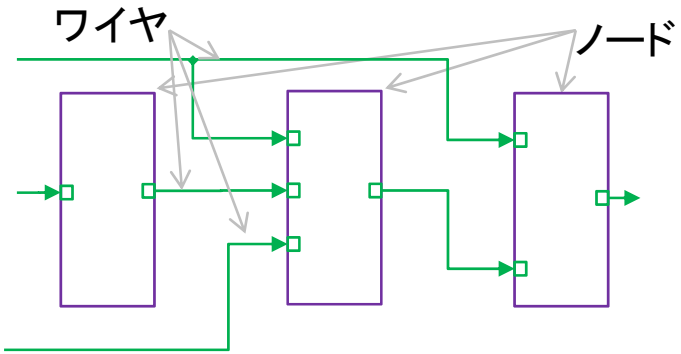


ワイヤのコード化関数：加算

- 双方の入力のmin, max, biasの和を出力のmin, max, biasとする
- negが異なれば減算、同じなら加算のノードを出力する
- 入力の一方が定数なら他方を出力にコピーするノードを形成



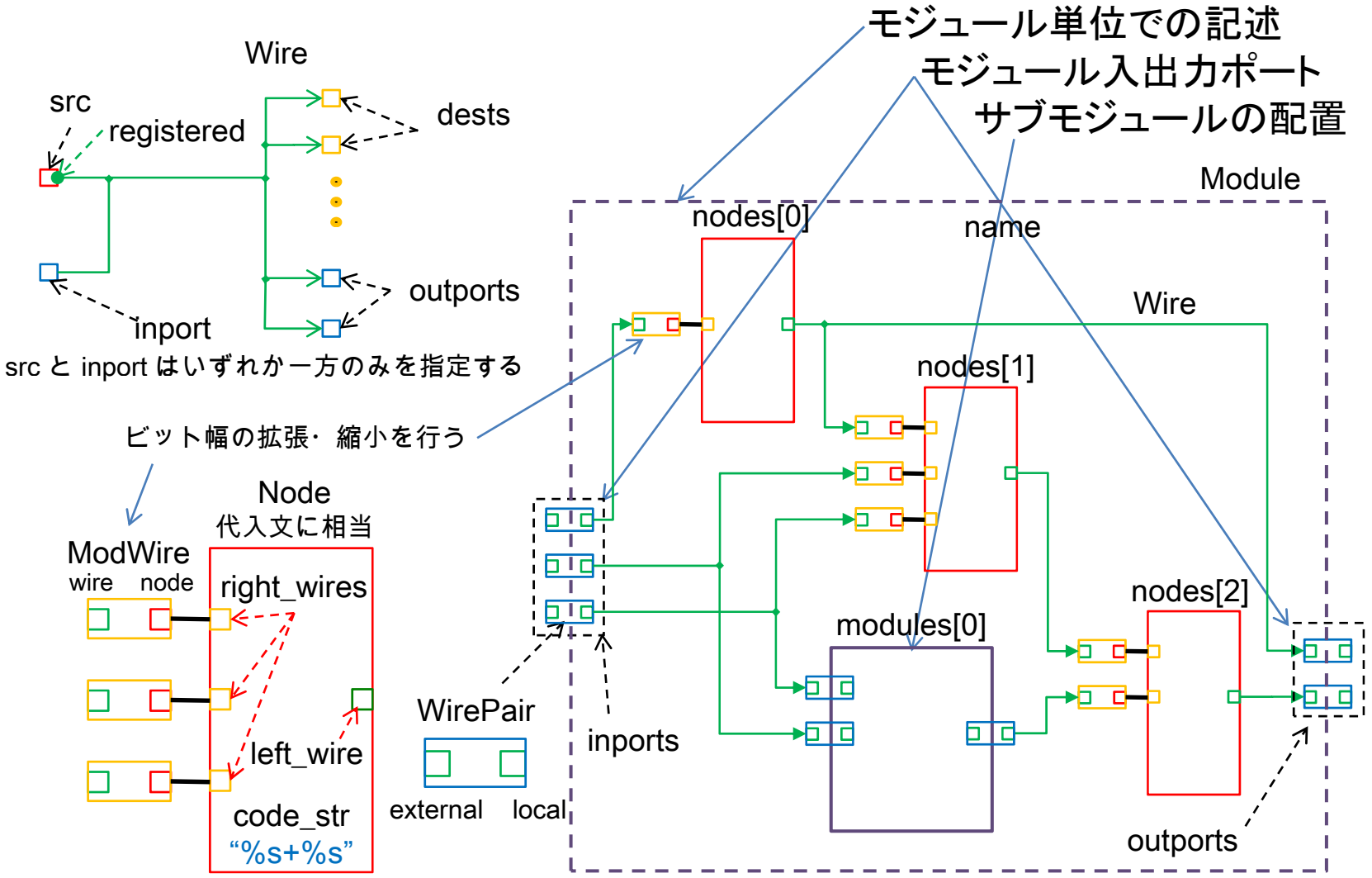
ハードウエア記述のデータ構造：ノードグラフ



- ノードグラフ
  - ノードをワイヤで接続して構成される有方向グラフ
- ノード
  - FPGAの論理要素1段に対応
  - Verilog HDLの代入文で記述される
  - 出力側にレジスタを挿入することができる

  → パイプライン化の最小単位となる
- ノードグラフの解析によりタイミングを解決
- Verilog HDLに変換しやすいデータ構造で表現

Verilog HDLに対応したノードグラフのデータ構造



タイミング処理（レジスタ挿入）

