



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

要求仕様

- ◆ 甘い見積もりによる再設計
- ◆ 仕様の誤解釈

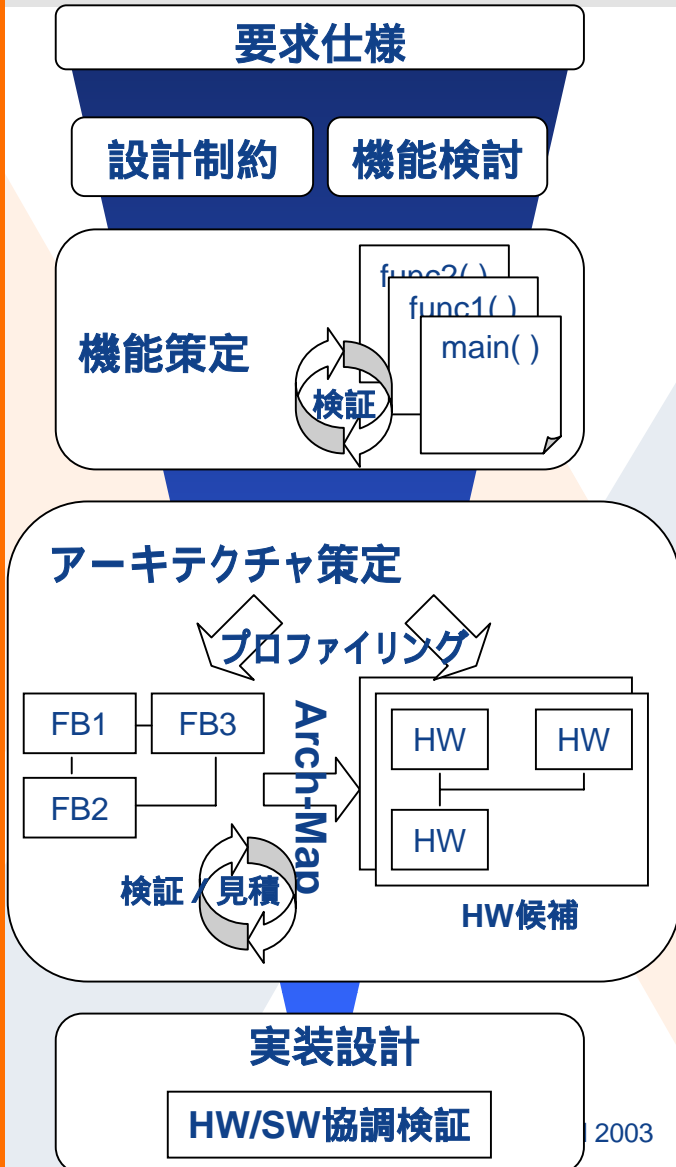


実装設計

- ◆ 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



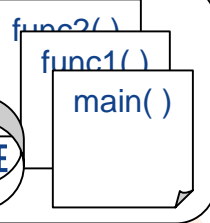
C言語ベースの設計の可能性

要求仕様

設計制約

機能検討

機能策定



◆ 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

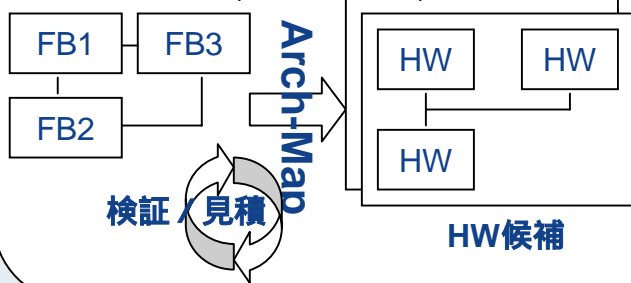
◆ 実装設計

- ◆ 「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SystemC(RTL)で実装設計

アーキテクチャ策定



HW候補

実装設計

HW/SW協調検証



One Step Ahead



System Design



VisualElite with SystemDesign

< VisualElite / VHDL & Verilog >



< System Design & FastC >



テキストtoグラフィックス
(ブロック図変換は標準)

デザインチェック & アナリシス

デザインチェック & アナリシス

波形エディタ

グラフィカルデバッグ
& シミュレーション

グラフィカルデバッグ
& シミュレーション

協調検証

HDL生成
VHDL Verilog言語変換
(合成可能性チェッカー)

コード生成

カバレッジ & デバッグ
[HDL Score]
ポストシミュレーションデバッグ
[Sim I/F Plus]

SystemC to HDL

FastC記述
(SystemC-RTL)

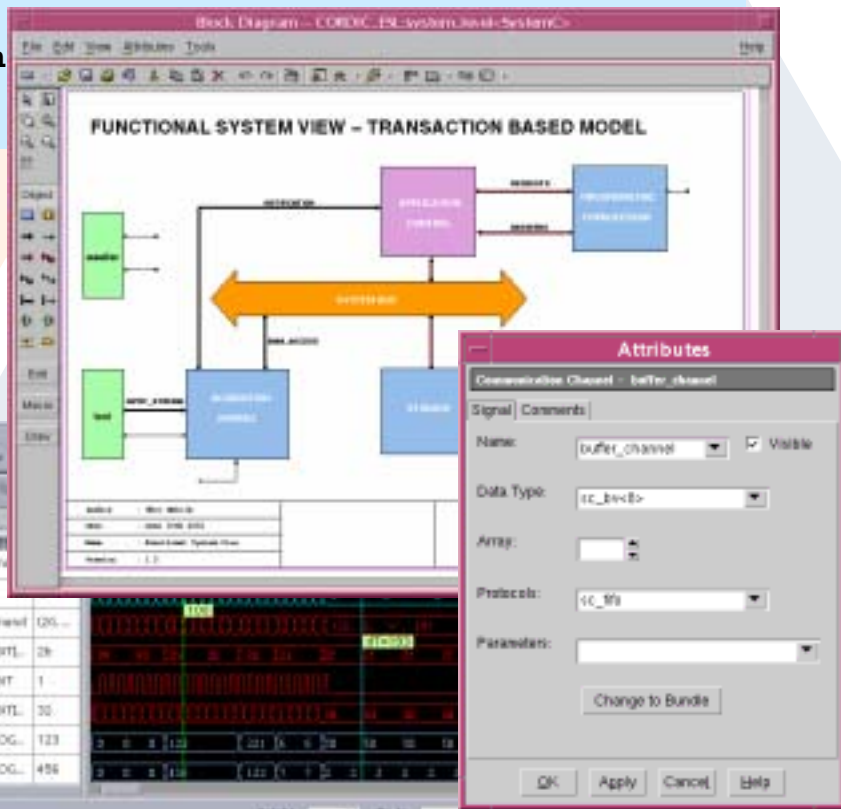
論理合成ツールへ

SystemCコード



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のチャンネル
 - ◆ SystemC-RTL(FastC)
 - ◆ ユーザー定義型/クラス



