Solution When You Need It

One Step Ahead

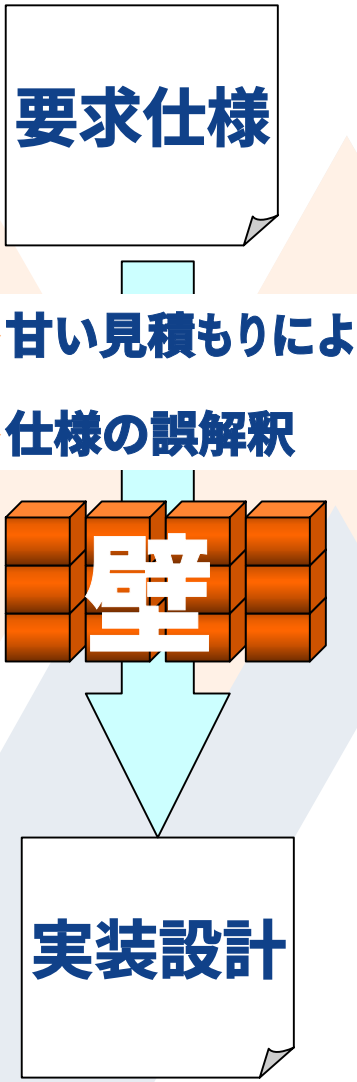
Summit社システムレベル設計ソリューション

C/C++/System C設計環境
VisualElite with System Design

2003年 11月

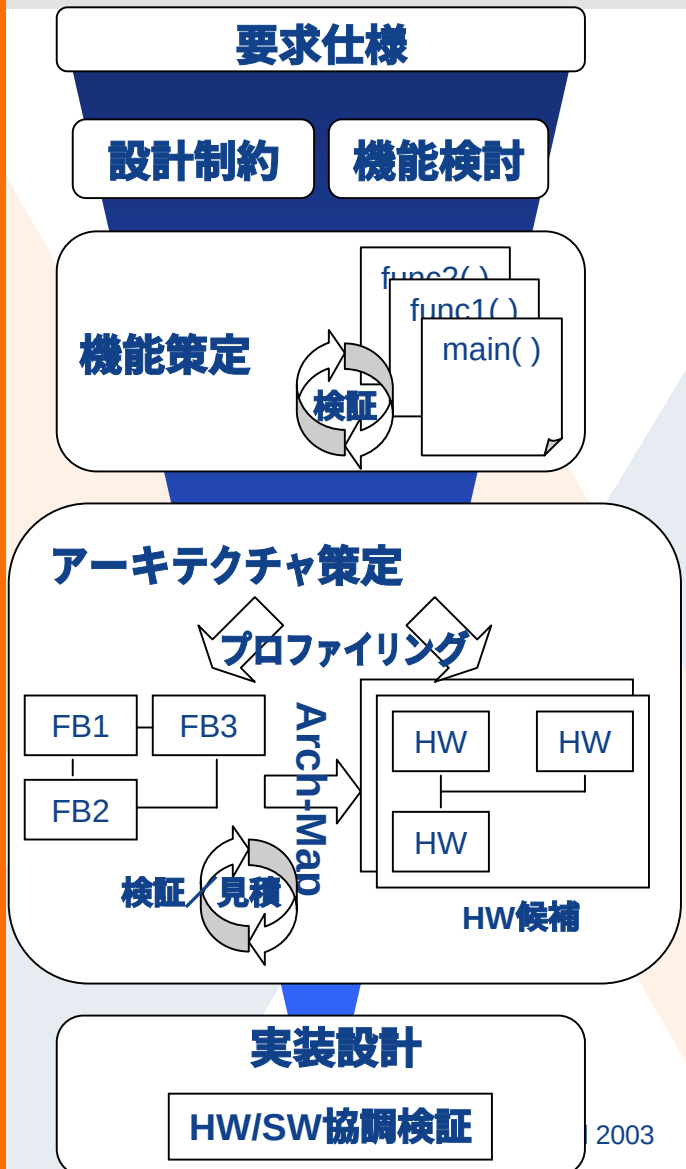
Summit Design Japan Co., Ltd

現状のシステムレベル設計の問題点



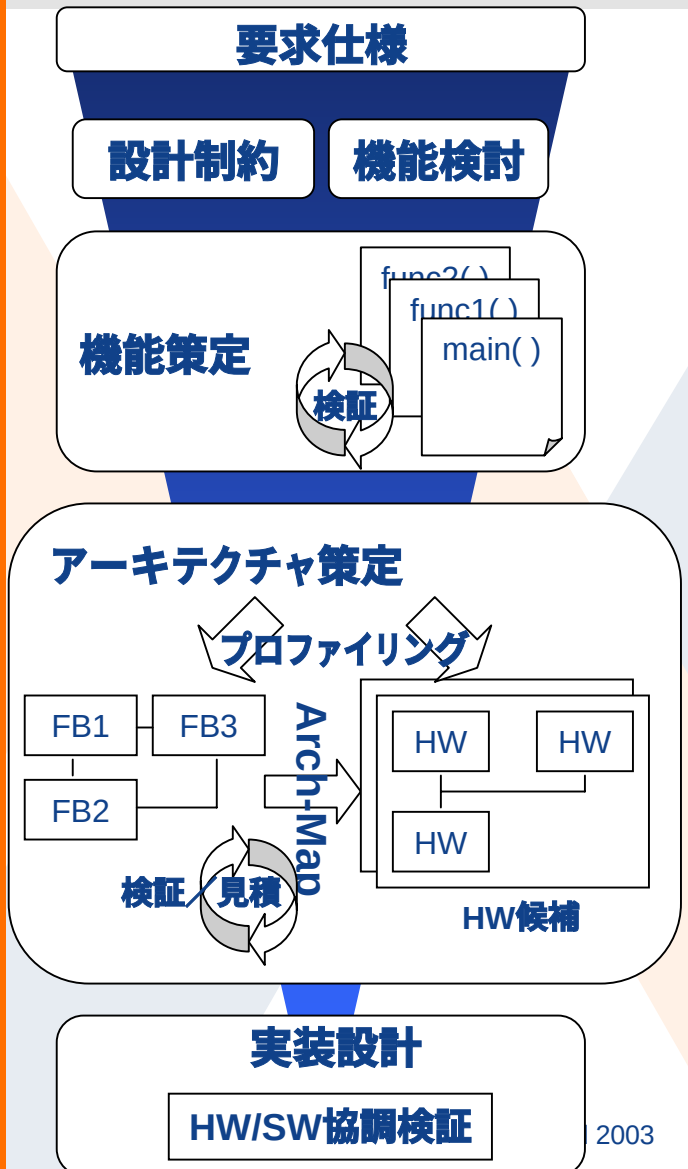
- 増大する要求仕様
- 相次ぐ設計仕様の変更
  - 本当に必要な仕様は何か？
  - 変更可能なSW（FW）の方が便利では？
- 要求仕様⇔実装設計間の巨大な壁
  - 仕様があいまい
  - 仕様の誤解釈が起こりやすい
- HW設計のメリットを訴えて行くには
  - システムレベル設計で対策

システム設計の対策は？



- 仕様設計から実装設計までシームレスに接続
- 各フェイズで検証し問題点を洗い出し
  - 曖昧点/甘い見積もりの洗い出し
  - 最適構造の模索
  - 再設計を削減
- プログラム言語を使用して設計の初期段階から検証
  - 機能策定
  - アーキテクチャ策定
  - 実装設計

C言語ベースの設計の可能性

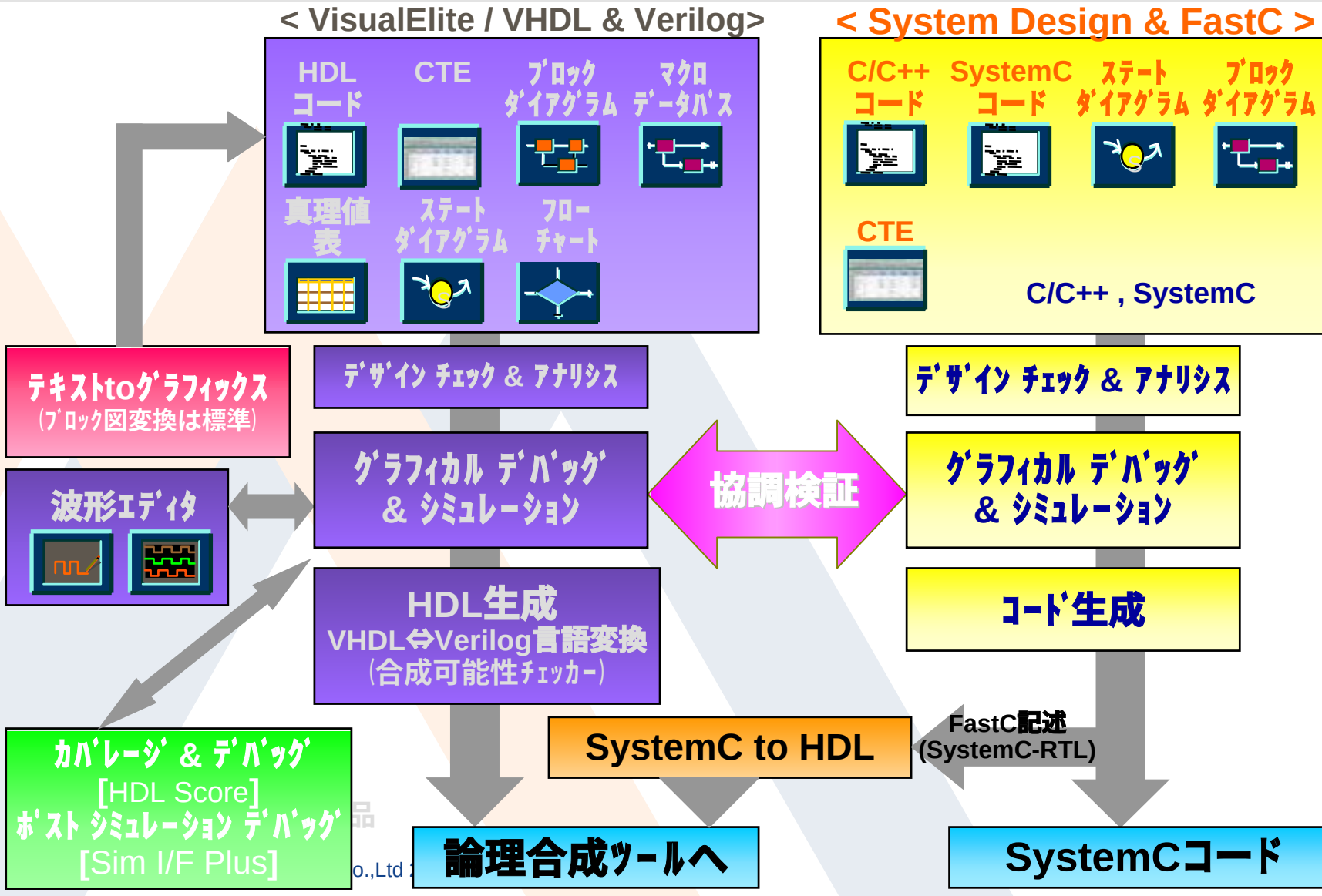


- **機能検討・機能策定**
  - C/C++等で「機能」を決める
- **アーキテクチャ策定**
  - 決定された「機能」を実現する「最適構造」を決める
  - C/C++系なら改良が楽
  - SystemC Ver2.0以上で検証・見積もり可能
- **実装設計**
  - 「タイミングを含めた詳細構造」を決める
  - 実装設計は論理合成前提
  - VisualEliteならSysteC(RTL)で実装設計