ノード（class Node）

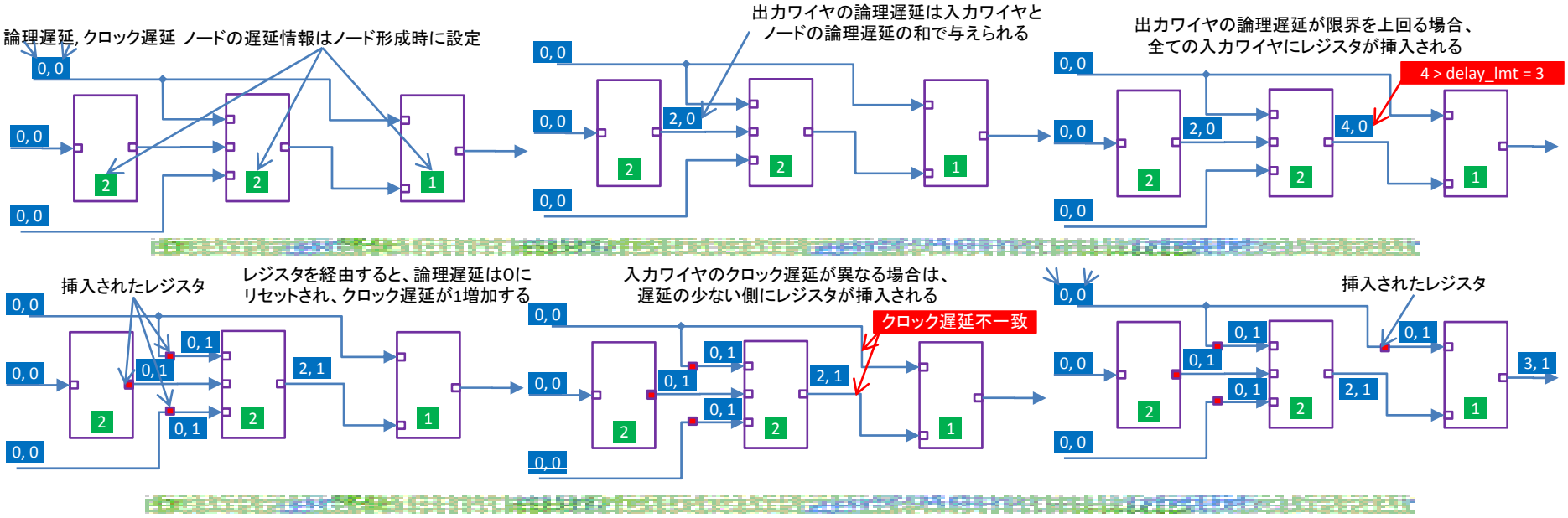
シンボル	type	用途
logic_delay	double	論理遅延時間* 2
clock_delay	int	クロック遅延**

*: 入力側の配線遅延時間を含む
**: 遅延要素で使用される

ワイヤ（class Wire）

シンボル.	type	用途
registered	bool	レジスタ付き入力
logic_delay	double	論理遅延時間
clock_delay	int	クロック遅延

3, 1



- 入力のクロック遅延が一致しない場合は遅延の少ない入力ワイヤにレジスタを挿入
- ノード出力の論理遅延時間 = 入力の論理遅延時間最大値 + ノードの論理遅延時間
- ノード出力の論理遅延が規定値以上の場合は全ての入力ワイヤにレジスタを挿入
- レジスタを経由すると、論理遅延はリセットされクロック遅延が1つ増加する
- レジスタが挿入できない場合は代入のみを行うノードを挿入