SystemDesignを使えば・・・

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

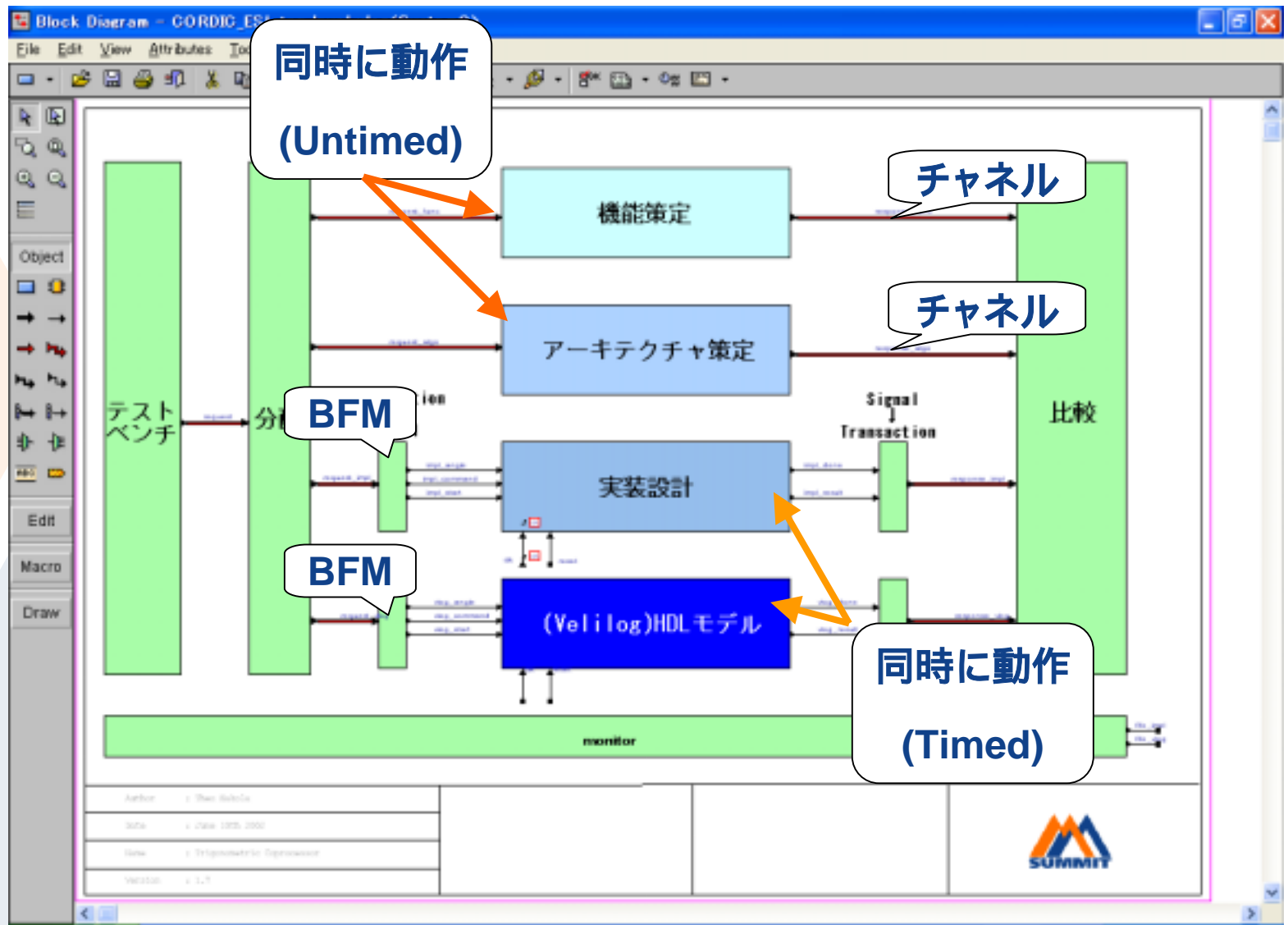
同一環境で機能策定から実装まで

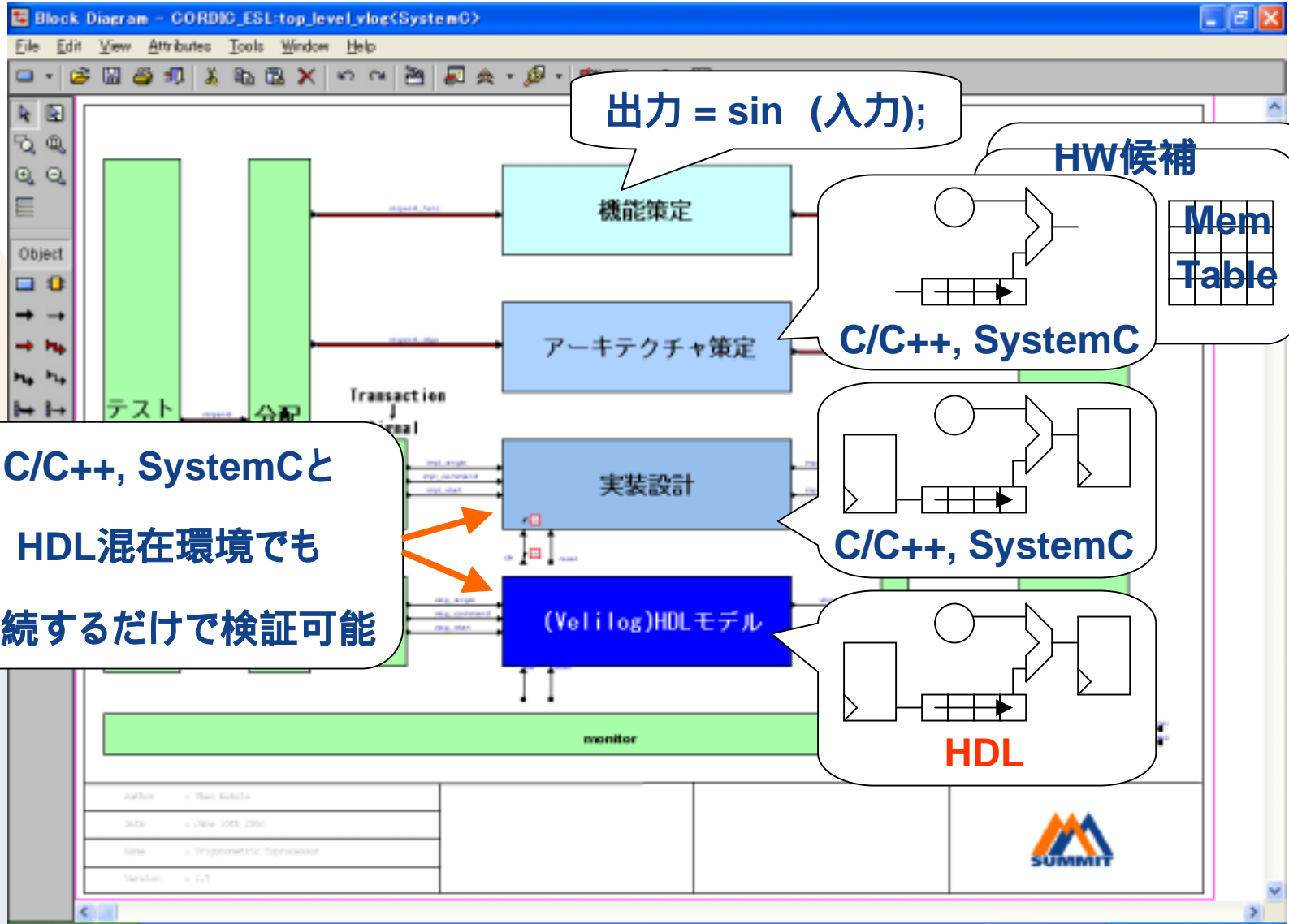
- ◆ 設計の流れをシームレスに接続
- ◆ C/C++/SystemC/HDL混在検証可能

CODEC_ESL

Top_level_verilog

SystemDesignを使用した設計・1



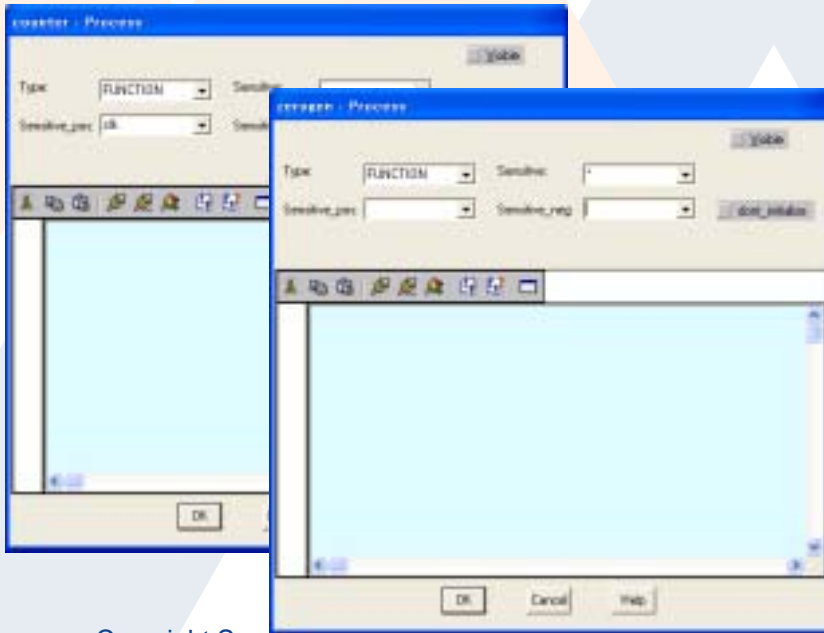
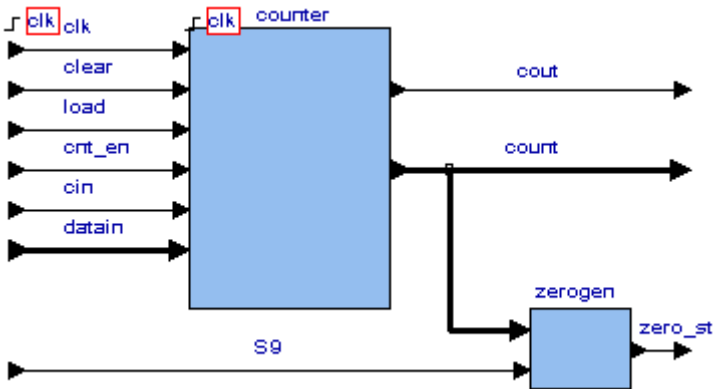


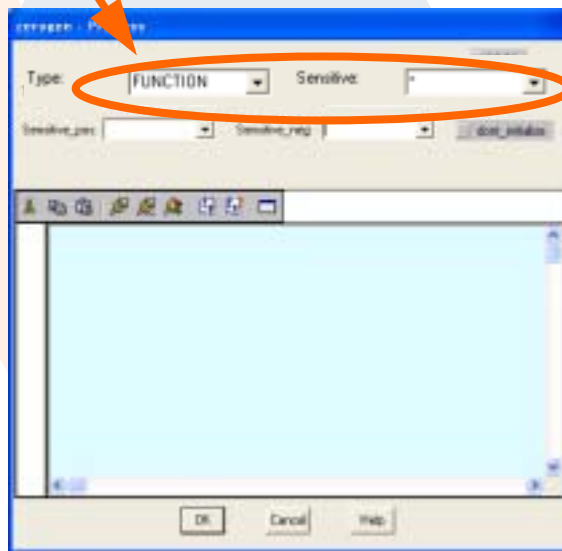
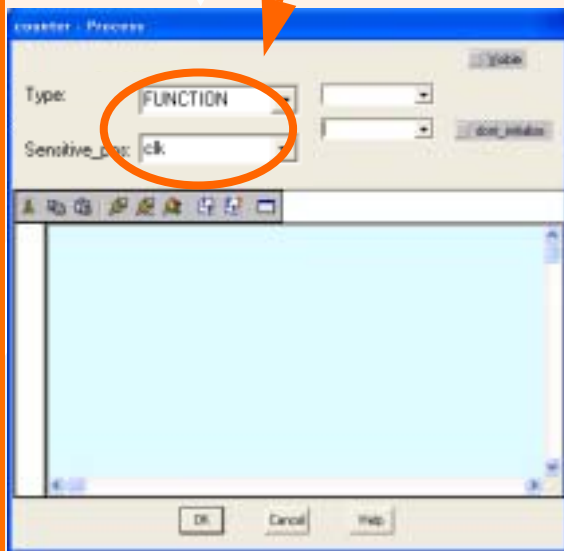
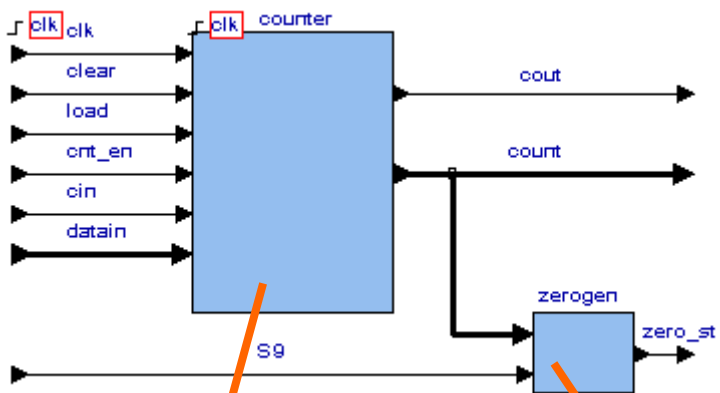
◆ 複雑な構文を覚える必要なし

```

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;
    }
    :
    :
};