One Step Ahead

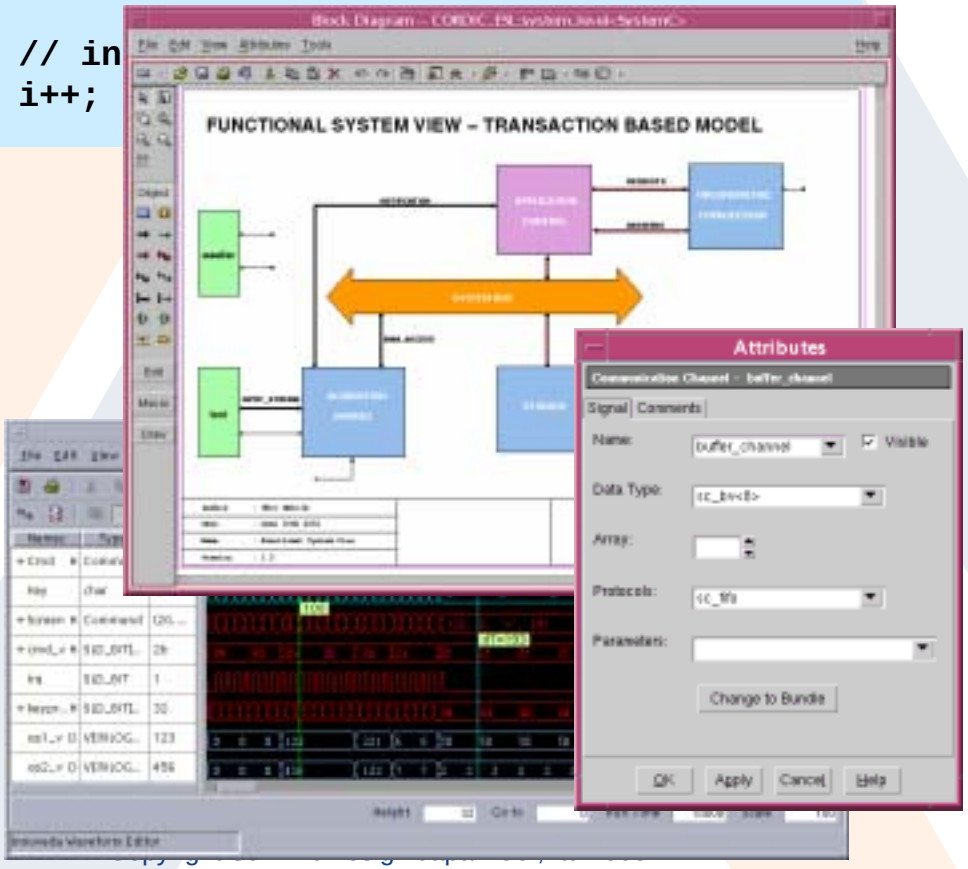
System Design

VisualElite with SystemDesign



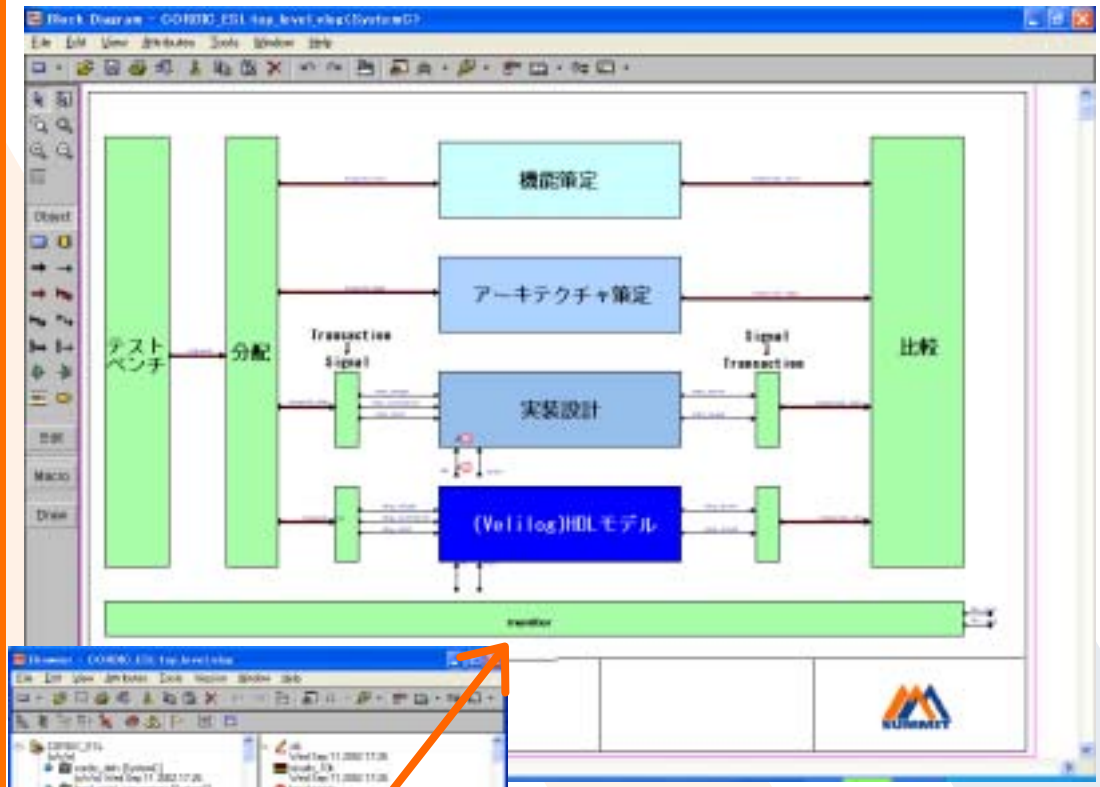
SystemC v2.0 サポート

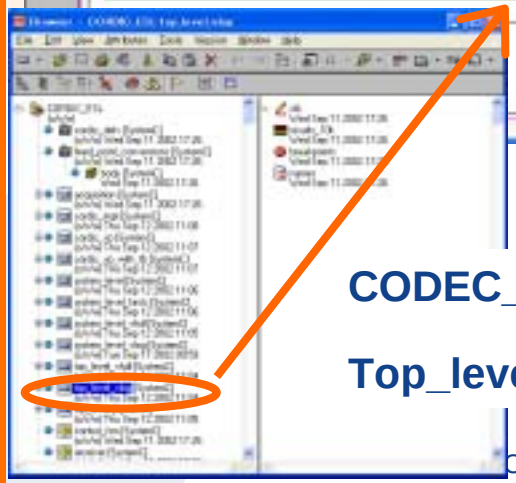
```
// sc_fifo blocking write method
buffer.write(i);

// wait for a "generic" amount of time
wait ( parameter, SC_NS );

// in
i++;
```



- SystemC 2.0サポート
  - OSCI・SystemC2.0.1 リファレンスシミュレータ内臓
- SystemC特有の表現を詳細に覚える必要なし
- 直感的で理解しやすい
- 豊富な表現をサポート
  - C/C++モデル
  - 各種抽象度SystemCモデル
  - SystemCのチャネル
  - SystemC-RTL(FastC)
  - ユーザー定義型/クラス