Verilog HDL 代入文の形成

ノードをVerilog HDLに変換する際には、左辺Wireが
・モジュールの出力ポートに接続されているか否かと、
・レジスタが挿入されているか否か
の組合せにより、異なる形式の代入文が形成される。

outport	registered	Verilog HDL codes*
no	false	wire [*msb*:0] *name* = 式;
no	true	reg [*msb*:0] *name*; *name* <= 式;**
yes	false	output [*msb*:0] *name*; *** assign *name* = 式;
yes	true	output reg [*msb*:0] *name*; *** *name* <= 式;**

*: 左辺Wireが符号付の場合は定義に“signed”が挿入される
**: クロック同期部分に記述される
***: 入出力ポート宣言はポートリストに基づいて行われる

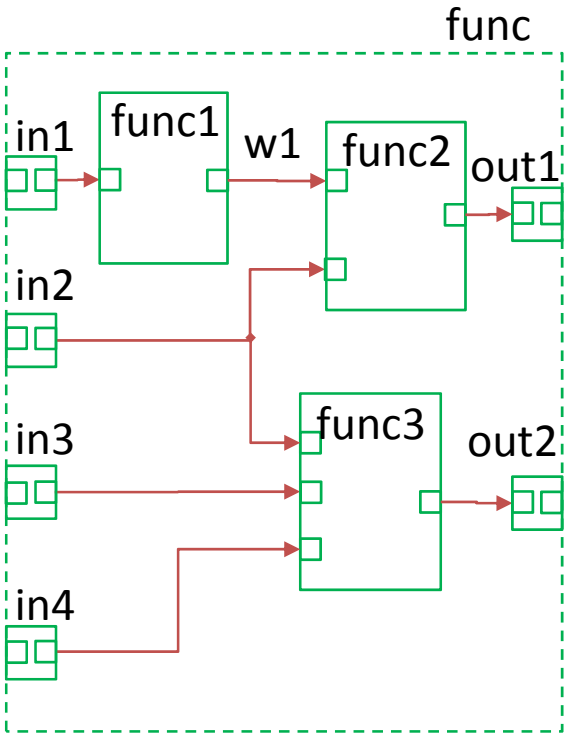
ファンクション・ダイアグラムの入力

- 論理仕様の入力は言語形式で行うのが効率的
- このための言語“MHDL”を提案したい

MHDL文法（案）

- 構成要素：ファンクション宣言文と代入文
- リンクが複数の場合はリスト表記する
  （かっこ内にコンマで区切って記述）

```
func.(out1, out2) (in1, in2, in3, in4)
    w1 = func1(in1)
    out1 = func2(w1, in2)
    out2 = func3(in2, in3, in4)
```



言語仕様に関しては、原案作成後、幅広く意見を募りたい
このための論議の場として http://mhdl.org を準備した

数値表現方式の比較

タイプ	fixed	real	IEEE754
論理規模	○	△	×
応答時間	◎	△	×
ダイナミックレンジ	×	◎	◎
安全性・精度	△	◎	○～◎
設計容易	×	○	◎
誤差キャンセル	△	×	○

誤差のキャンセル

(1) x = a + b
(2) y = x – b
→ bの誤差がキャンセルされる

b の誤差が大きい場合、
(1) で無効桁を切り捨てると
誤差キャンセルが働かない

- IEEE754 標準は単一の演算器を用いるVon Neumann型計算機に最適
- 固定小数点表現は、応答時間に優れるが、安全性と精度を確保するための設計作業が複雑になる
- 今回提案する real の論理規模と応答時間は、fixed に比べて劣るが、IEEE754 標準に比べれば大幅に改善されている
- fixedもrealも、IEEE754 標準で可能な誤差のキャンセルには頼れない
  → 解法アルゴリズムで対処すべき問題と判断

まとめと今後の課題

- 並列処理に最適化した浮動小数点表現方式を用いた論理設計支援システムを開発した
- 浮動小数点数を用いることで、単純なアルゴリズムで抽象的論理仕様をハードウエア記述に変換できる
- 今後の課題は以下の通り
  - 信号処理論理設計支援システムとしての製品化
  - 抽象的論理仕様記述言語“MHDL”の規格化
  - 誤差キャンセルに頼らない演算アルゴリズムの自動形成手法 ⇒ 一般的な数値演算への応用

更新情報は以下のURLに掲載します
製品全般：http://signal-process-logic.com
MHDL関係：http://mhdl.org　(現在工事中)

ご清聴ありがとうございました