```





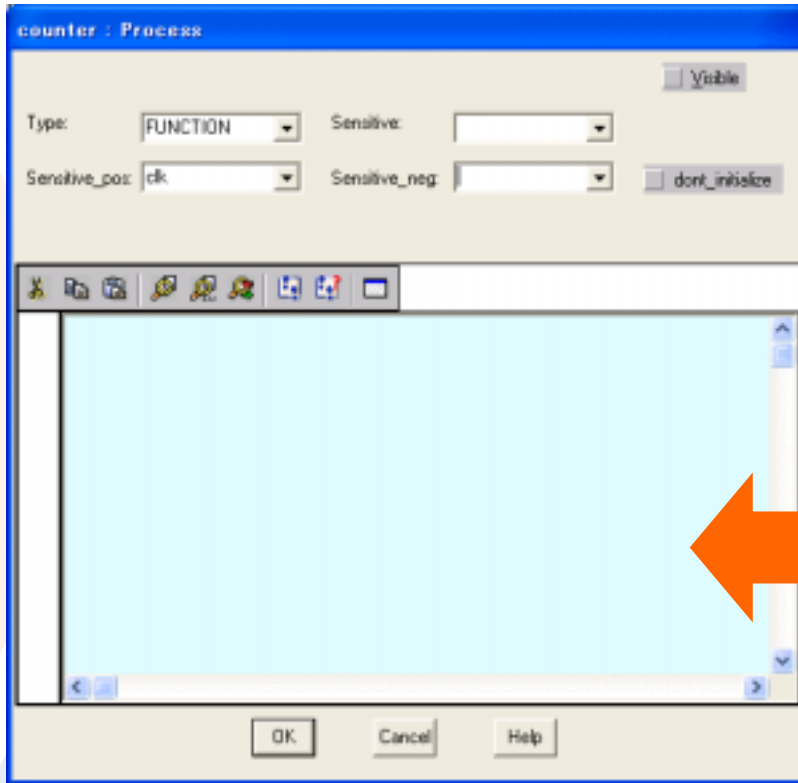
◆ センシティブリティ設定

◆ 動的センシティブリティ

- ◆ `a = b + c;`
`wait(a,sc_ns);`
- ◆ プロセス動作が動作条件で動的に変化する
- ◆ SystemC 2.0以上でサポート

◆ 静的センシティブリティ

- ◆ プロセス自体に定義
- ◆ `sensitive_pos`(立上り)
- ◆ `sensitive_neg`(立下り)
- ◆ `sensitive`
- ◆ *にて全ての信号を一括宣言可能



◆ 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;
    }
}
```

◆レビュー性の向上

◆デバッグの効率化

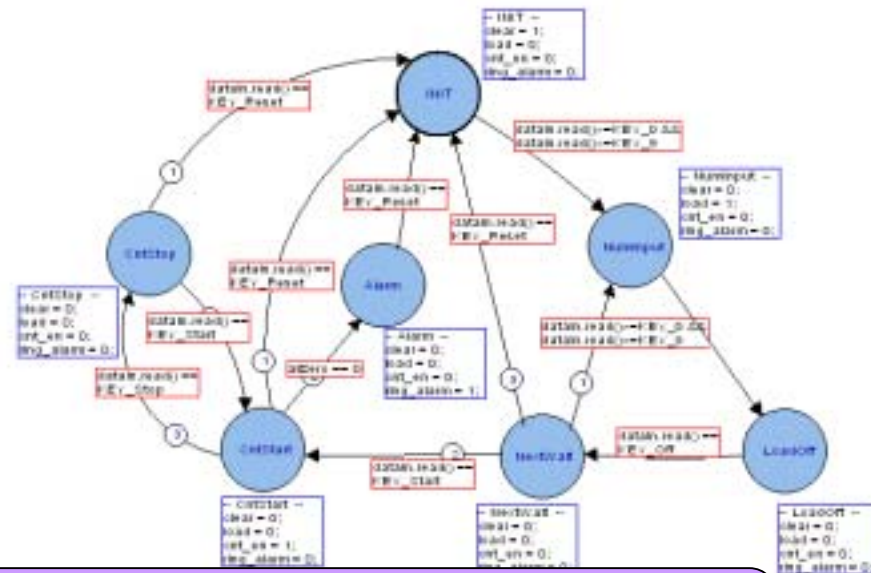
```
void dwlib1_Control_fc :: state_case(){
    clear_int = 0;
    cnt_en_int = 0;
    load_int = 0;
    ring_alarm_int = 0;

