ハードウエア設計とSystemC

長尾文昭
naga046053@swan.sanyo.co.jp

三洋電機株式会社
セミコンダクターカンパニー

発表内容

(1)背景

(2)なぜSystemCを選んだか

(3)現状のSystemCの設計環境

(4)SytemCの設計環境の将来への希望

(5)まとめ

(1)背景

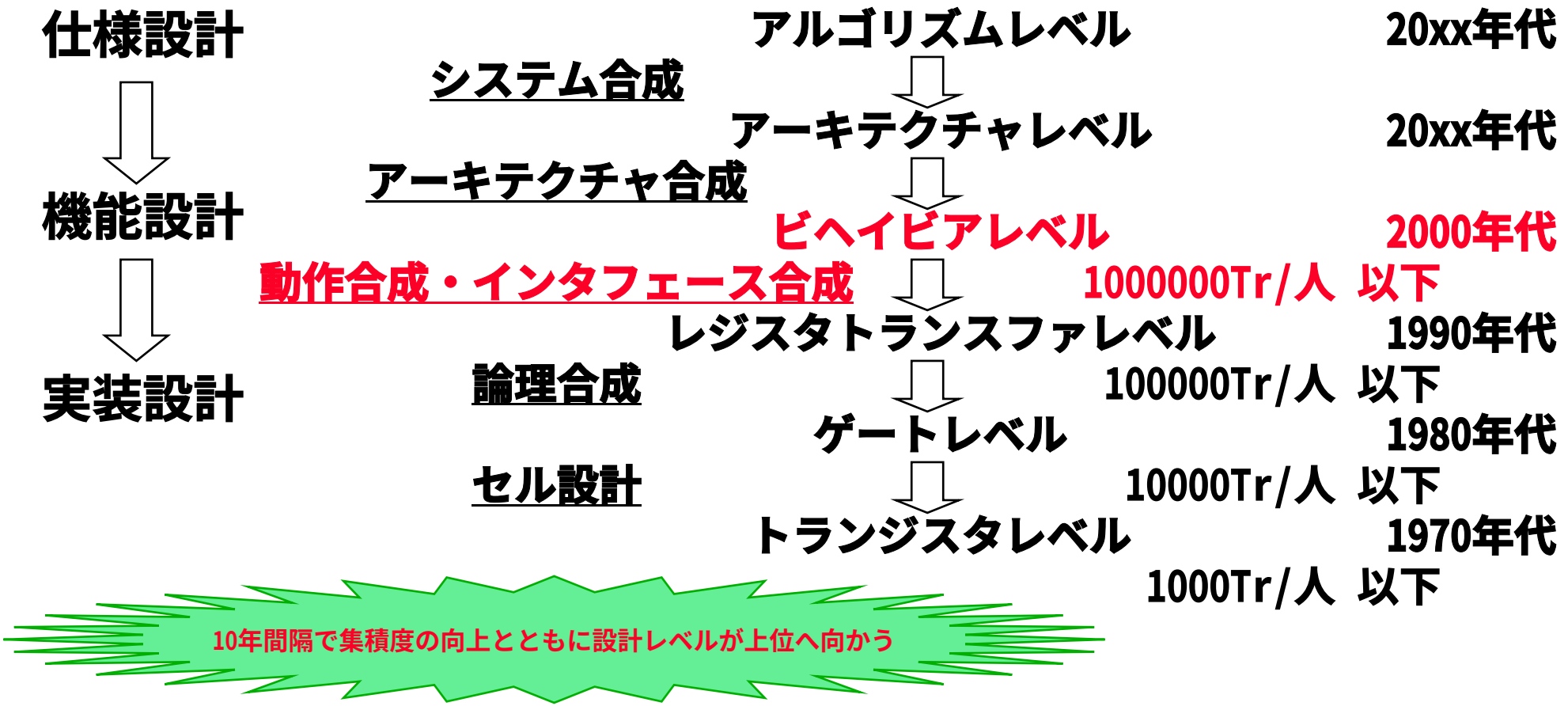
(2)なぜSystemCを選んだか

(3)現状のSystemCの設計環境

(4)SytemCの設計環境の将来への希望

(5)まとめ

デジタルLSI設計の設計レベル

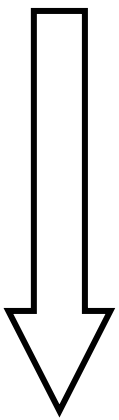


論理合成

1990年代に普及した設計ツール

データパスやステートマシンの設計

- 論理式とレジスタの接続を指定



- ブール代数式の簡単化
- 演算ライブラリの呼び出し
- レジスタの実装

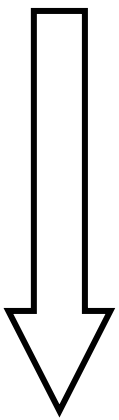
ゲートレベルの回路を生成

動作合成

2000年代に普及する設計ツール

信号処理回路の設計

- ブロックの動作と入出力のタイミングを指定



- スケジューリング
  演算、レジスタ、メモリインタフェース
- リソースシェアリング
  演算器とレジスタの共有化

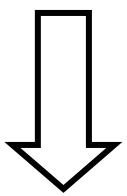
データパスとステートマシンを対で生成

インタフェース合成

あともう少しだが…

インタフェースの設計

- データ転送方法を指定
  同期転送、非同期転送、AMBAなど
- ブロック間の関係を指定
  送信機、受信機、バッファ、コントローラ



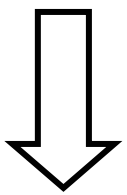
インタフェース回路を生成

アーキテクチャ合成

2010年代の設計ツール？

システムの設計(ブロックの配置と接続)

- ブロック要素を指定
  プロセッサ、メモリ、カスタム演算器など
- アルゴリズムの分割と分担を指定
- バス・アーキテクチャを指定
  同期転送、非同期転送、AMBAなど



ブロックの動作モデルとインタフェースモデルを生成
ファームウエアのコードを生成

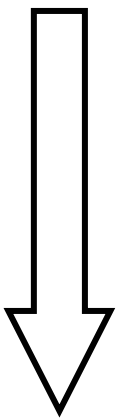