SystemDesignを使えば・・・





CODEC_ESL

Top_level_verilog

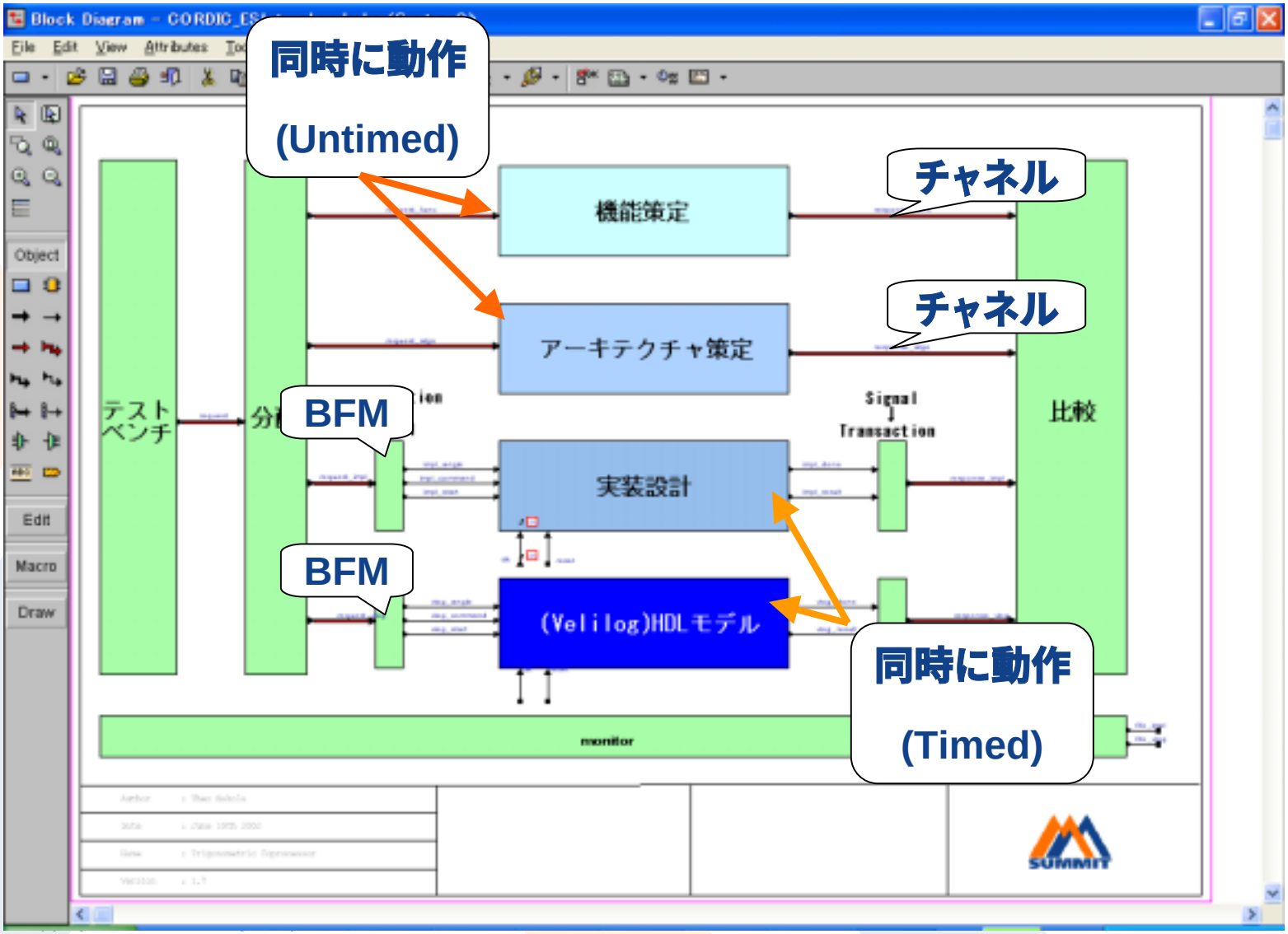
1つのエディタ上で

- 機能策定
- アルゴリズム策定
- 実装設計

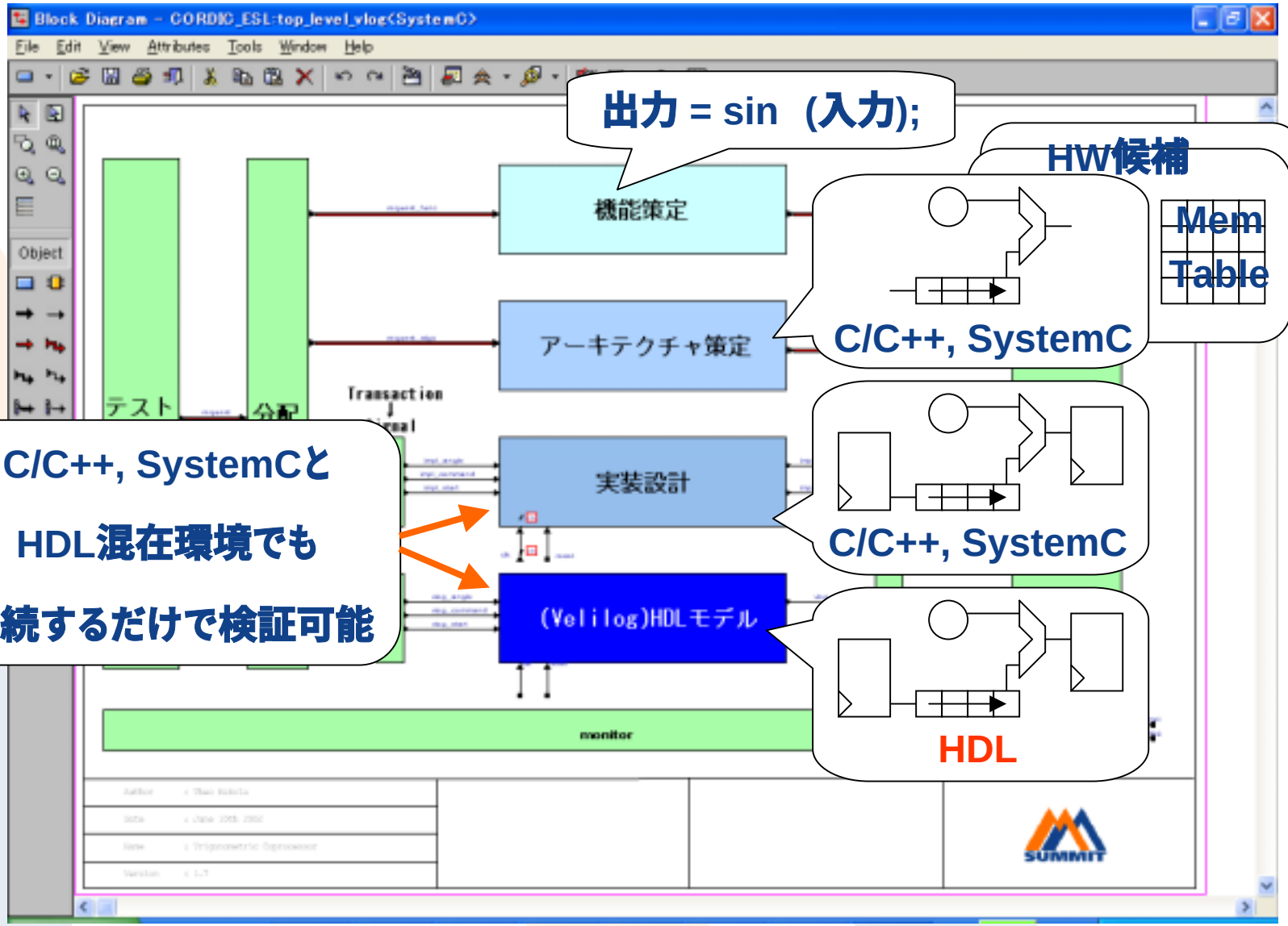
同一環境で機能策定から実装まで

- 設計の流れを
  シームレスに接続
- C/C++/SystemC/HDL混在
  検証可能

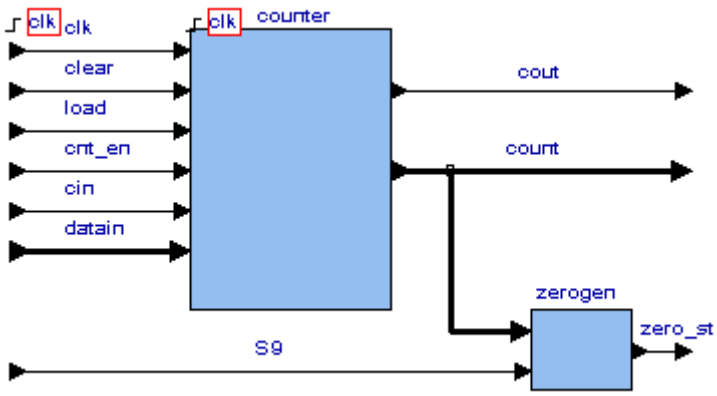
SystemDesignを使用した設計・1

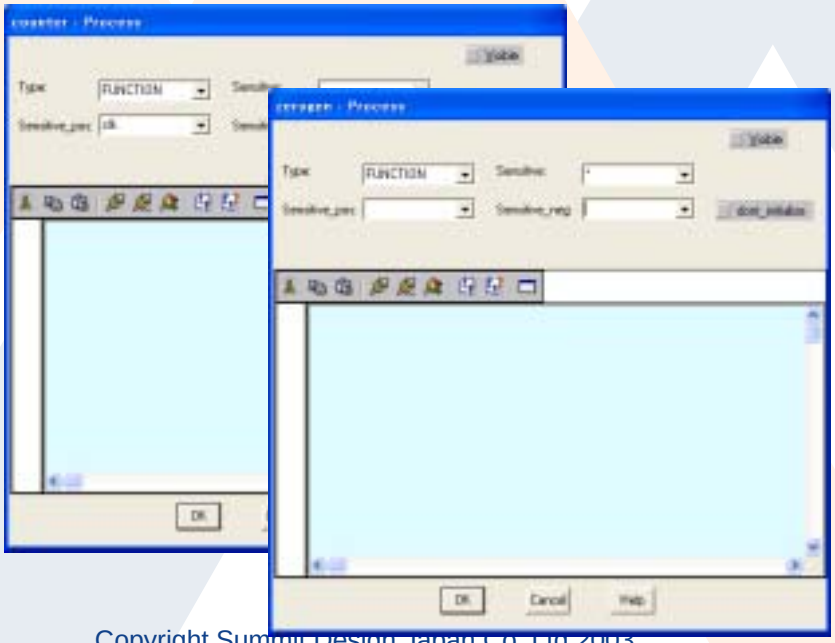


SystemDesignを使用した設計・2



グラフィカルツールのメリット

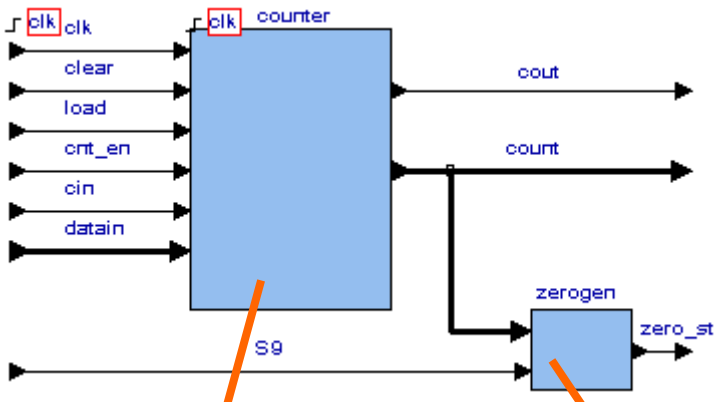


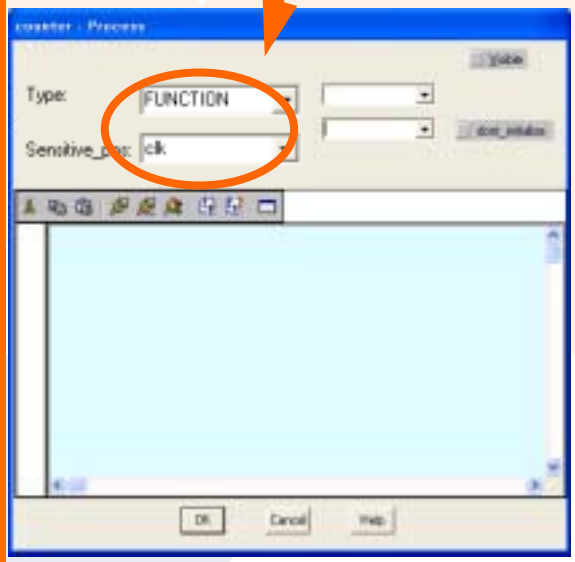


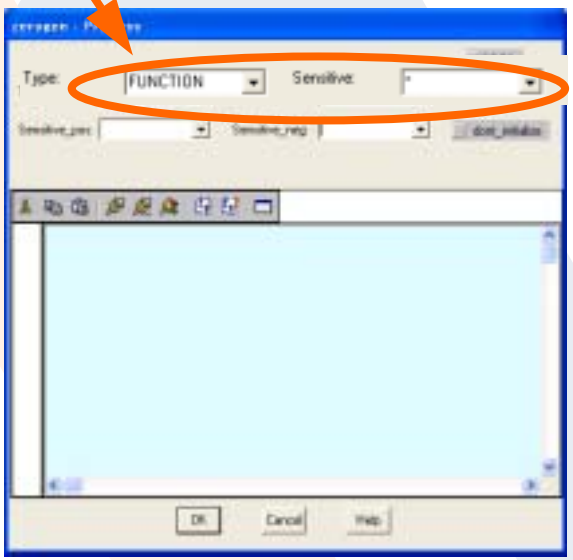
- **複雑な構文を覚える必要なし**

```
SC_MODULE (dwlib1__D_counter_sc) {
  sc_in_clk clk;
  sc_in<bool> clear;
  sc_in<bool> load;
  sc_in<bool> cnt_en;
  sc_in<bool> cin;
  sc_in<sc_uint<4> > datain;
  sc_in<bool> zero_stin;
  sc_out<bool> cout;
  sc_out<bool> zero_st;
  sc_out<sc_uint<4> > count;
    :
    :
  void counter();
  void zerogen();
    :
    :
  SC_CTOR(dwlib1__D_counter_sc){
    SC_METHOD(counter);
      sensitive_pos << clk;
    SC_METHOD(zerogen);
      sensitive << zero_stin << count;
  }
    :
    :
};
```

グラフィカルツールのメリット

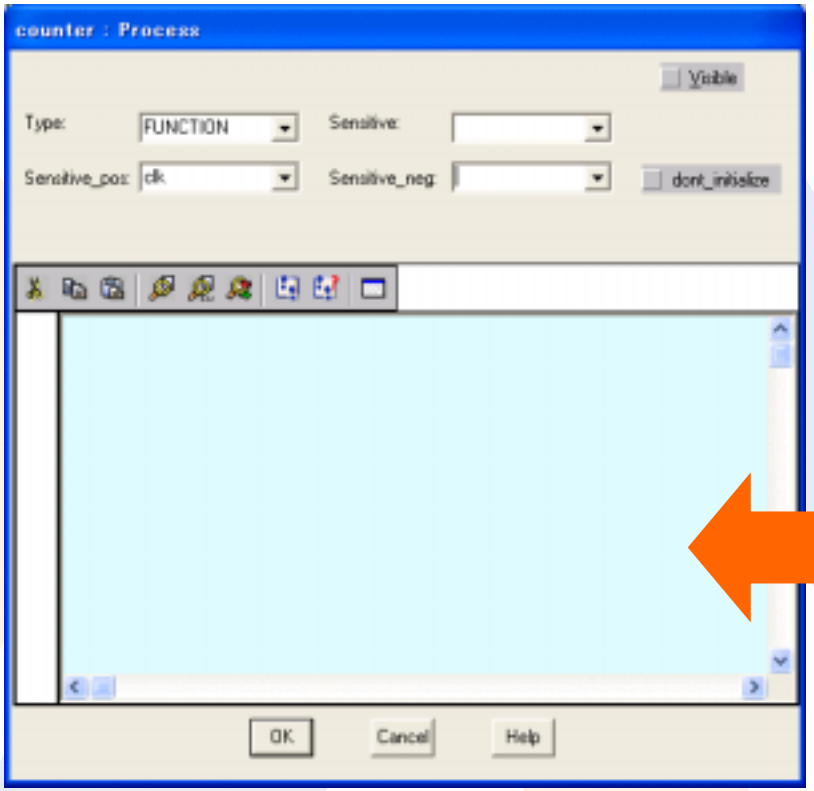






- センシティビティ設定
  - 動的センシティビティ
    - a = b + c;
wait(a,sc_ns);
    - プロセス動作が動作条件でダイナミックに変化
    - SystemC 2.0以上でサポート
  - 静的センシティビティ
    - プロセス自体に定義
    - sensitive_pos(立上り)
    - sensitive_neg(立下り)
    - sensitibe
    - ＊にて全ての信号を一括宣言可能

グラフィカルツールのメリット



- カウンタの動作を記述

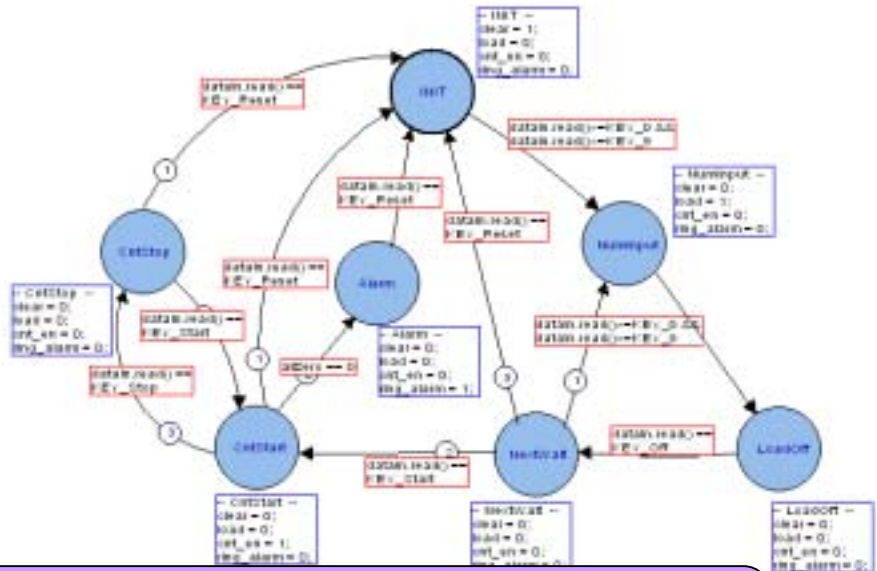
```
cout = 0;
if (clear.read() == 1) {
  count = 0;
} else if (load.read() == 1) {
  count = datain;
} else if (cnt_en.read() == 1 &&
          cin.read() == 1) {
  if (count.read() == 0) {
    if (zero_stin.read() == 1) {
      count = radix - 1;
      cout = 1;
    }
  } else {
     count = count.read() - 1;
  }
}
```

グラフィカルツールのメリット

- レビュー性の向上
- デバッグの効率化

```
void dwlib1__Control_fc :: state_case(){

    clear_int = 0;
    cnt_en_int = 0;
    load_int = 0;
    ring_alarm_int = 0;

     switch (current_state.read()) {
      case INIT:  {
        {
          clear_int = 1;
          load_int = 0;
          cnt_en_int = 0;
          ring_alarm_int = 0;

          if (datain.read()>=KEY_1 &&
              datain.read()<=KEY_9) {
            next_state = NumInput;
          } else {
            next
          }
        }
      }
      break;

      case NumInput:  {
        {
```