    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？



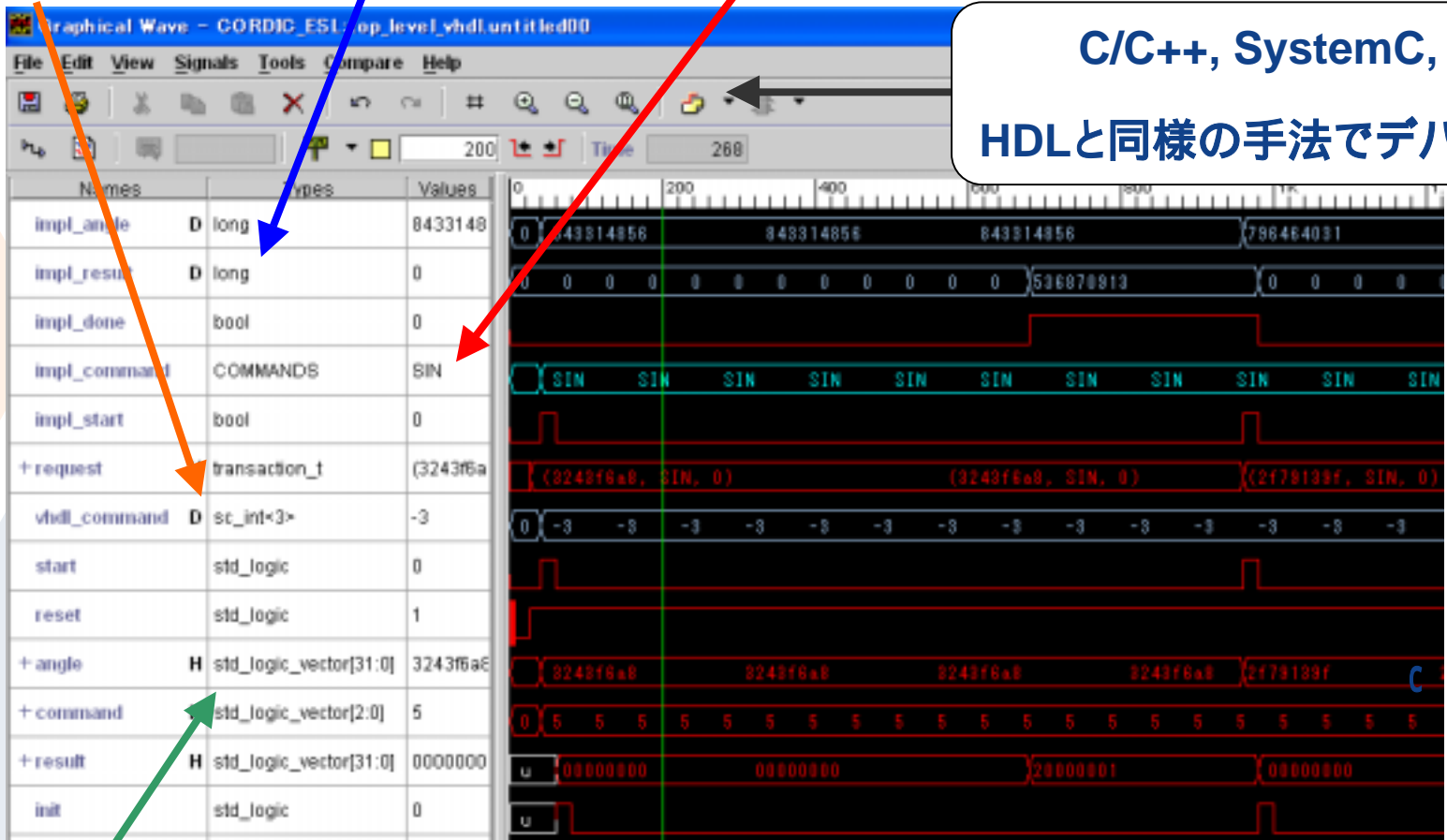
C/C++, SystemC, HDLを同一表示

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グ



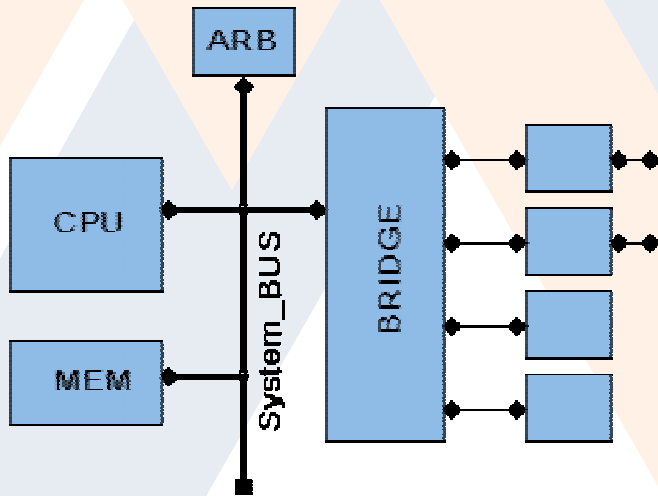
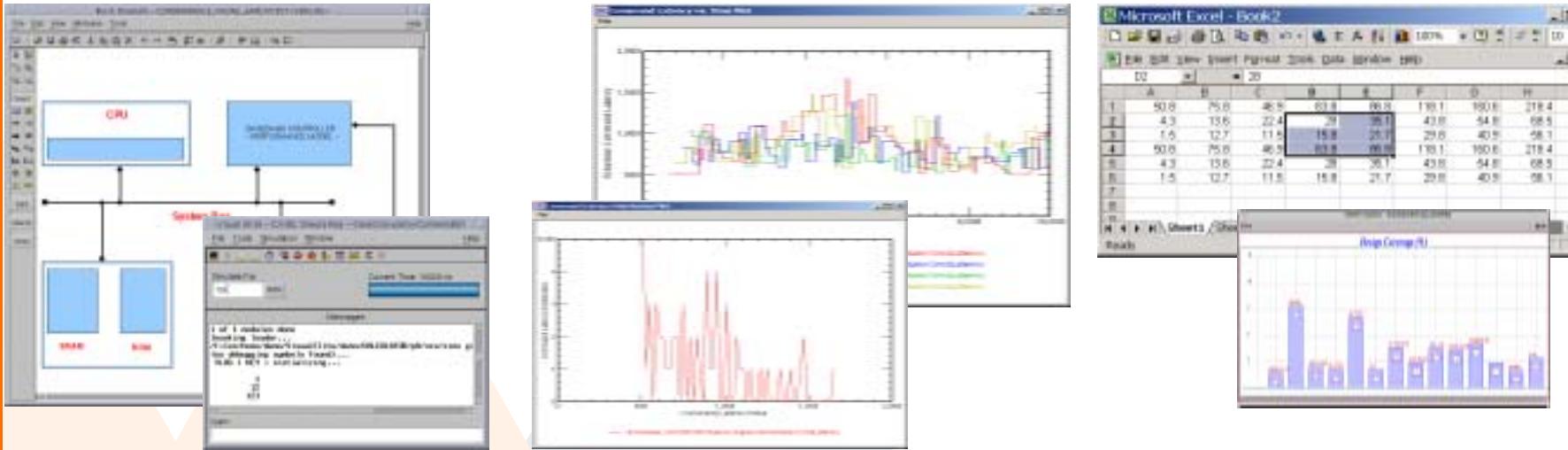
SystemCツールとのI/F

```
Generated SystemC - CORDIC_ESL:top_level_vhdl.top_level...
File Tools Window Help
SC_MODULE (CORDIC_ESL__top_level_vhdl)
{
  sc_in_clk clk;
  int max_waiting_time;
  sc_signal<int> fifo_impl;
  sc_signal<int> fifo_vhdl;
  sc_signal<long> impl_angle;
  sc_signal<COMMANDS> impl_command;
  sc_signal<bool> impl_done;
  sc_signal<long> impl_result;
  sc_signal<bool> impl_start;
  sc_fifo<transaction_t> request;
  sc_fifo<transaction_t> request_algo;
  sc_fifo<transaction_t> request_func;
  sc_fifo<transaction_t> request_impl;
  sc_fifo<transaction_t> request_vhdl;
  sc_signal<bool> reset;
  sc_fifo<transaction_t> response_algo;
  sc_fifo<transaction_t> response_func;
  sc_fifo<transaction_t> response_impl;
  sc_fifo<transaction_t> response_vhdl;
  sc_signal<long> vhdl_angle;
  sc_signal<sc_int<3>> vhdl_command;
  sc_signal<bool> vhdl_done;
  sc_signal<long> vhdl_result;
  sc_signal<bool> vhdl_start;
}
```

- ◆ 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

The MathWorks
MATLAB COMPILER

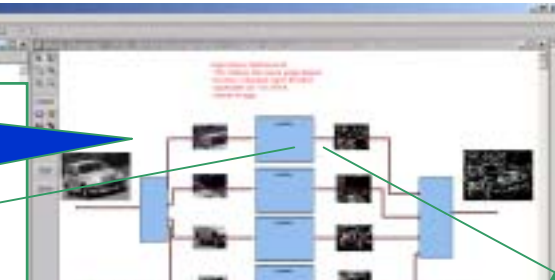
.M ファイルから ANSI C/C++
への変換



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



```
Edge_Filter : Process
Initialization -> Resonator -> Local Functions
// Line 0 :
write_line_data(lineout, edgebuff, apdata.width, 0);
// Line 1 to height-3
for (i=1; i<apdata.height-2; i++)
{
SOBEL_CONVOLUTION(line1, line2, line3, lineout, apdata.width, emap, void);
write_line_data(lineout, edge_buff, apdata.width, i);

"line1 = "line2;
"line2 = "line3;
line3 = get_line

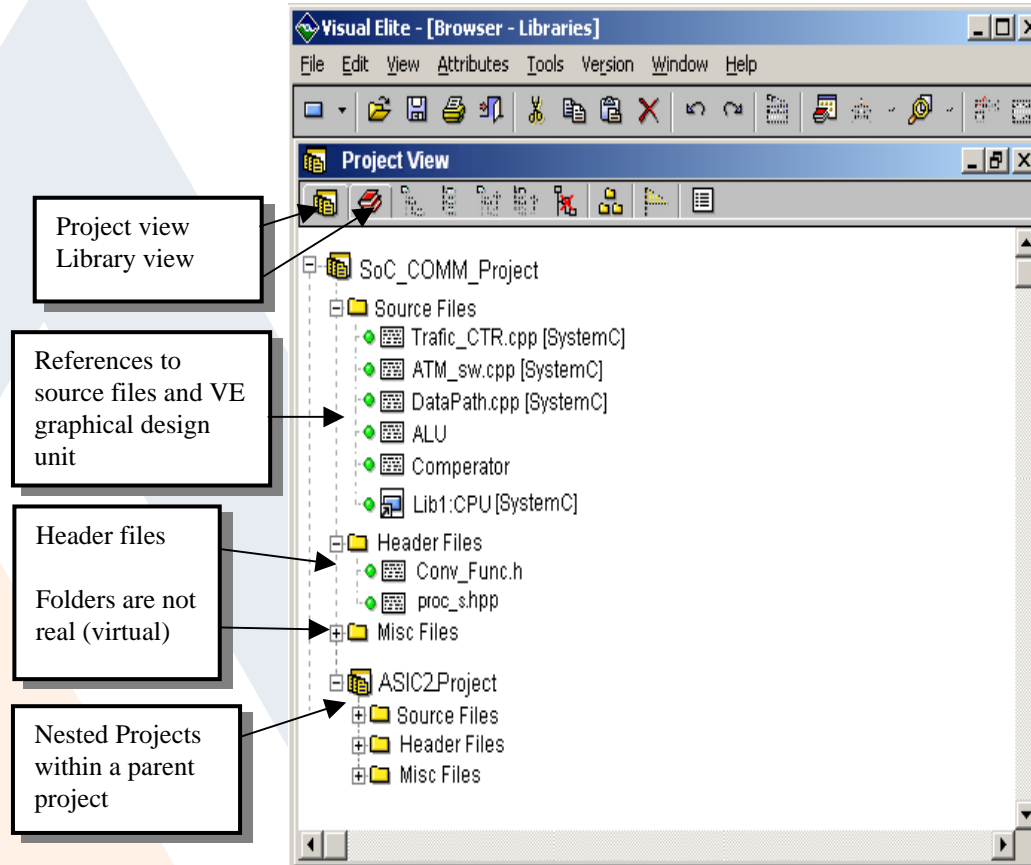
waitfor VERB
printf("F
atendit
}
```

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

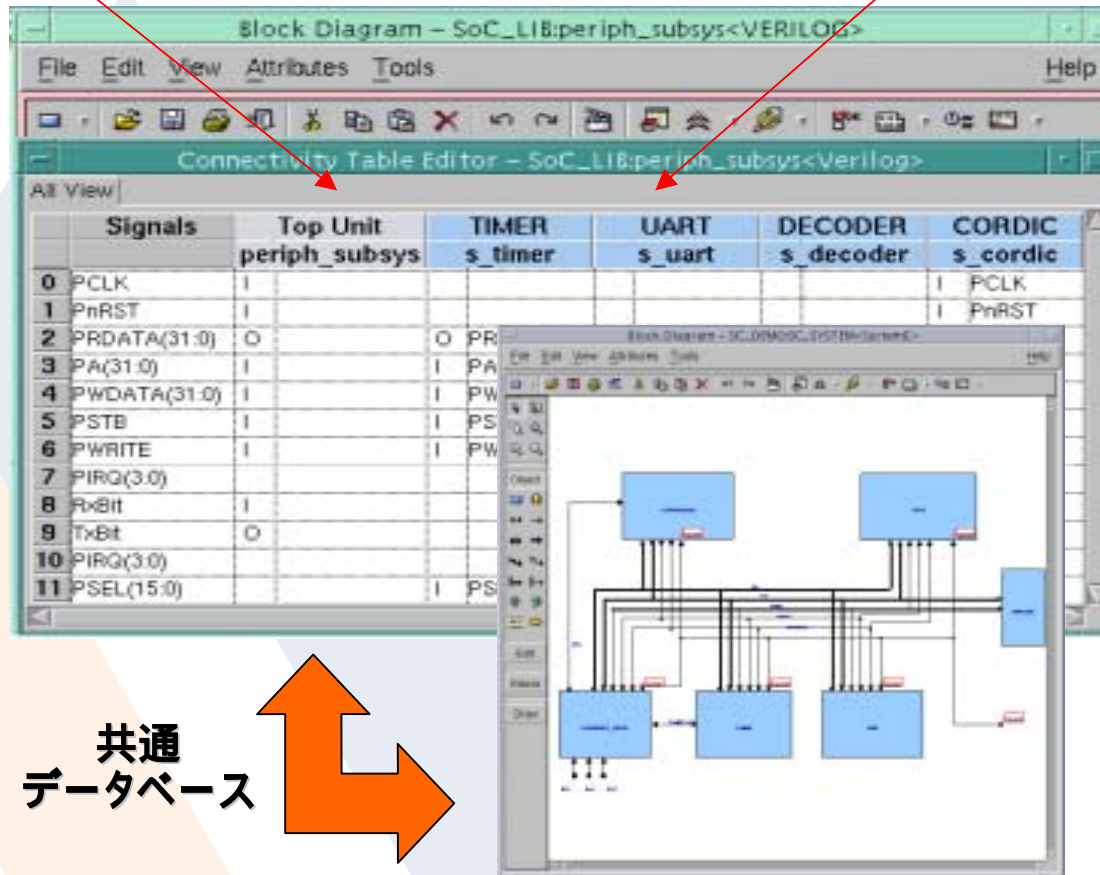
- ◆ あらゆるSystemC / HDLテキスト環境にプラグイン
- ◆ グラフィクスとテキスト混在デザインへのアプローチに適合
- ◆ テキスト・ユーザーのためのVisualによる検証手段とマネジメント機能
- ◆ 設計資産を新しい設計プロジェクトにシームレスに統合
- ◆ C, SystemC, VHDL, Verilogすべてのファイル及び構造をサポート