システム合成

まだまだ…

ブロック構成の最適化設計

・アルゴリズム(システムの動作)と入出力を指定

・各アーキテクチャでの性能の見積もり



アーキテクチャの選択
ソフトウエアとハードウエアの分離
アルゴリズムの分割と分担の決定

(1)背景

(2)なぜSystemCを選んだか

(3)現状のSystemCの設計環境

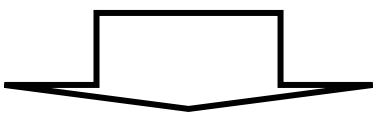
(4)SytemCの設計環境の将来への希望

(5)まとめ

アルゴリズム開発の変化

1990年前後

- C言語を使用。
- OSのパイプ機能やファイルを使って機能ブロックを細分化
  パイプ機能ではデータが片方向のみ
  ファイルの入出力では処理速度が低下
- 演算精度確認のため演算に対応した関数を作成
  アルゴリズムの視認性が低下



手続き型からオブジェクト指向へ

2000年前後

- C++言語を使用。(1998年に標準化が完了)
- クラスを使って機能ブロックを細分化。
- 演算子のオーバーロードを利用して実数演算以外も表現可能
  特殊な演算体系や演算精度を制限した演算体系を実現
  アルゴリズムの視認性、修正の容易さが向上

Verilog言語の機能不足

Verilog言語の概要（IEEE Std 1364-1995 -> 2001)

- C言語ライクなハードウエア記述言語(HDL)
- 機能ブロックをモジュールとして表現
- 信号の入出力をポートで行なう
- 並列処理を表現可能
- 共通機能は階層化か関数によって共有化

Verilog言語の不足な点

- 演算体系が整数演算に限定される
- データ表現がビットベクタのみ
- インタフェース記述の抽象化が困難
  -> SystemVerilogで改善されているが不十分

C/C++言語ベースの記述言語

シミュレーション環境を無償で入手できるもの

- SystemC　http://www.systemc.org/
- SpecC　http://www.cecs.uci.edu/~specc/reference/

ツールと一体で入手できるのもの

- CoWareC　(CoWare)
- Handel-C　(Celoxica)
- FDA-C (フューチャーデザインオートメーション)

LSIメーカが開発したもの

- Bach C (シャープ)
- BDL　(NEC)