VS



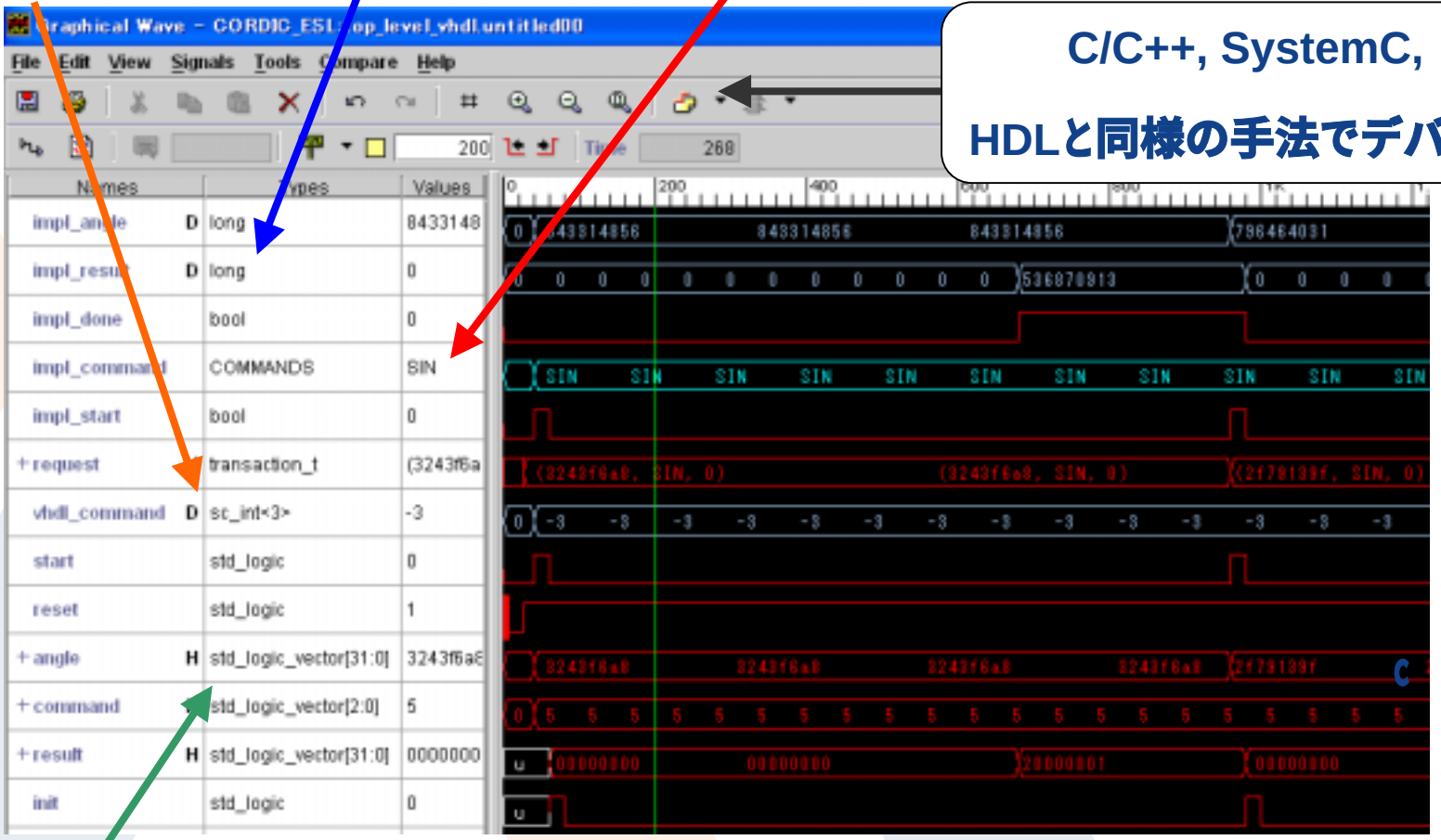
どちらがレビューしやすいですか？

C/C++, SystemC, HDLを同一表示

SystemCの型

C/C++の型

ユーザ定義ラベルで値表示



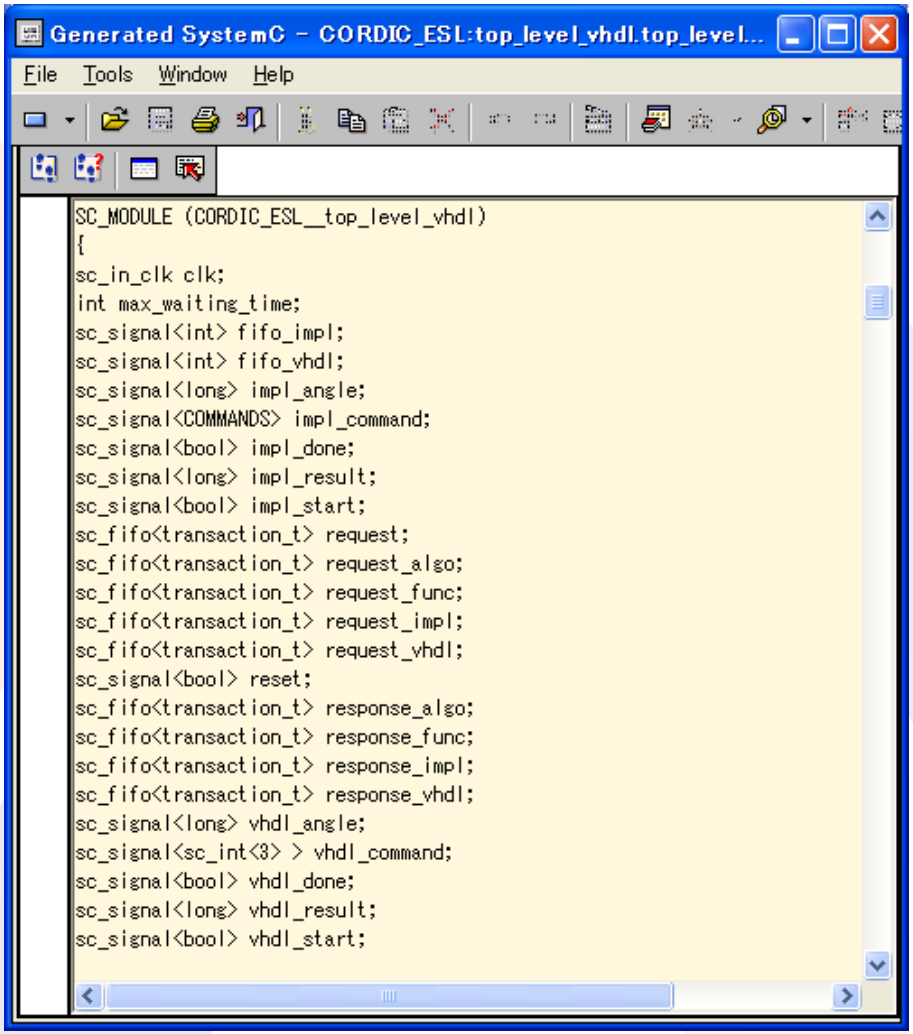
C/C++, SystemC,

HDLと同様の手法でデバッグ

VHDL/Verilogの型

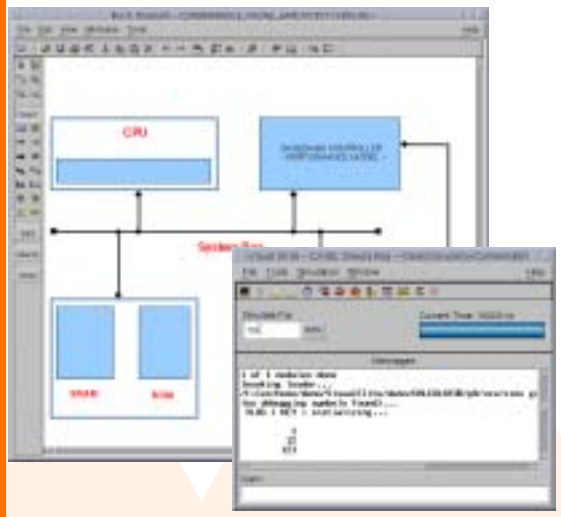
C/C++, SystemC, HDL を同じ画面上でデバック

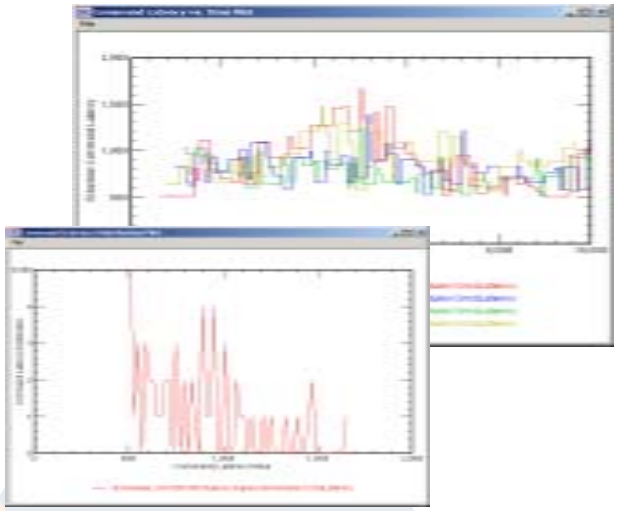
SystemCツールとのI/F

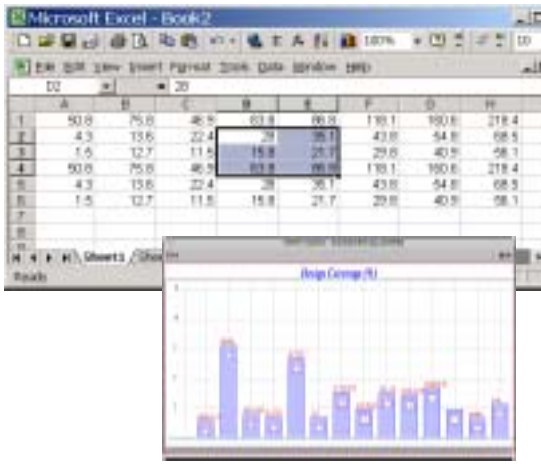


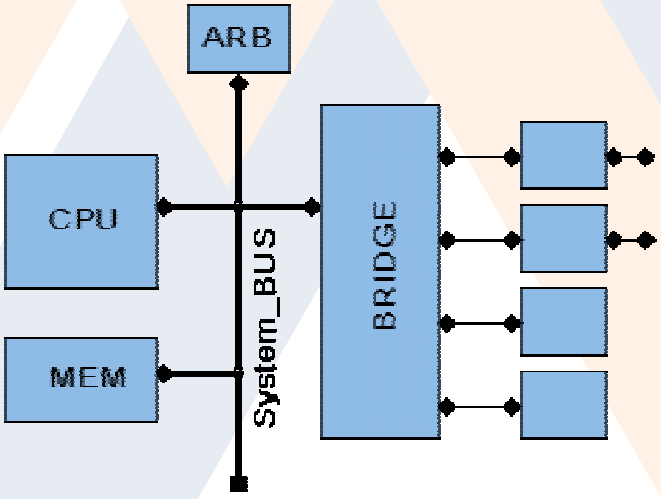
- SystemCツールへ
  - SystemCコード生成
- SystemCツールから
  - SystemCコード登録/利用
- C/C++モデル
  - C/C++モデルも再利用可能
  - 他のC/C++系ツールとの接続

パフォーマンス解析ツール
System Architect









- 標準プラットフォームの＆トークンベース・パフォーマンス解析ツール
- 定量解析
  - スループット/レイテンシ/バンド幅
- 詳細調査
  - バス衝突/バッファ量の最適化

Matlab インテグレーション

Mathworks

.M ファイルの作成･評価

The MathWorks
MATLAB 6

Matlab
.M ファイル

C/C++
ソース
コード

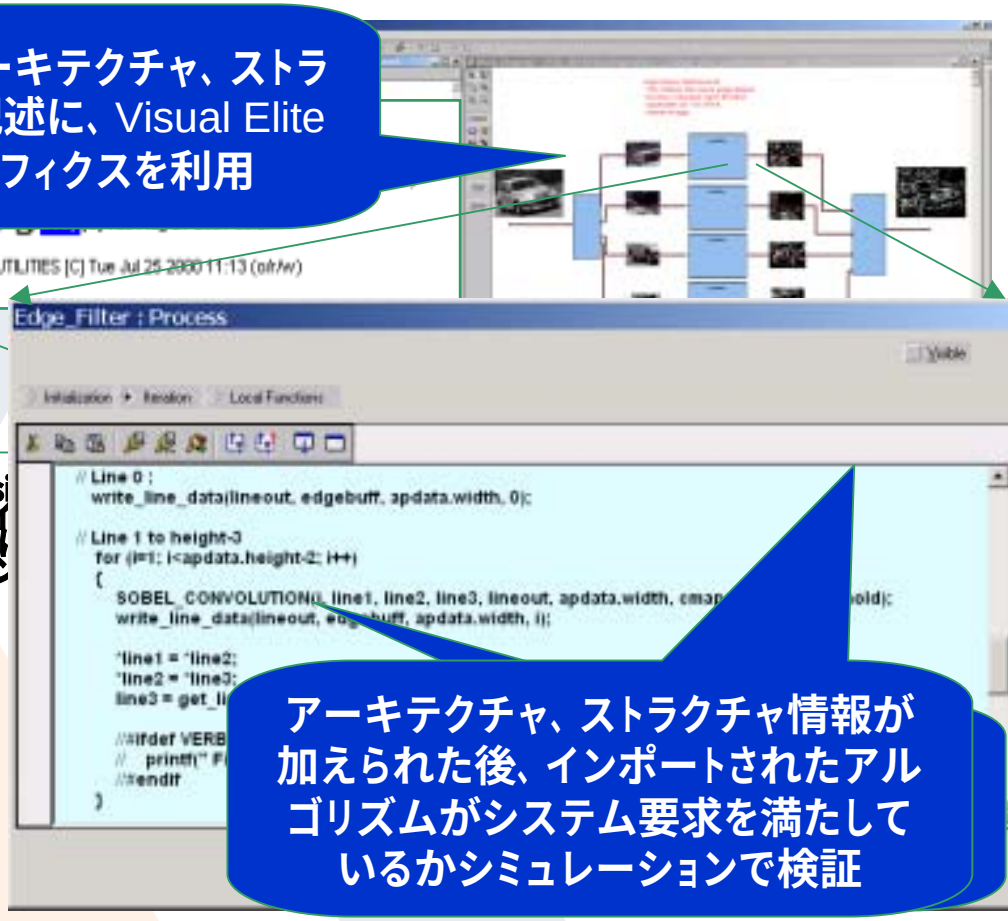
The MathWorks
MATLAB COMPILER

.M ファイルから ANSI C/C++
への変換

Summit

Visual Elite – System Design

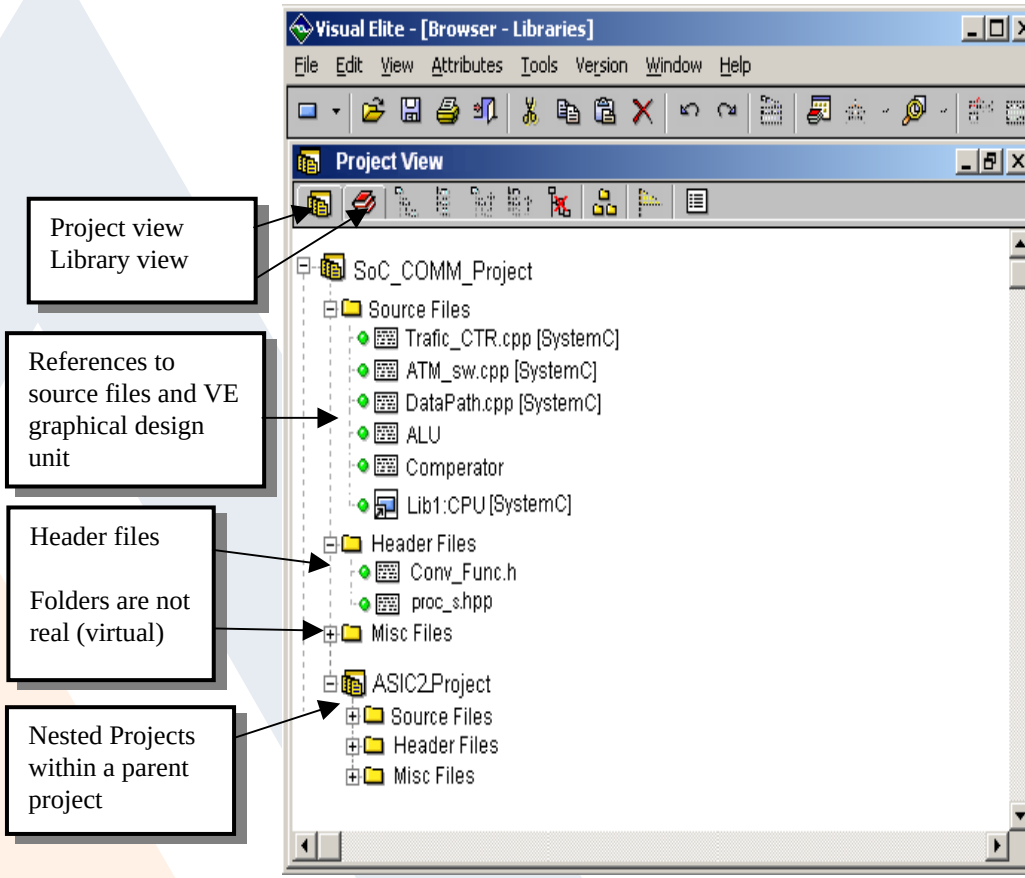
設計のアーキテクチャ、ストラクチャの記述に、Visual Elite のグラフィクスを利用



アーキテクチャ、ストラクチャ情報が加えられた後、インポートされたアルゴリズムがシステム要求を満たしているかシミュレーションで検証

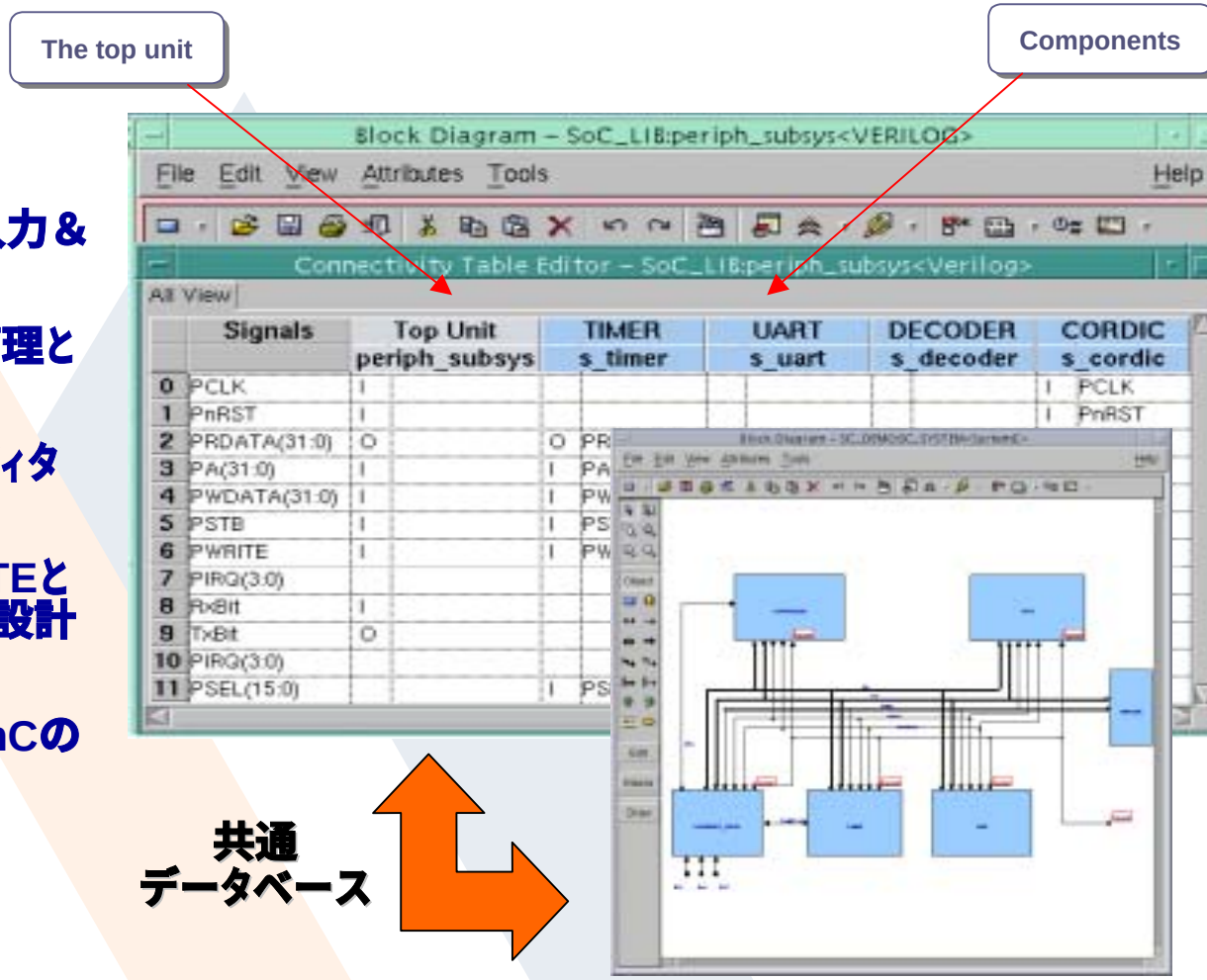
SystemC Text Centric

- あらゆるSystemC / HDLテキスト環境にプラグイン
- グラフィクスとテキスト混在デザインへのアプローチに適合
- テキスト・ユーザーのためのVisualによる検証手段とマネージメント機能
- 設計資産を新しい設計プロジェクトにシームレスに統合
- C, SystemC, VHDL, Verilog すべてのファイル及び構造をサポート