The top unit

Components

- ◆ 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



| Signals | Top Unit | TIMER | UART | DECODER | CORDIC |
|----------------|---------------|---------|--------|-----------|----------|
| | periph_subsys | s_timer | s_uart | s_decoder | s_cordic |
| 0 PCLK | I | | | | I PCLK |
| 1 PnRST | | | | | I PnRST |
| 2 PRDATA(31:0) | O | O PR | | | |
| 3 PA(31:0) | I | I PA | | | |
| 4 PWDATA(31:0) | I | I PW | | | |
| 5 PSTB | I | I PS | | | |
| 6 PWRITE | I | I PW | | | |
| 7 PIRQ(3:0) | | | | | |
| 8 RxBit | I | | | | |
| 9 TxBit | O | | | | |
| 10 PIRQ(3:0) | | | | | |
| 11 PSEL(15:0) | I | I PS | | | |

共通
データベース



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
 - (HDL-RTL比1000倍以上)
 - (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:

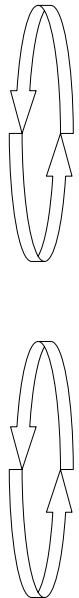
```
func Z(){
  if(A){
    B = 1; // Logic
  }else{...
}

func Y(){
  OUT = B & C;
}
```

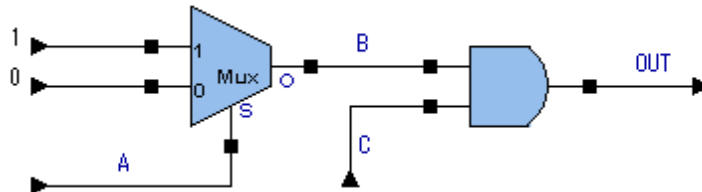
Verilog
の場合:

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

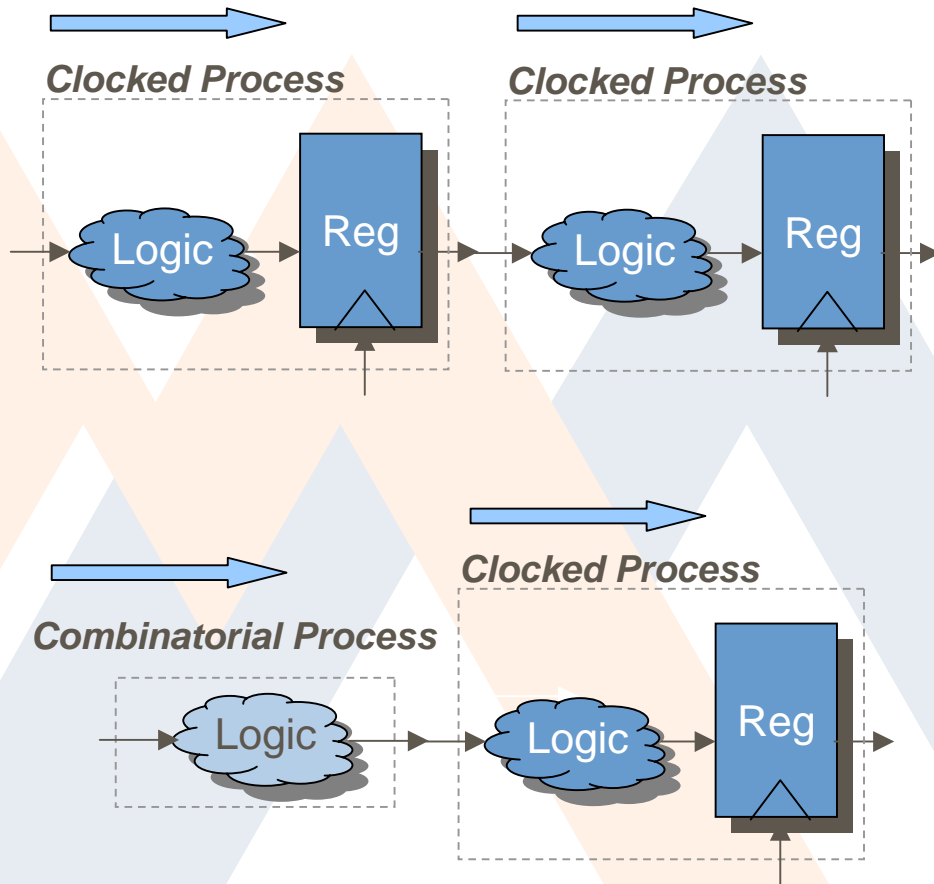
always @(B or C) begin
  OUT <= B & C;
end
```



C/C++では関数の処理順番を
間違えると正しい結果が出ません

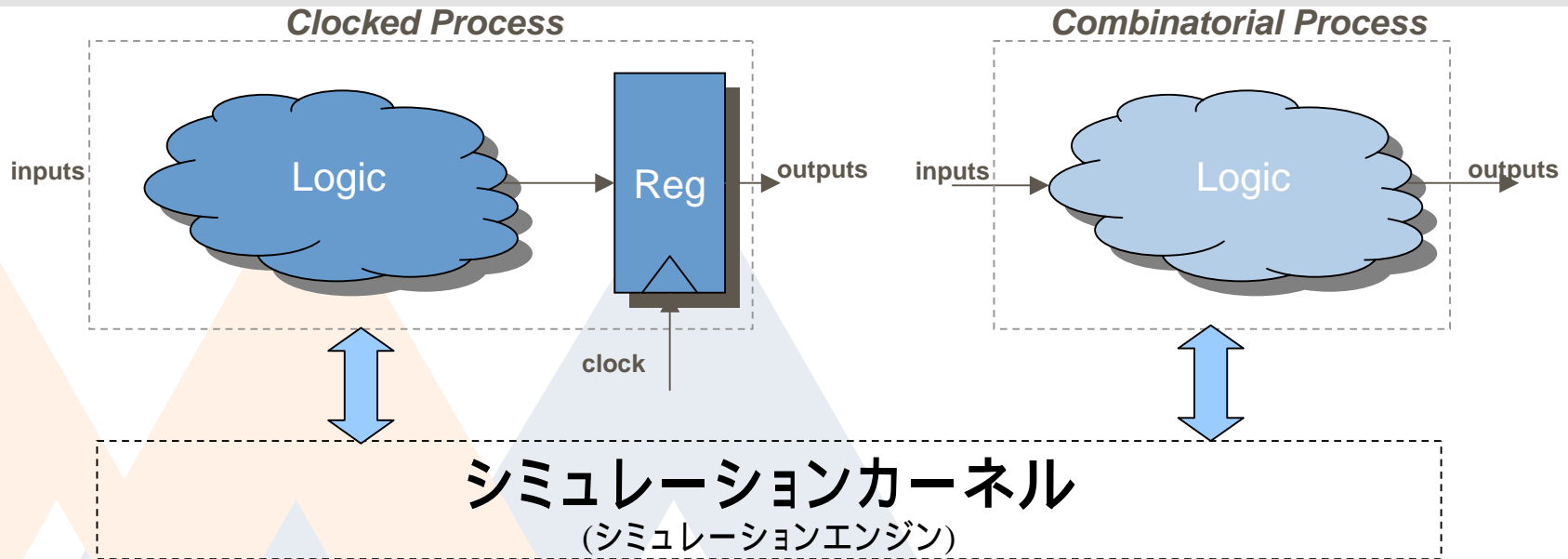


HDLでは各論理が独立動作するので
処理順番を気にする必要はありません

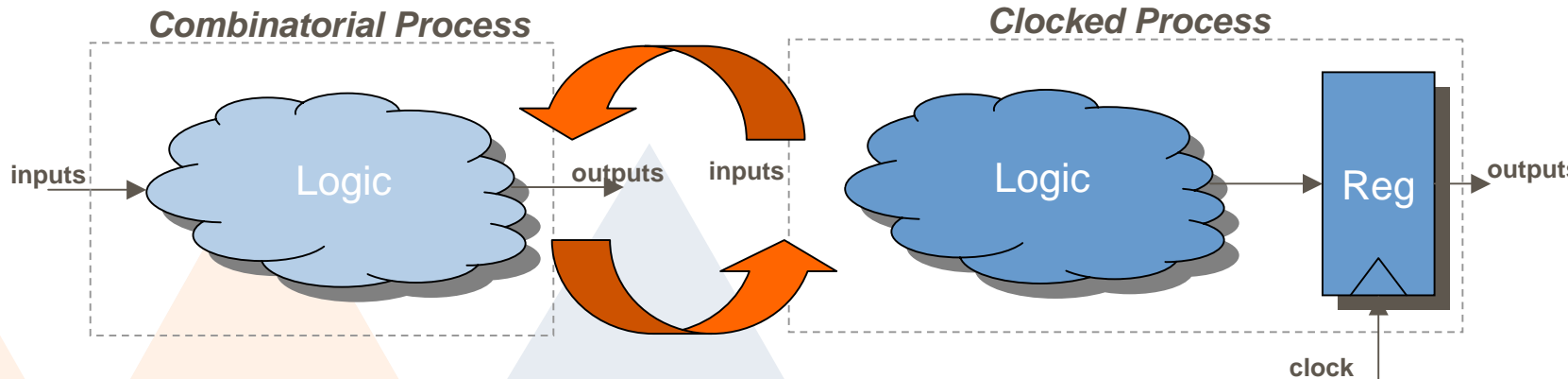


- ◆ 評価順番を自動認識
並べ替え
- ◆ 逐次処理のCソースを
内部生成
 - ◆ 実行イメージ作成
 - ◆ Simulatorカーネル
依存しないシミュレー
ション
 - ◆ Simulation中にダイ
ナミックに処理順番
が変わらず高速

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



シミュレーション前に評価順番を自動並べ替え(スタック・スケジュール)

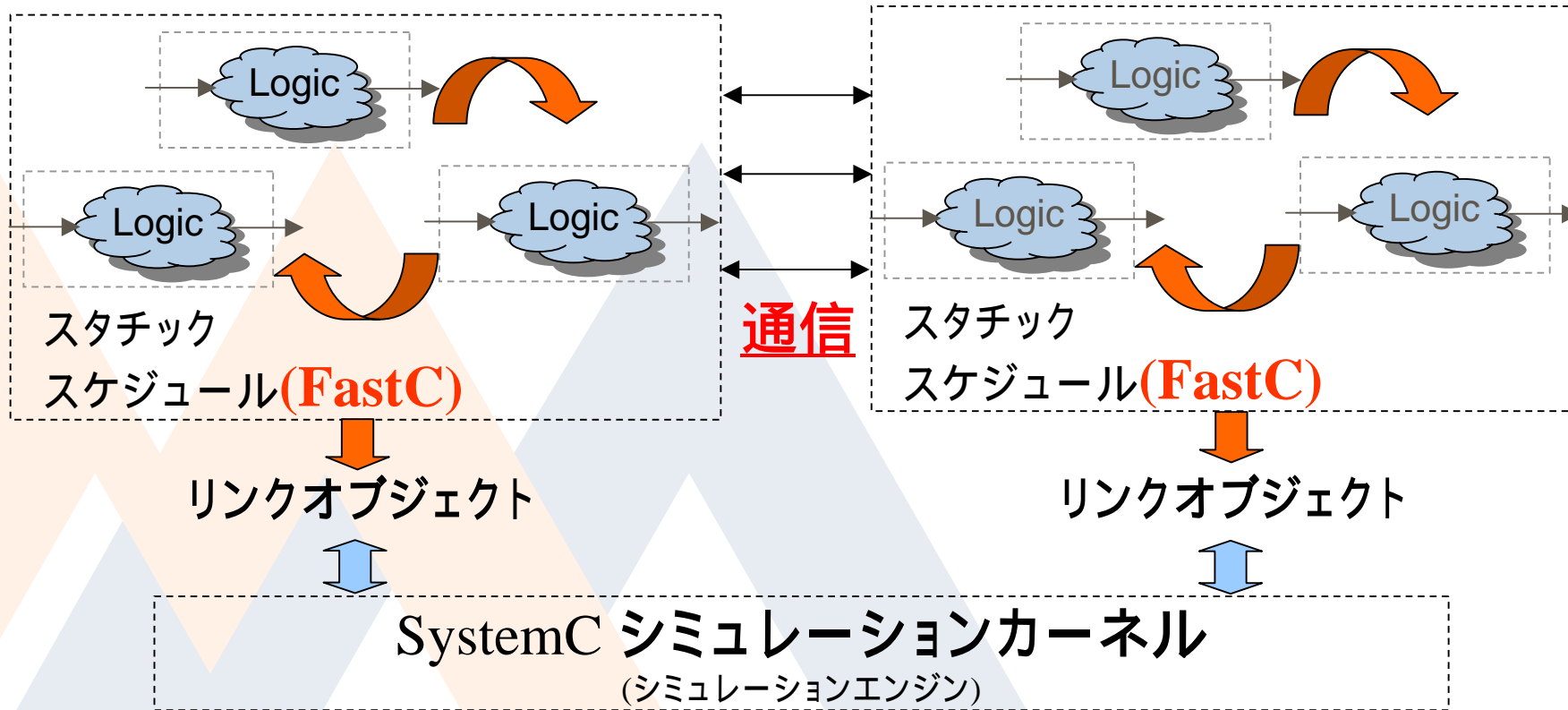


*.exe を作ってシミュレーション

- ◆ 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



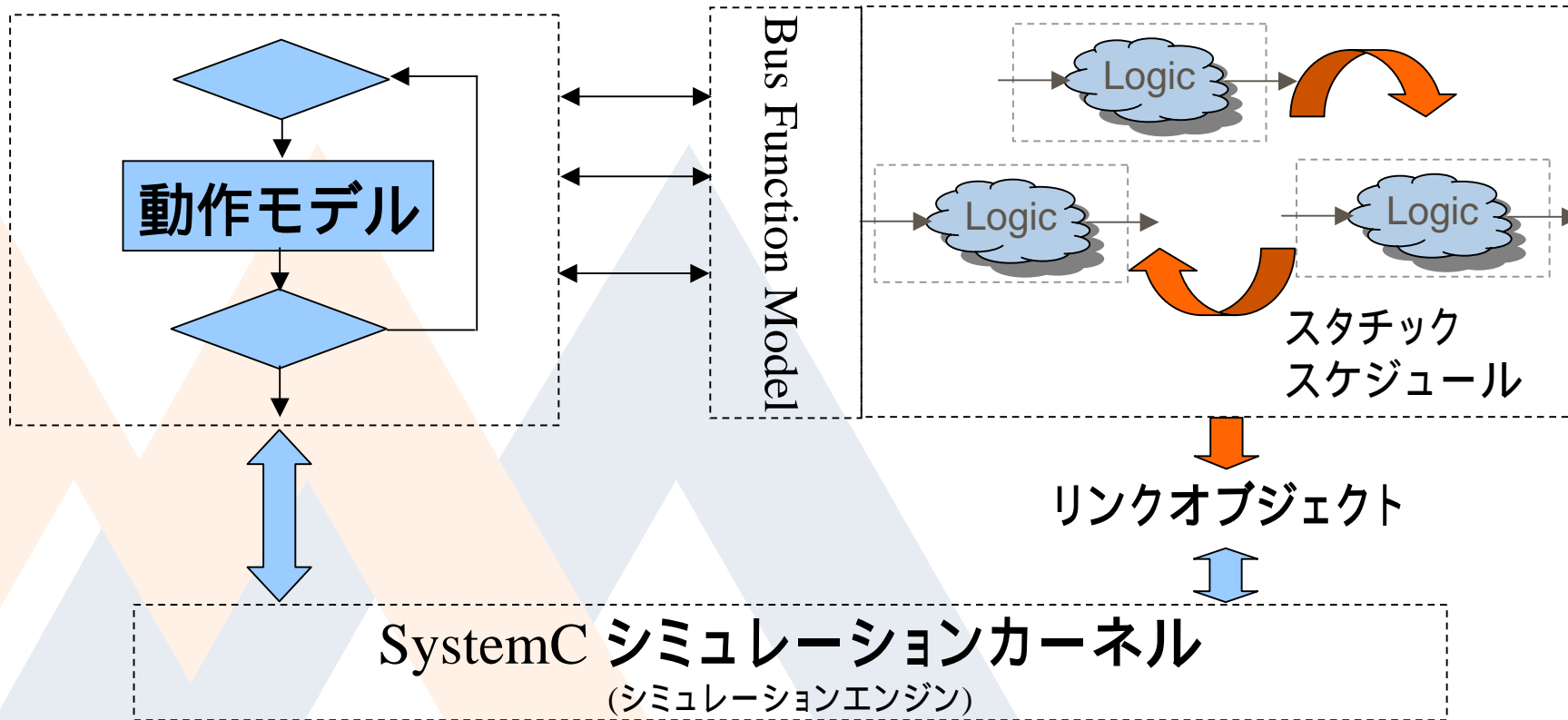
種々のHW表現も SystemC-Simと協調で解決！



- ◆ 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テクノロジーで処理可能



抽象度が異なるモデルでも FastC+SystemC-Simで解決！

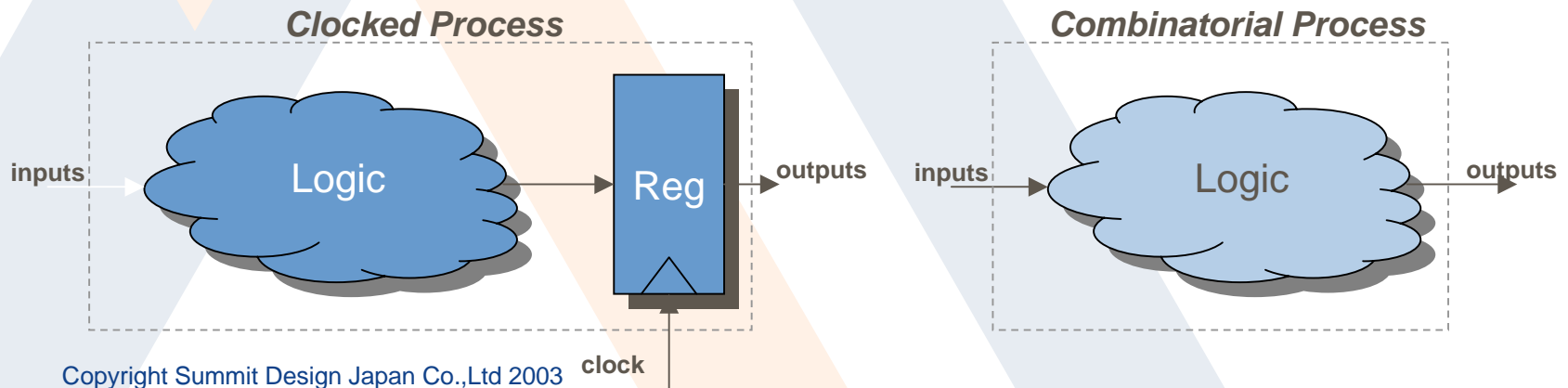


- ◆ **VisualElite は SystemCシミュレータ内蔵**
- ◆ **抽象度が異なるモデルでも検証可能**
- ◆ **RTL抽象度のC/C++,SystemCはFastCで高速シミュレーション**



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

GALC_CTRL_BLK_A : Process

Type: FUNCTION Sensitive: Sensitive_pos: clk Sensitive_neg: clr dont_initialize

```
if( clr == CLR_ENABLE ){
  inter_reg = 0;
  current_state = S0;
}else{
  switch( current_state ){
  case S0 :
    if( enc.read() != ALL_RELEASED ){
      inter_reg = enc.read();
      inter_reg[4] = (sc_uint<1>)0;
      current_state = S1;
    }
    break;
  case S1 :
    if( enc.read() == ALL_RELEASED ){
      current_state = S2;
    }
    break;
  case S2 :
    if( enc.read() != ALL_RELEASED ){
      inter_reg = enc.read() + inter_reg;
      current_state = S3;
    }
    break;
  }
```

OK Cancel Help

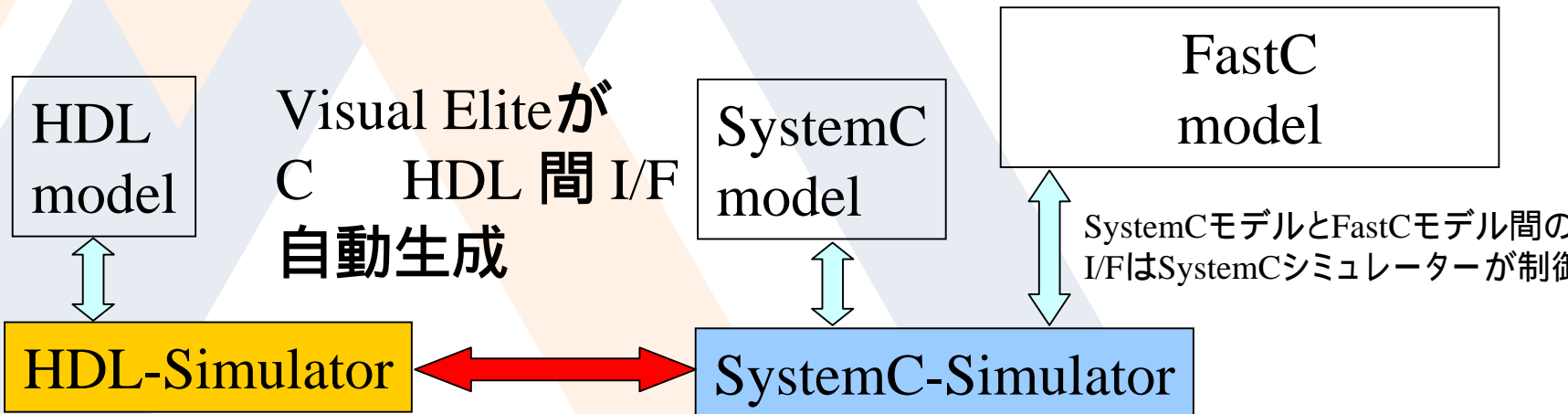
Generated Verilog - FastC_calc:FastC_calc_blk.FastC_cal...