C++言語ベースなのはSystemCだけ

SystemCの利点

C++言語のライブラリで実現

- ユーザ定義の信号型(演算体系)を使用可能
- まるめやオーバーフロー処理を備えた固定小数点型が利用可能
- C++/C言語で書かれた通常のプログラムと組み合わせ可能
- インタフェース記述の抽象化が可能

無償でライセンス

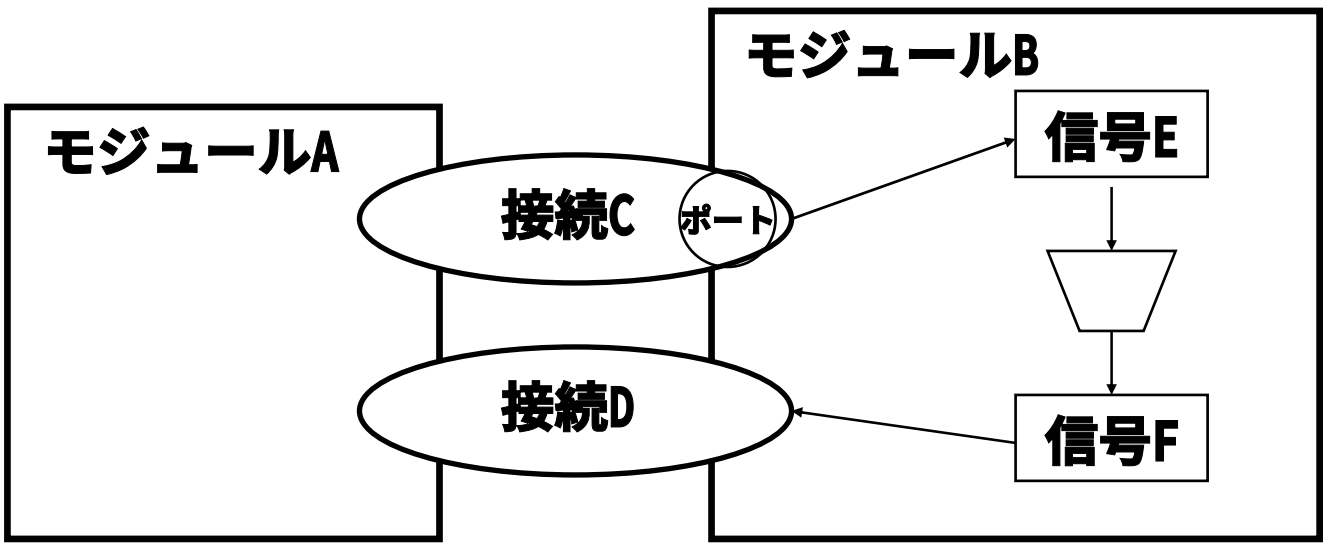
- 言語の普及や教育面で有利
- 複数の会社からツールが提供される

言語を非排他的な場で共同開発

- 言語の将来に対し能動的に対応可能

SystemCで利用できるオブジェクト指向

モジュール(ブロック)
接続(インタフェース)
信号(データと処理)



演算は信号に付随する処理(手続き)と考える。

(1)背景

(2)なぜSystemCを選んだか

(3)現状のSystemCの設計環境

(4)SytemCの設計環境の将来への希望

(5)まとめ

ハードウエア設計に必要なツール

検討中のもの

- デザイン入力とデバッグの統合環境
- シミュレータ
- 合成ツール

未検討のもの

- 設計制約/構文チェックツール
- 等価性チェックツール
- 検証率チェックツール

入手可能なツール　1

デザイン入力とデバッグの統合環境

- CoCentric SystemC Studio (Synopsys)
- Visual Elite (Summit Design)
- N2C System Designer (CoWare)

シミュレーションツール

- SystemC 2.0.1 リファレンスシミュレータ (OSCI)
- NC-SystemC (Cadance)
- VCS (Synopsys)
- N2C System Verifier (CoWare)
- Fast-C (Summit Design)

入手可能なツール ２

SystemC入力動作合成ツール

- CoCentric SystemC Compiler (Synopsys)
- Cynthesizer (Forte Design System)
- Design Prototyper (フューチャーデザインオートメーション)