Connectivity Table Editor

- 表形式による構造記述入力＆編集エディタ
- 複雑なデザイン構造の管理と効果的な入力が可能
- ブロックダイアグラムエディタから自動的に生成可能
- ブロックダイアグラムとCTEと双方の同期を取りながら設計可能
- Verilog, VHDL, SystemCのサポート



One Step Ahead

FastC

FastCの基本的な考え方

- SystemC RTL サブセットを利用
- RTL の SystemC モデルを高速検証
- 何故？
  - 検証速度：機能検証 ＞ アーキテクチャ検証 ＞ 実装検証
    - 機能検証 ↓ (HDL-RTL比1000倍以上)
    - 実装検証 ↓ (通常方法ならHDL-RTLとほぼ同じ)
  - 現在の合成技術では、RTLからの合成が一番確実
  - 現状技術では論理合成は、避けて通れない

- Fast-C で RTL でも検証速度を上げて補完
  (HDL-RTL比10～300倍)

Cでも論理を表現できます・でも・・・

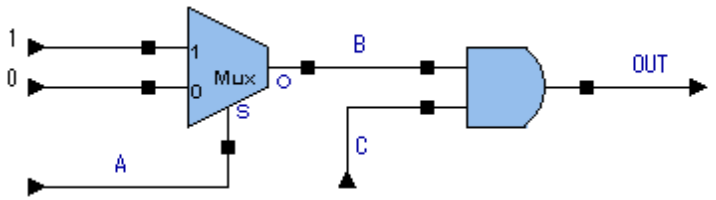
- C/C++でもHDL-RTLと同等な論理を表現できます
- C/C++は逐次処理。しかしHDL-RTLは同時処理を表現できます。

C/C++の場合：

```
func Z(){
  if(A){
    B = 1; // Logic
  }else{...
}
func Y(){
  OUT = B & C;
}
```

Verilog の場合：

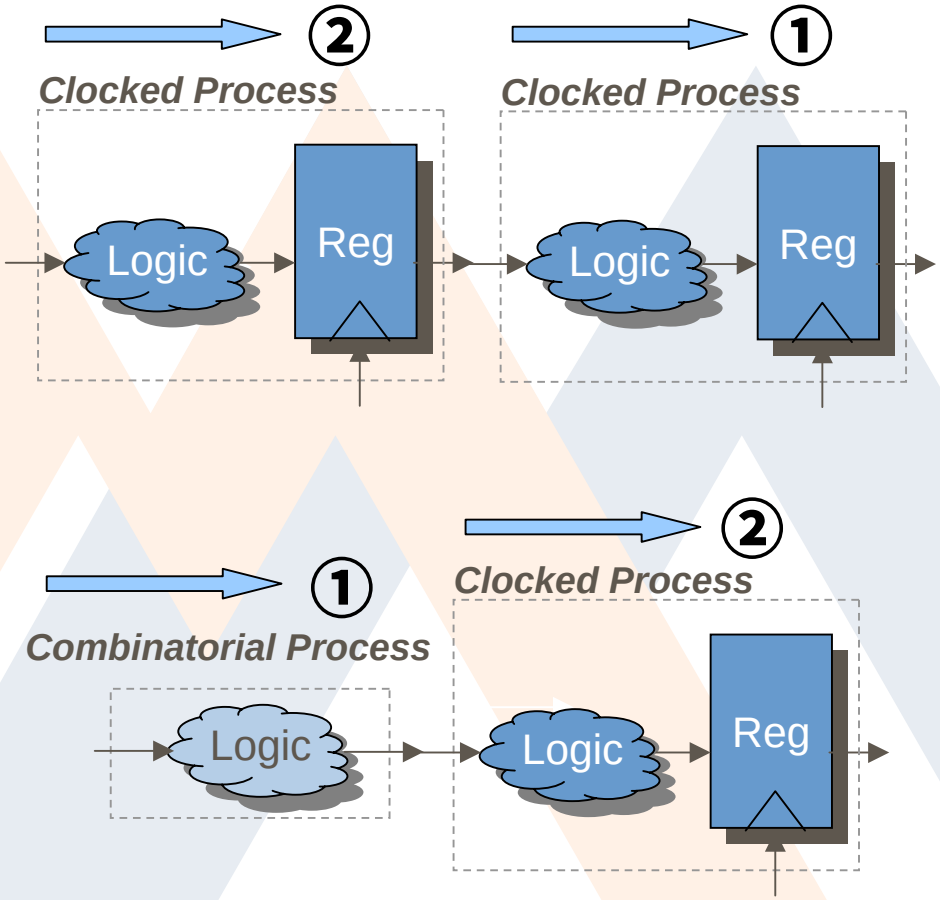
```
always @(a) begin
  if(A){
    B <= 1; // Logic
  }else{...
end
always @(B or C) begin
  OUT <= B & C;
end
```



C/C++では関数の処理順番を間違うと正しい結果が出ません

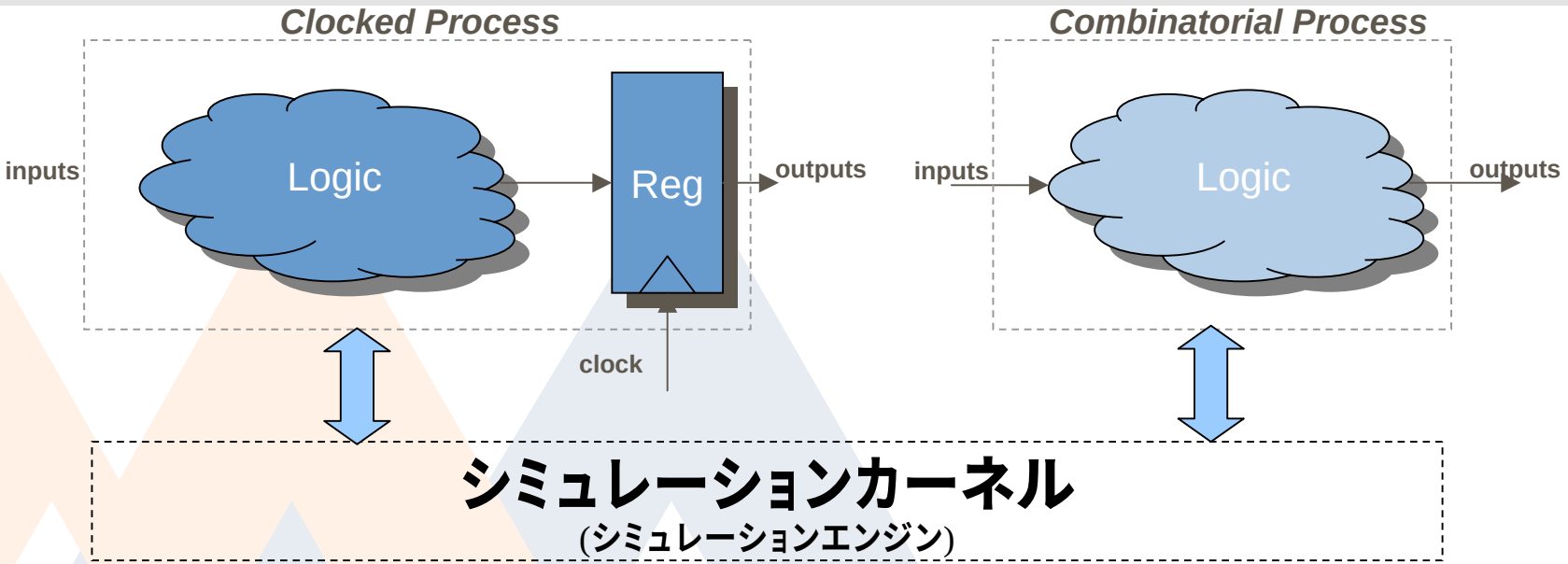
HDLでは各論理が独立動作するので処理順番を気にする必要はありません

Static Scheduling Technology (FastC)



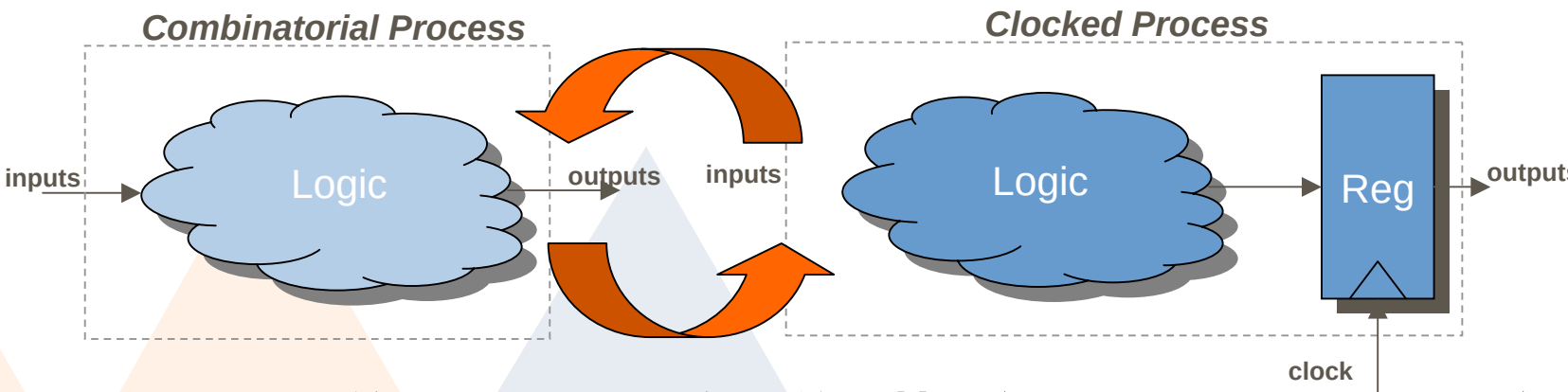
- 評価順番を自動認識→並べ替え
- 逐次処理のCソースを内部生成
  - 実行イメージ作成
  - Simulatorカーネル依存しないシミュレーション
  - Simulation中にダイナミックに処理順番が変わらず高速

通常のシミュレータの構成



- シミュレーションカーネルが並列動作を補償
- メリット／デメリット
  - 処理順番など記述制約が少ない
  - さまざまな抽象度の混在
  - Tri-Stateなど、さまざまなHW表現可能
  - 処理順番がダイナミックに変化するので遅い