```
always @ (posedge clk or negedge clr)
begin : visual_CALC_CTRL_BLK_A
  if (clr == 0)
  begin : visual_begin_if_0
    inter_reg = 0;
    current_state = `S0;
  end
  else
  begin : visual_begin_else_0
    case (current_state)
    `S0 :
      begin : visual_begin_1
        if (enc != `hf)
        begin : visual_begin_if_2
          inter_reg = enc;
          inter_reg[4] = 0;
          current_state = `S1;
        end
      end
    `S1 :
      begin : visual_begin_3
        if (enc == `hf)
        begin : visual_begin_if_4
          current_state = `S2;
        end
      end
    `S2 :
      begin : visual_begin_5
```



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

The screenshot displays two windows from a software tool. The top window, titled 'ctl : Process', shows configuration options for a process: Type is set to 'THREAD', Sensitive is empty, Sensitive_pos is 'CLK', and Sensitive_neg is empty. The bottom window, titled 'Generated Verilog - SII_Fair1:SysC_Wrappe...', shows the resulting Verilog code. The code defines a clock-driven process 'visual_ctl' that contains a 'while (1)' loop. Inside this loop, there is a 'begin : visual_continue_6' block with a delay of 20 units and assignments to 'updw1' and 'updw2'. This is followed by a 'begin : visual_break_6' block with a delay of 40 units and assignments to 'updw1' and 'updw2'. The loop ends with 'end', 'end', and 'end' statements, followed by 'endmodule'.

```
while(1){
  wait(20, SC_NS);
  updw1 = 1;
  updw2 = 0;
  wait(40, SC_NS);
  updw1 = 1;
  updw2 = 1;
  wait(50, SC_NS);
  updw1 = 0;
  updw2 = 0;
  wait(40, SC_NS);
}
```