SystemCからVerilogへの変換ツール

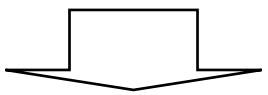
- FastC (Summit Design)
- N2C Advanced System Designer (CoWare)

Verilog入力動作合成ツール

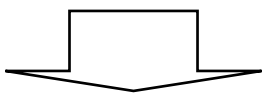
- Behavioral Compiler (Synopsys)
- G2C-AC (Get2Chip)

動作合成ツールの評価

- 500行程度の信号処理回路で評価
- 記述は各社の入力フォーマットに合わせて修正
- 設計制約を設定
  メモリ2個、乗算器1個、加算器1個、32クロック以下
  手設計で熟練者が時間をかけて実現できるレベル



- 制約条件を満たしたツールがある
- ツールによって得手不得手の項目がありそう



- 使い方の工夫で運用可能と判断
- 新規開発機種で適用開始

オリジナルツール sc2v

暫定的な対処

- 2001年2月開発スタート。この時点での合成可能な構文に不満
- sc2vはSystemCからVerilog-RTLへの変換ツール
- sc2vに論理合成および機能合成に類する機能はない
- SystemCのMETHODプロセスとCTHREADプロセスに対応
- 型合成や型変換を自動で行うのでSystemCで用意されている型であれば変換後の形について考慮不要
- SystemCの固定小数点型を限定付きながら対応
- オブジェクト指向構文に部分対応（ユーザ定義クラスなど）

あくまでも暫定的な対応 -> 要求事項やノウハウをOSCI合成WGへ

現状のSystemC設計環境

必要最低限が揃ったところ

デザイン入力とデバッグの統合環境

- 実用レベルのものが入手可能
- 選択肢あり

シミュレータ

- リファレンスシミュレータが使用可能
- その他の高速シミュレータも年内に立ちあがるだろう

合成ツール

- 動作合成は実用レベルのものが登場
- 市販ツールの入力構文がHDLレベルを超えず不満あり
  -> OSCI合成WGが始まったので今後に期待

(1)背景

(2)なぜSystemCを選んだか

(3)現状のSystemCの設計環境

(4)SytemCの設計環境の将来への希望

(5)まとめ

SystemCの特徴を生かした設計環境

協調検証のための設計環境

- アルゴリズム検証用モデルをリファレンスとして使用
- 専用プロセッサIP
- 専用ICEとのインタフェース

ハードウエア合成のための設計環境

- モジュール、インタフェース、ユーザ定義型の3種のオブジェクトに対応した統合環境
- 合成用構文の標準化
- ハードウエア合成用のルールチェックツール
- 合成前後の仕様の静的な等価性チェックツール

オブジェクト指向の記述スタイル

モジュール設計は、オブジェクト指向に近い

ユーザ定義型の利用
- 信号処理の流れと演算器の設計を別管理
- 記述しやすさや読み易さを向上させる

テンプレートや継承を使った設計管理
- 再利用性の向上
- 理解しやすさの向上

C++言語ベースである特徴を最大限に活用したい

ユーザ定義型の利用例

演算器の定義　-> 論理合成

```
class gf256{
  sc_uint<8> data;
public:
  gf256(int di){data=di};
  gf256 operator -(gf256);
  gf256 operator *(gf256);
  ...
};
```

実際の設計と同じように記述も分離
その設計工程中心に記述するのでわかりやすい

演算フローの設計　-> 動作合成

```
gf256 Gx, x;
Gx = (x - gf256(0))*(x - gf256(1))
    *(x - gf256(2))*(x - gf256(4));
```

(1)背景

(2)なぜSystemCを選んだか

(3)現状のSystemCの設計環境

(4)SytemCの設計環境の将来への希望

(5)まとめ

まとめ

背景

・論理合成では回路規模の増大から設計コストが莫大になる。設計手法が変わる10年毎の節目。

なぜSystemCを選んだか

・選択肢の中でオブジェクト指向の可能性が最も高い言語。

・言語そのものはオープンで複数ベンダのサポートが期待できる。

現状のSystemCの設計環境

・シミュレーションと動作合成は実用レベルにある。

SystemCの設計環境の将来への希望

・オブジェクト指向対応の設計手法と環境の確立

結び

システム設計もハードウエア設計もオブジェクト指向の時代

SystemCなら少しの工夫でそれが可能