FastCはStaticScheduleで高速Simulation

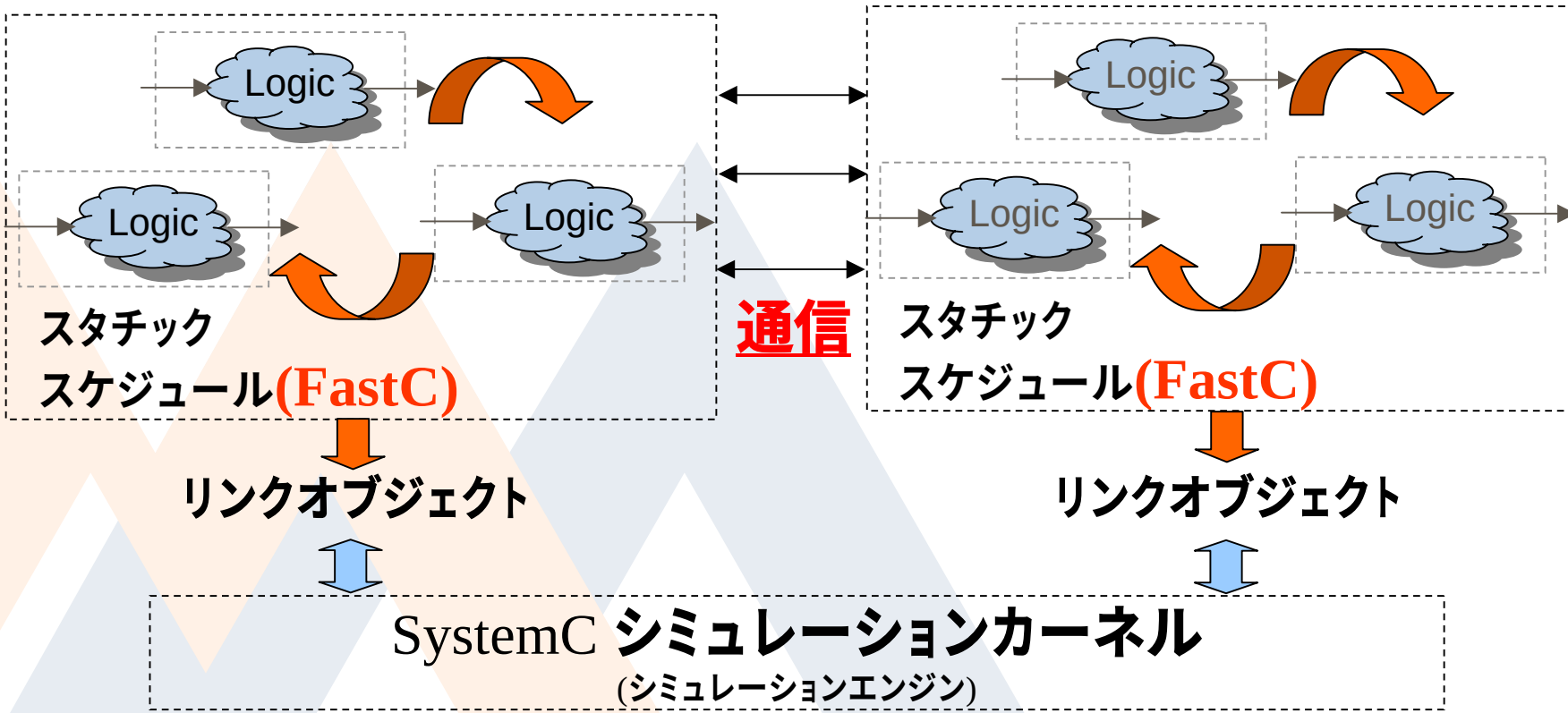


①シミュレーション前に評価順番を自動並べ替え（スタチック・スケジュール）

②*.exeを作ってシミュレーション

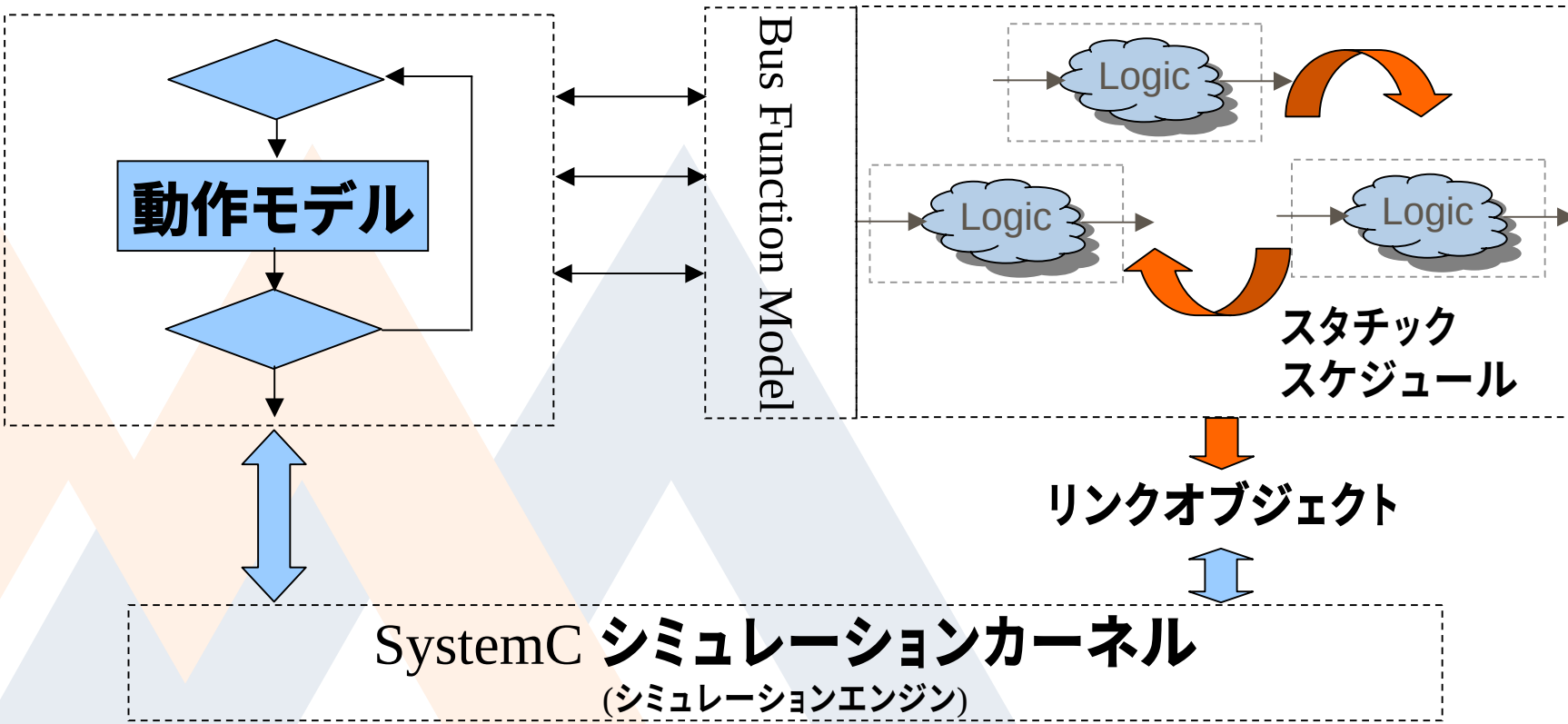
- シミュレーション前に評価順番決定→自動並べ替え
- メリット/デメリット
  - 速い（x10～x300　HDL-RTL比）
  - 前処理は評価順番の検出と関数並べ替えのみ
  - さまざまな抽象度、さまざまなHW表現不可

種々のHW表現も SystemC-Simと協調で解決！



- スタチックスケジュールで解決不可能箇所をSystemC シミュレータで実行
  - RTL部の殆どをFastCテクノロジで処理可能

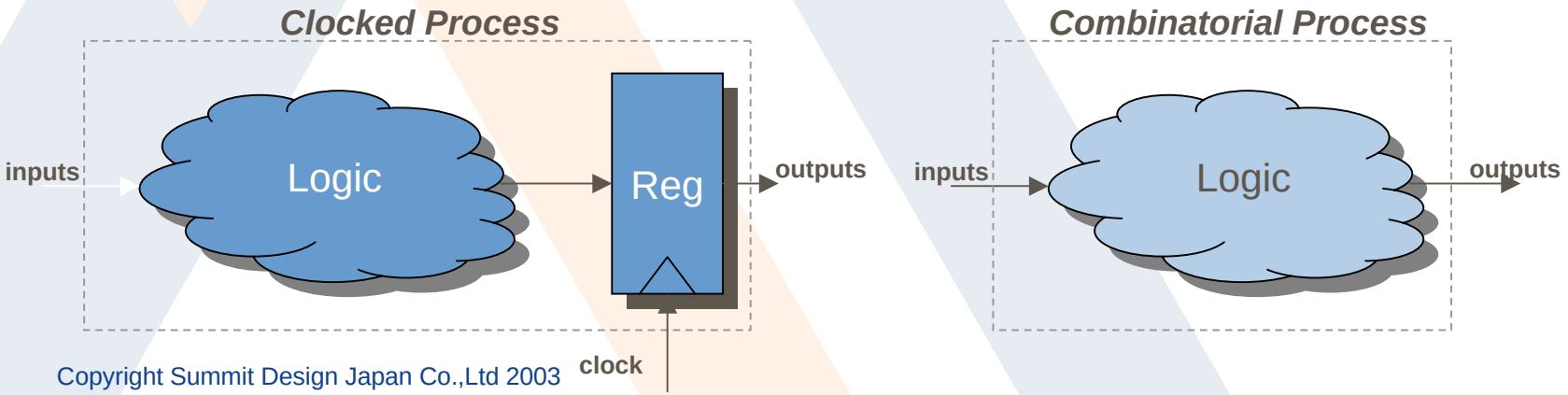
抽象度が異なるモデルでも FastC+SystemC-Simで解決！



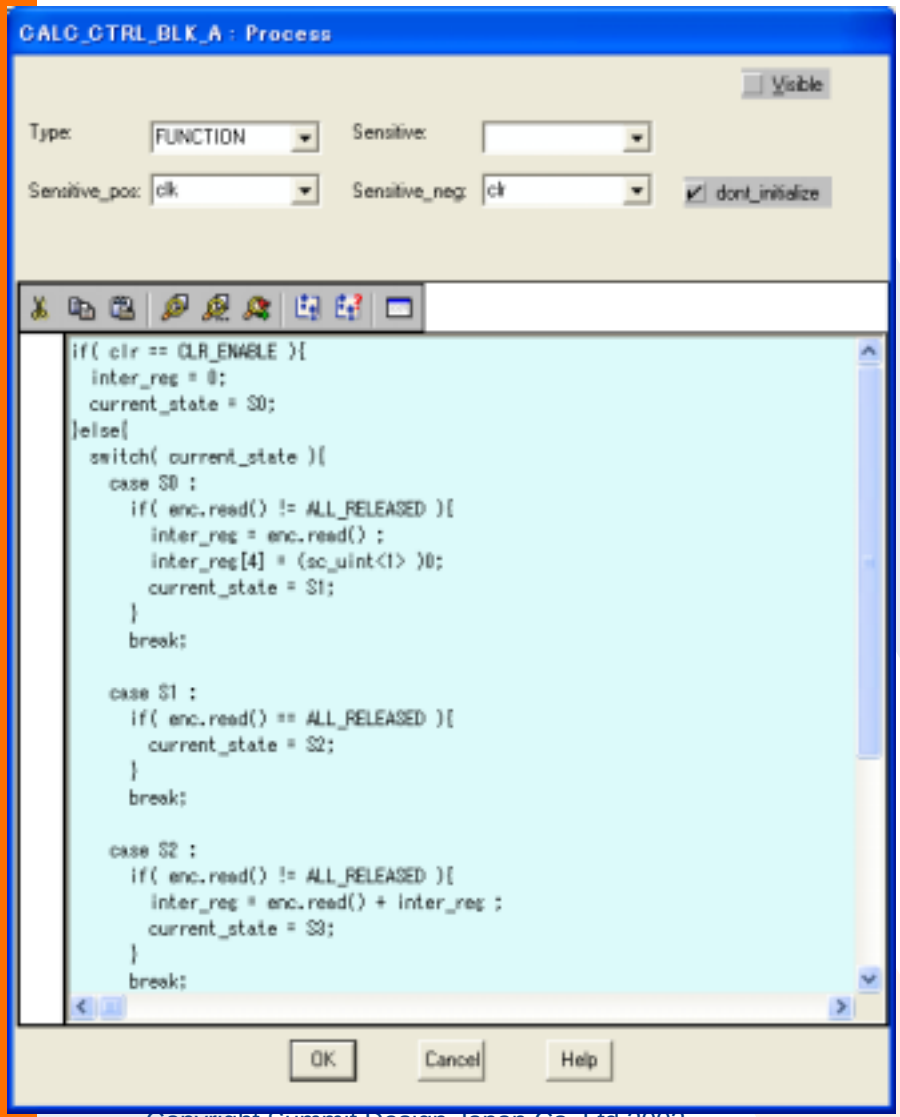
- VisualElite は SystemCシミュレータ内蔵
- 抽象度が異なるモデルでも検証可能
- RTL抽象度のC/C++,SystemCはFastCで高速シミュレーション

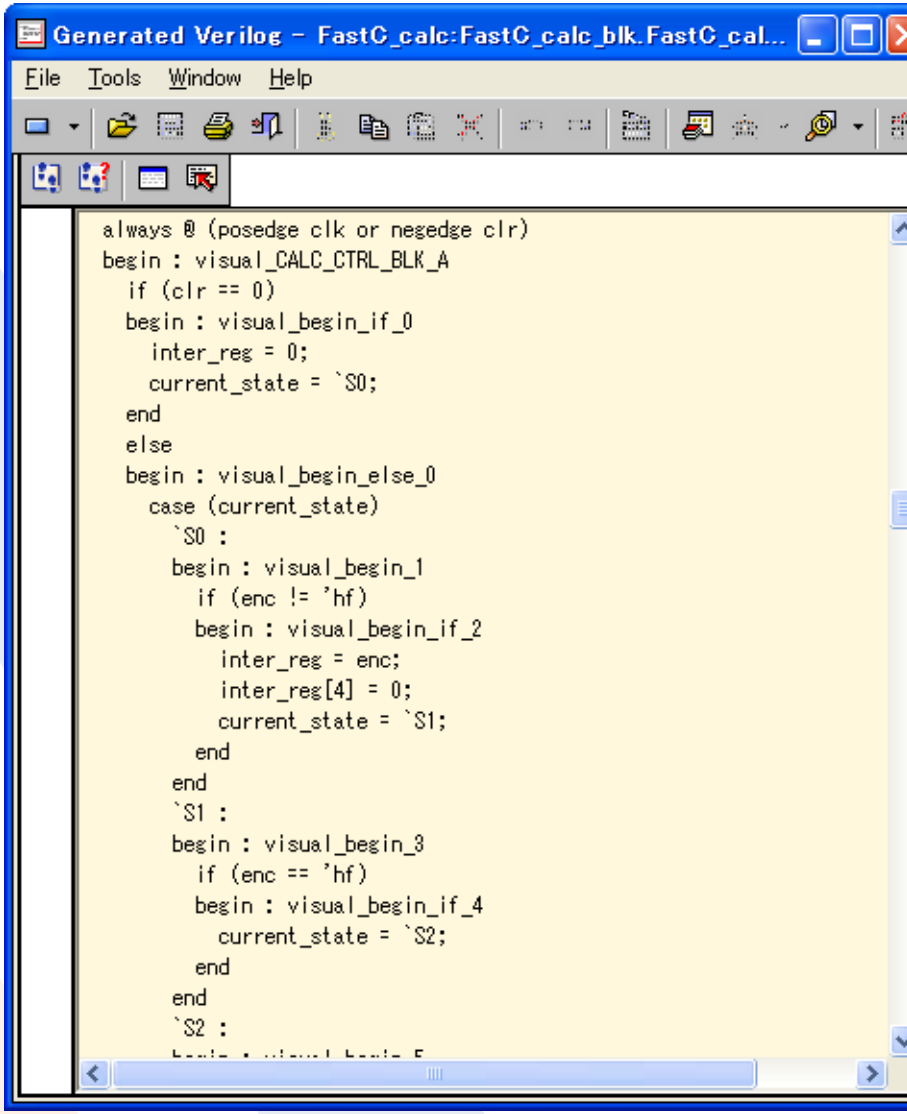
FastCコーディングスタイル

- SystemCサブセットを使用
  - SystemC-RTL
- FastC は2種のプロセスをサポート:
  - Clocked プロセス (順序回路)
  - Combinatorial プロセス (組合せ回路)



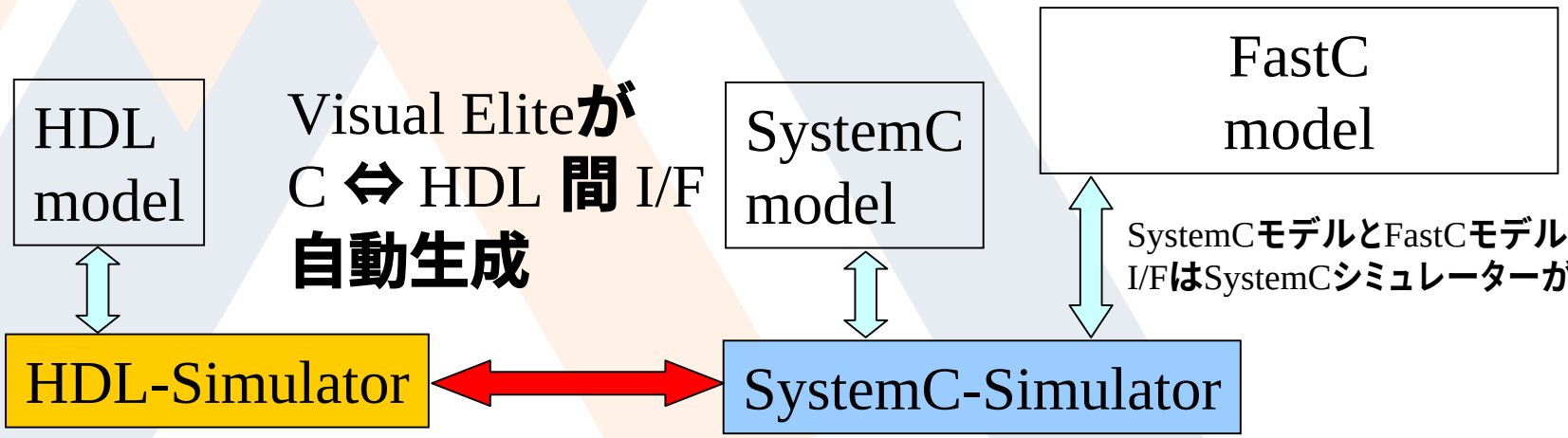
SystemC to HDL出力例



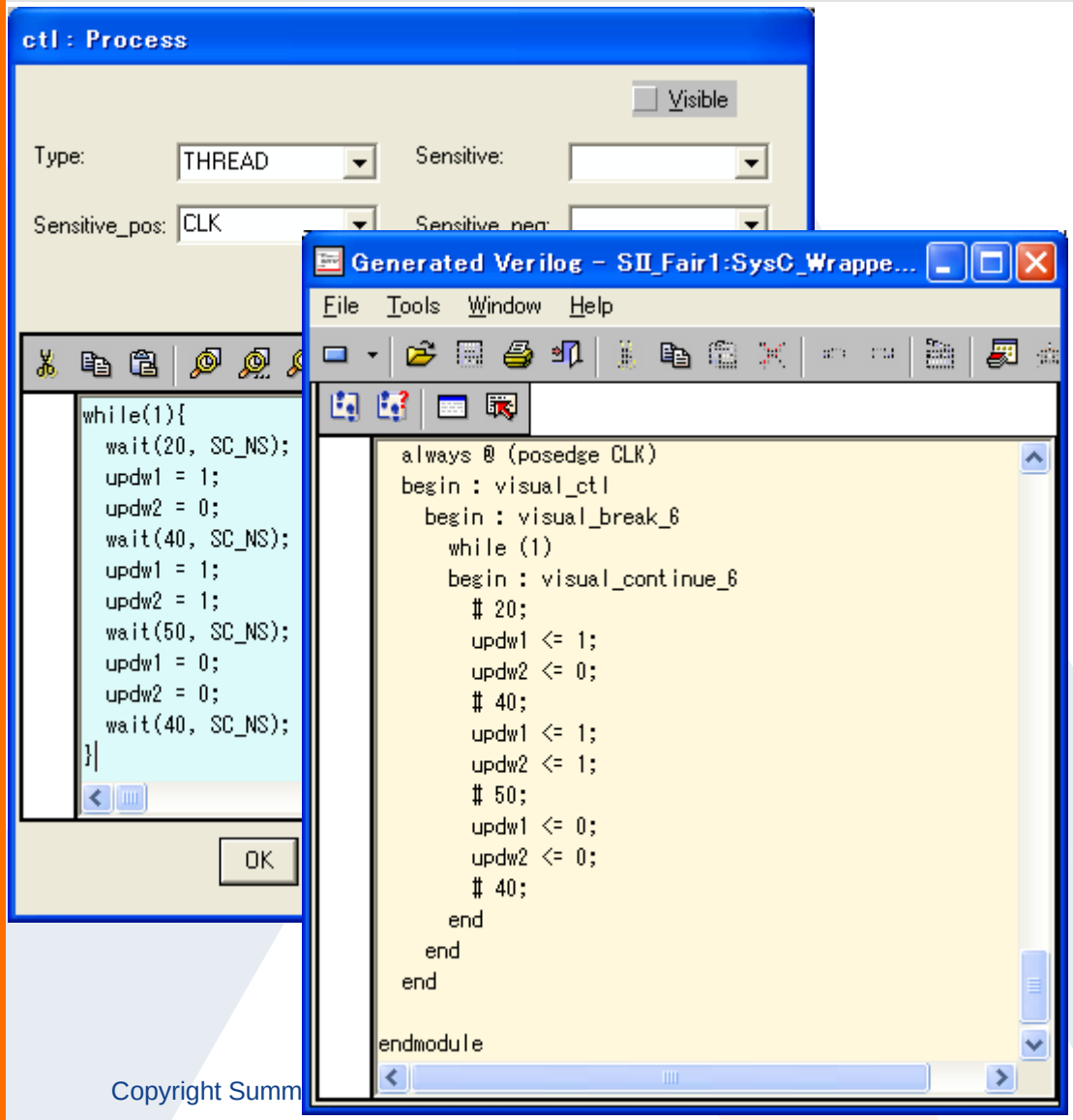


SystemC, FastC, HDL協調検証

- Visual Elite は SystemC, FastC, HDLを協調検証可能：
- HDL シミュレーションと協調すると・・・
  - SystemDesign は協調検証のためのインターフェイス(PLI, FLIなど)を自動生成
  - 自動生成した PLI は SystemDesign モデルの入出力を受け持つ



SystemCテストベンチ → HDL変換サポート



- SystemC-TBをHDLに変換
- SystemC⇔HDL間の等価なシミュレーションを実現

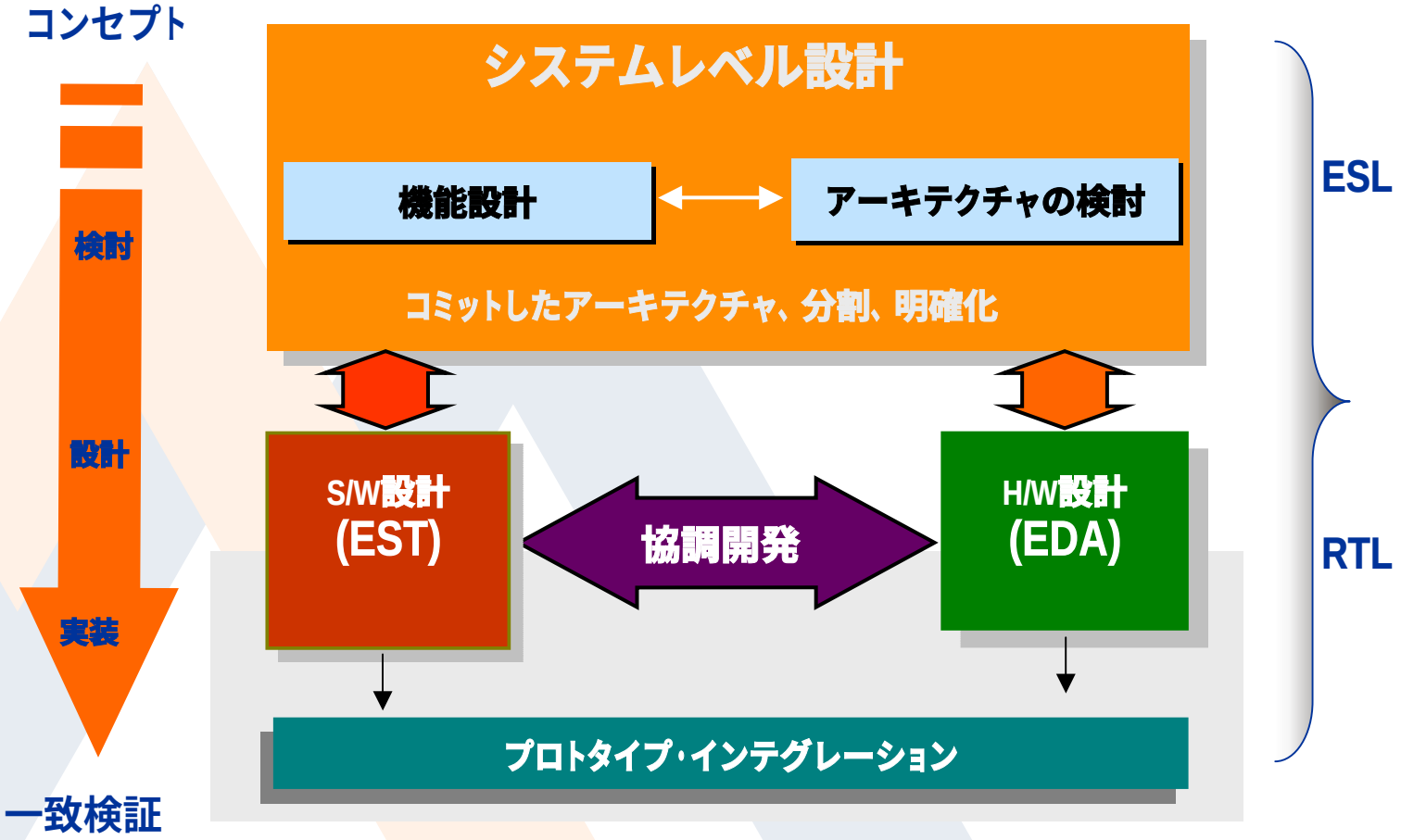
One Step Ahead

NEW!

Visual Elite ESC

組込みシステム
SystemCベースHW/SW協調検証環境

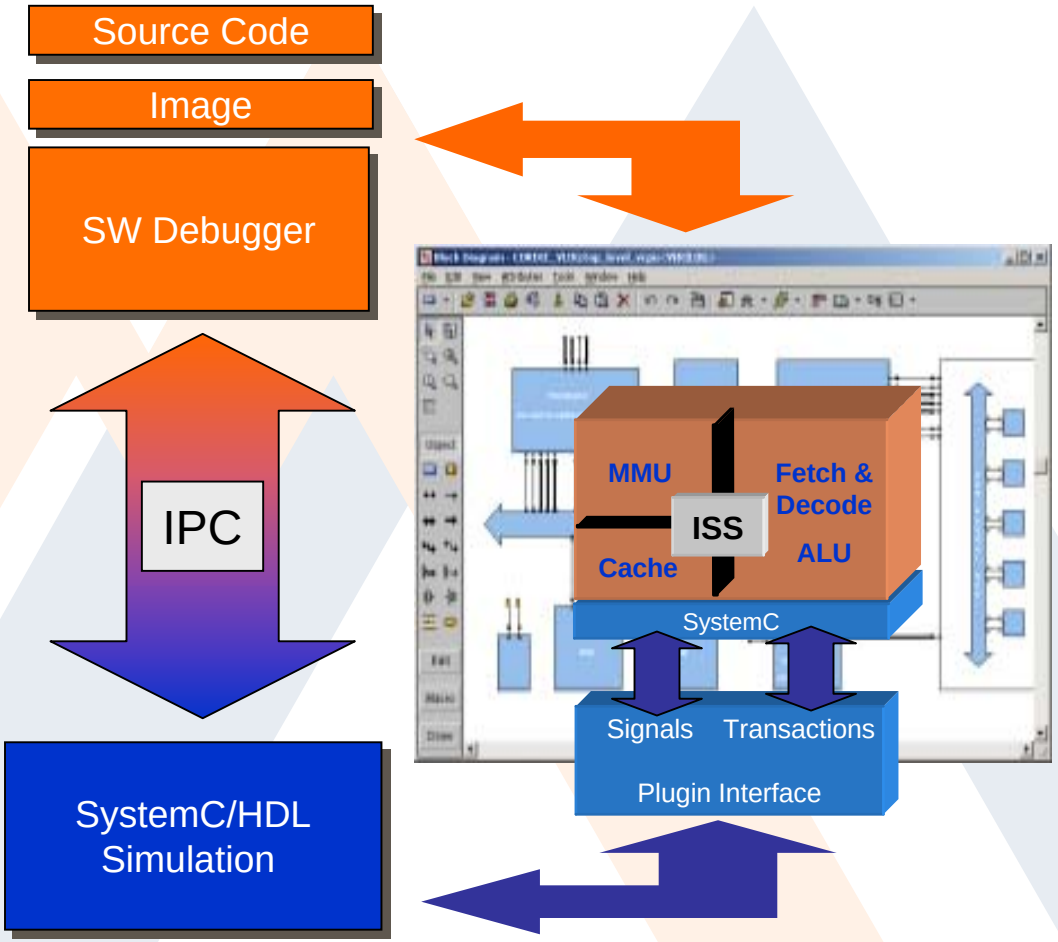
Electronic System Level 設計手法



SystemC Landscape

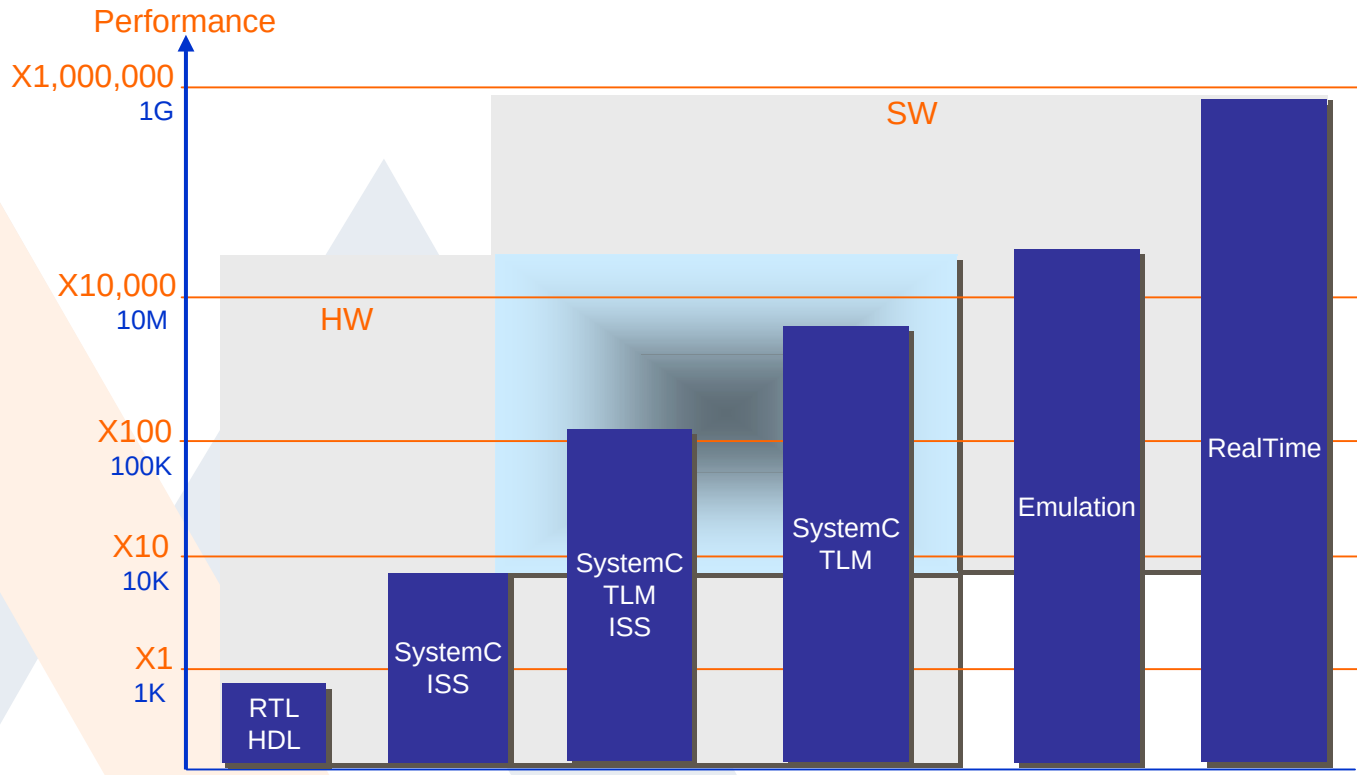
- SystemCは、システム・モデリングのすべての範囲にわたり、S/Wプロトタイピング、H/Wインプリメンテーション、統一されたHW/SW共通言語を提供する
- SystemCによる標準化により、ツール・ベンダーはオートメーション・メカニズムを開発
- SystemCは、過去10数年蓄積されたインフラ、C/C++ライブラリへの膨大な投資、ツール群を活用できる

Visual Elite ESC



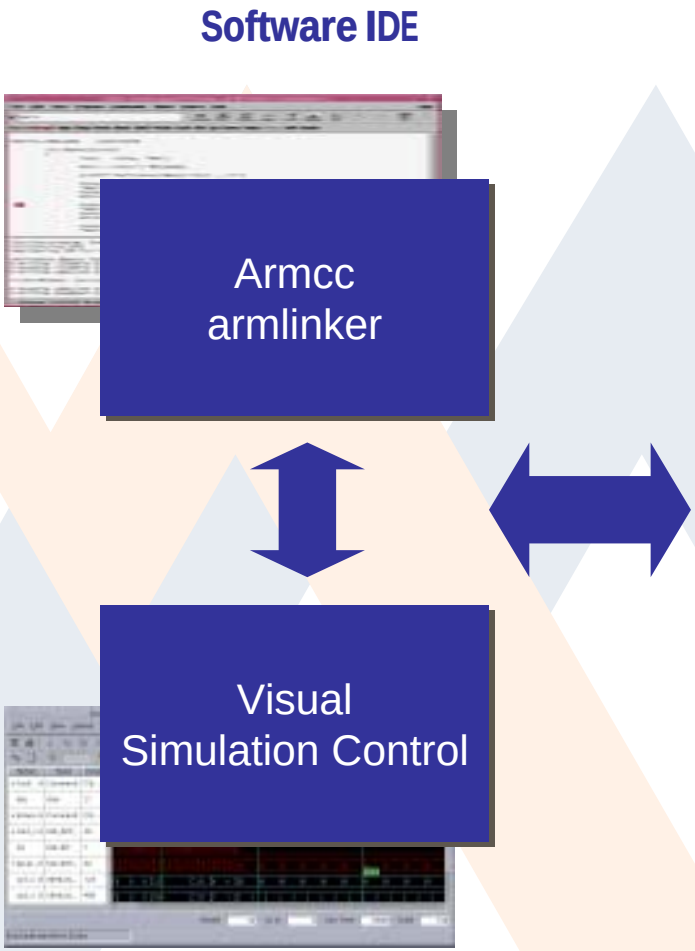
- ターゲット・プロセッサの為のSystemCによるハイレベル・機能＆アーキテクチャ設計環境
  - Xilinx V2P, ARM, Motorola
- 強力且つインタラクティブなHW/SW協調プラットホーム
- トランザクション・レベルでの通信
- バーチャル・プロトタイピング

検証プラットホームとパフォーマンス

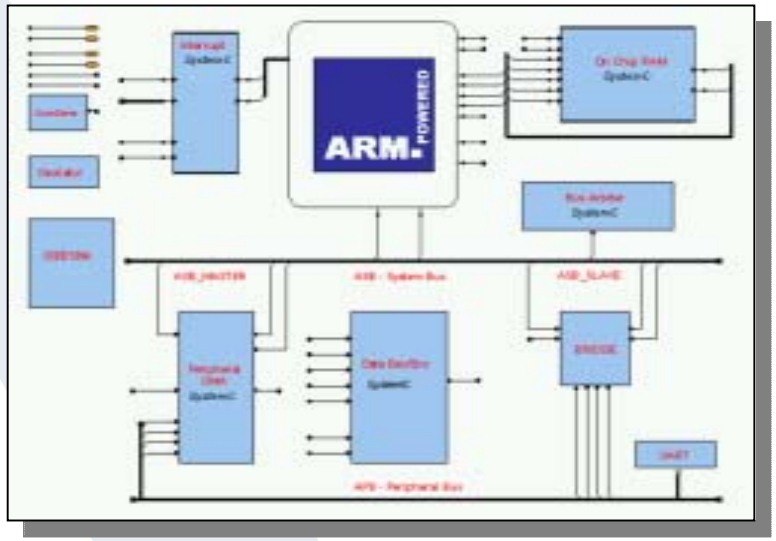


- SystemCとTLMにより、最も有効なHW/SW検証レベルを提供
- トランザクション・レベル
  - 現実的なパフォーマンスで機能検証を行うための精度
  - モデリングはRTLに近く、HDLへのインプリメンテーションが容易

Visual Elite ESC
ARM CCM プラットホーム



- ARM CCM Family
- AXD
- RVD in Q3/03 (multi-processors)

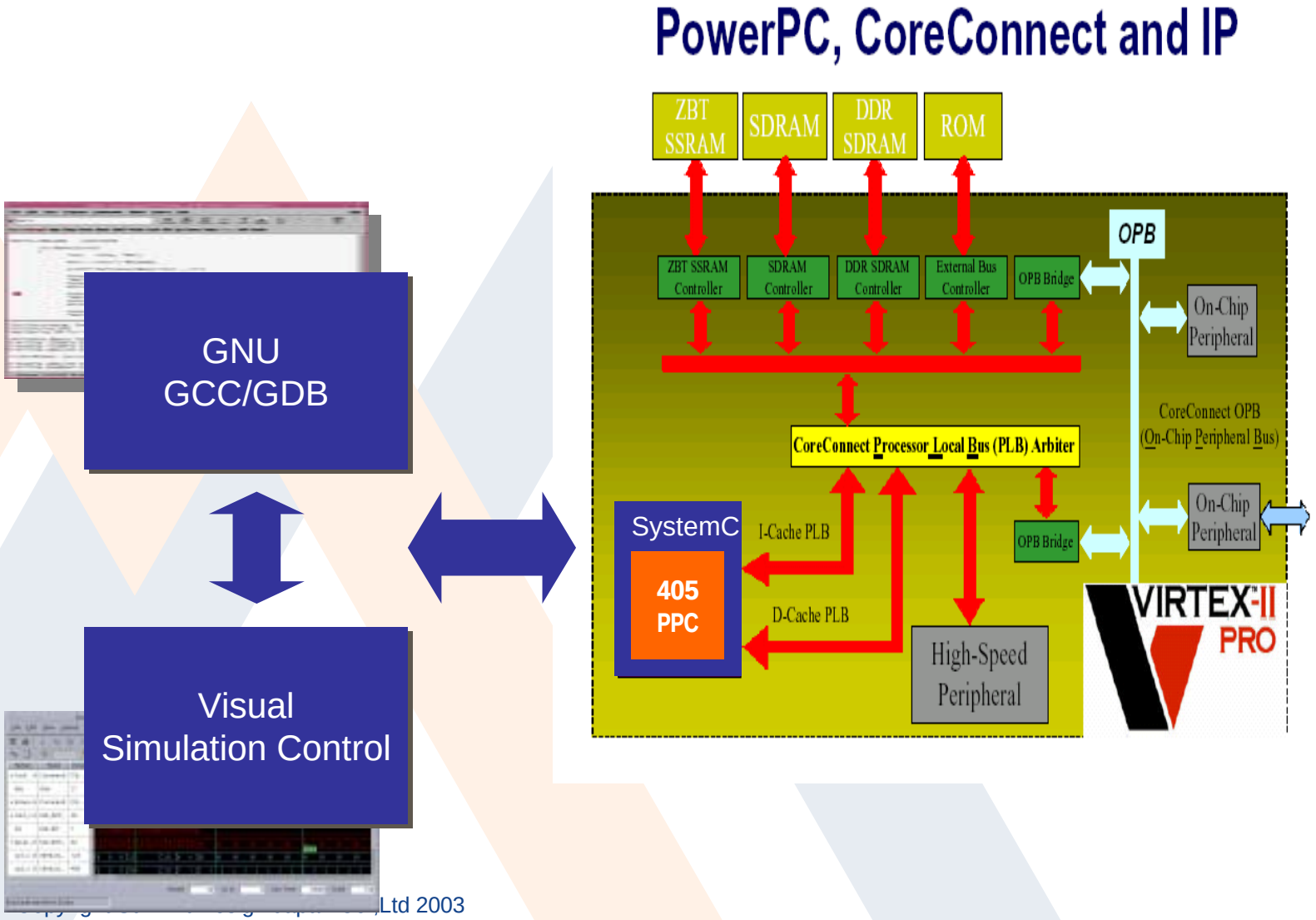


Visual Elite ESC
ARM CCM ソリューション

- CCM - Cycle Callable ISS Models (Cycle Accurate) をARMより供給
  - シグナル・トランザクション・レベルでのSystemCインターフェイス
  - SystemCあるいはHDLのペリフェラルに接続
- ネイティブなプロセッサ・デバッグ (armcc, armlinker)
  - アセンブラあるいは、ソースコードでのデバッグ
  - タイムスタンプによるソース管理や逆アセンブル
- HWシミュレーションとSW事項が完全に同期
- SW開発・テストの為の組込プロセッサ及び、システムの仮想プロトタイプ・モデル

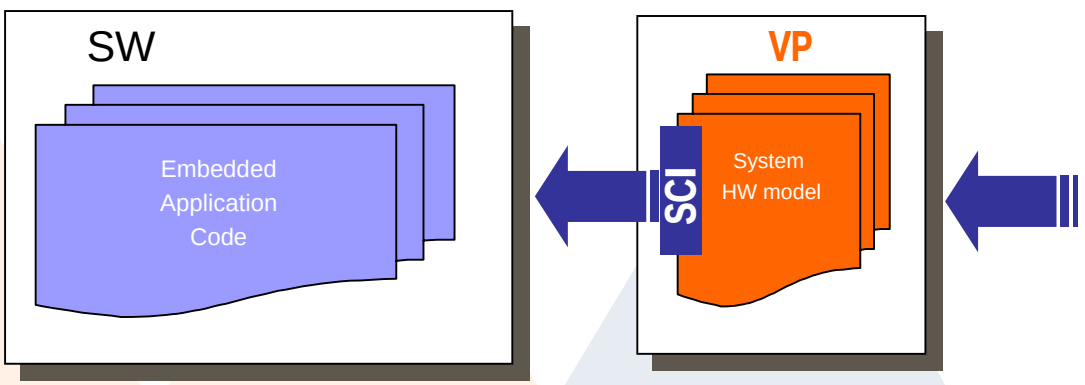
CCM Models
ARM9TDMI
ARM920T
ARM922T
ARM926EJ-S
ARM940T
ARM946E-S
ARM966E-S
ARM7TDMI
ARM7TMI-S
ARM720T

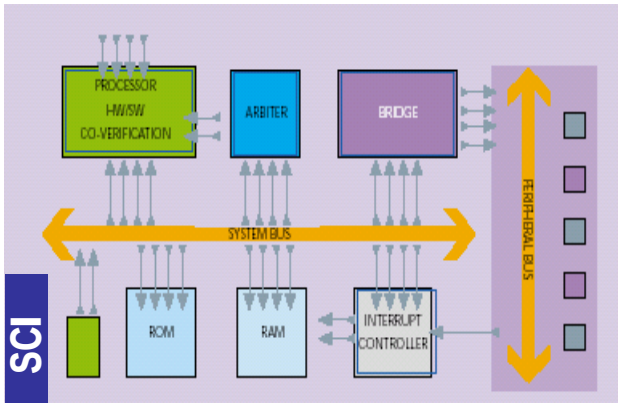
Visual Elite ESC
Virtex-II PRO プラットホーム

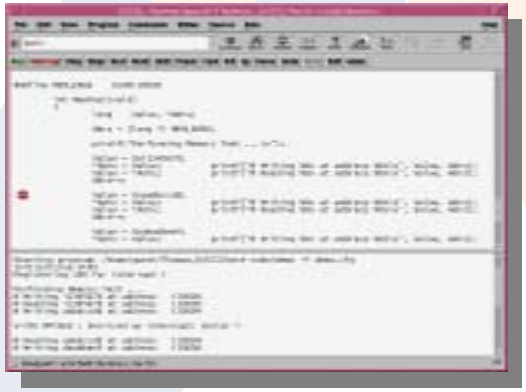


Virtual Prototype

C/SystemC based Virtual Prototype







Software IDE

- ネイティブなSW開発環境にリンクするHWの実行形式モデルを生成
- ISSの組込をサポート [ターゲット・モード]
- BFMのサポート [ホスト・モード]
  - SWインターフェイス
    - Read/Writeトランザクション
    - 割り込み
    - 手続きインターフェイス