```
always @ (posedge CLK)
begin : visual_ctl
  begin : visual_break_6
    while (1)
      begin : visual_continue_6
        # 20;
        updw1 <= 1;
        updw2 <= 0;
        # 40;
        updw1 <= 1;
        updw2 <= 1;
        # 50;
        updw1 <= 0;
        updw2 <= 0;
        # 40;
      end
    end
  end
end
endmodule
```

◆ 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

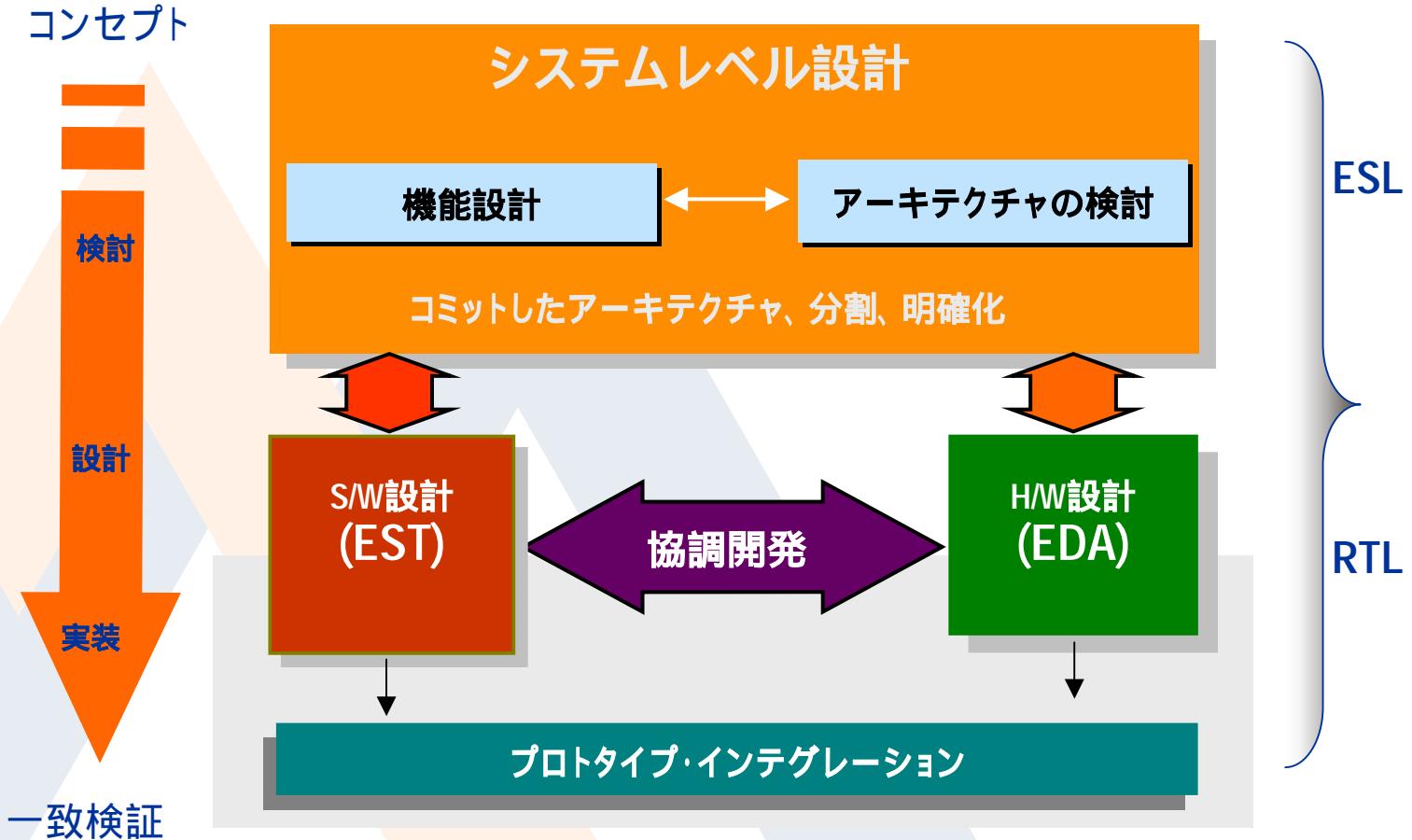


NEW!

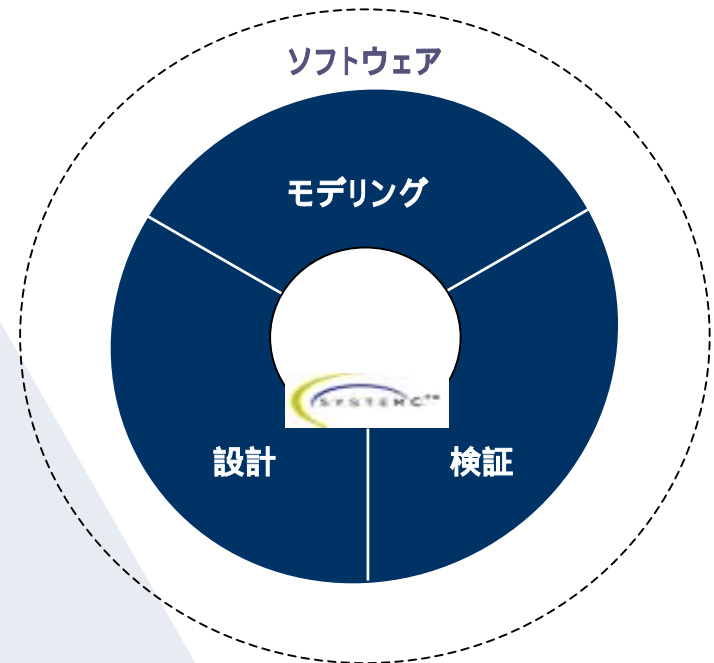
Visual Elite ESC

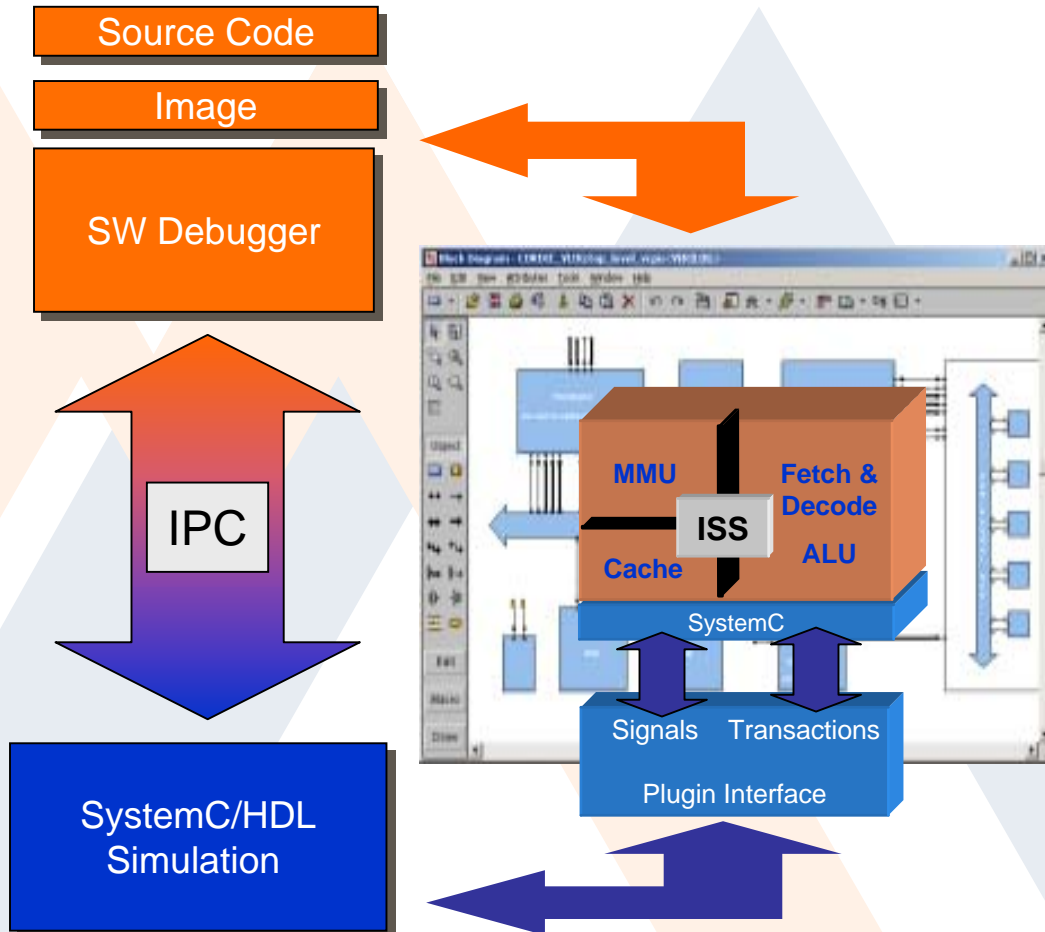
組込みシステム

SystemCベースHW/SW協調検証環境



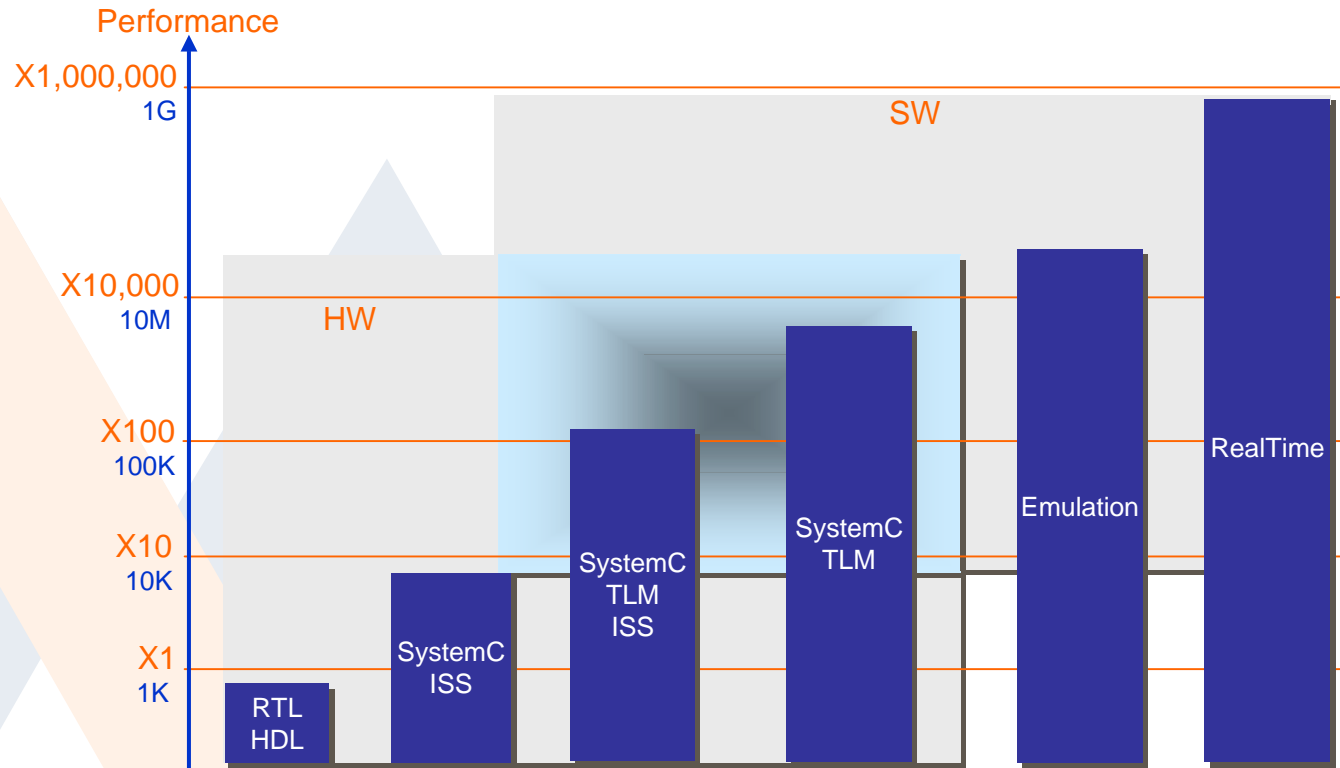
- ◆ SystemCは、システム・モデリングのすべての範囲にわたり、S/Wプロトタイピング、H/Wインプリメンテーション、統一されたHW/SW共通言語を提供する
- ◆ SystemCによる標準化により、ツール・ベンダーはオートメーション・メカニズムを開発
- ◆ SystemCは、過去10数年蓄積されたインフラ、C/C++ライブラリへの膨大な投資、ツール群を活用できる





- ◆ 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協調プラットフォーム
- ◆ トランザクション・レベルでの通信
- ◆ パーチャル・プロトタイピング

検証プラットフォームとパフォーマンス



Verification Platforms

- ◆ 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 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**

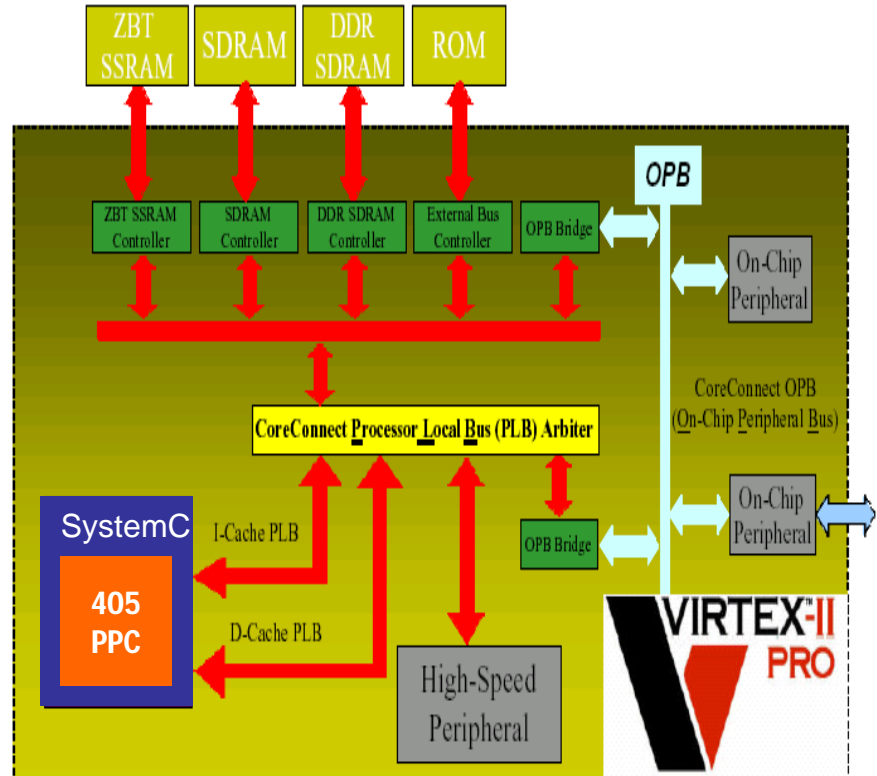
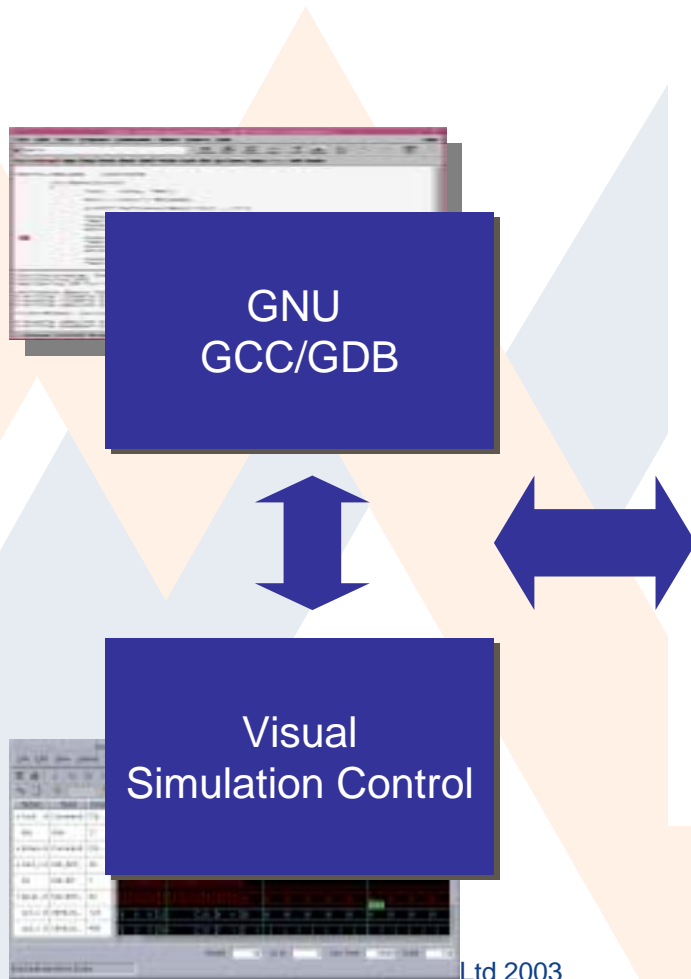
CCM Models

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

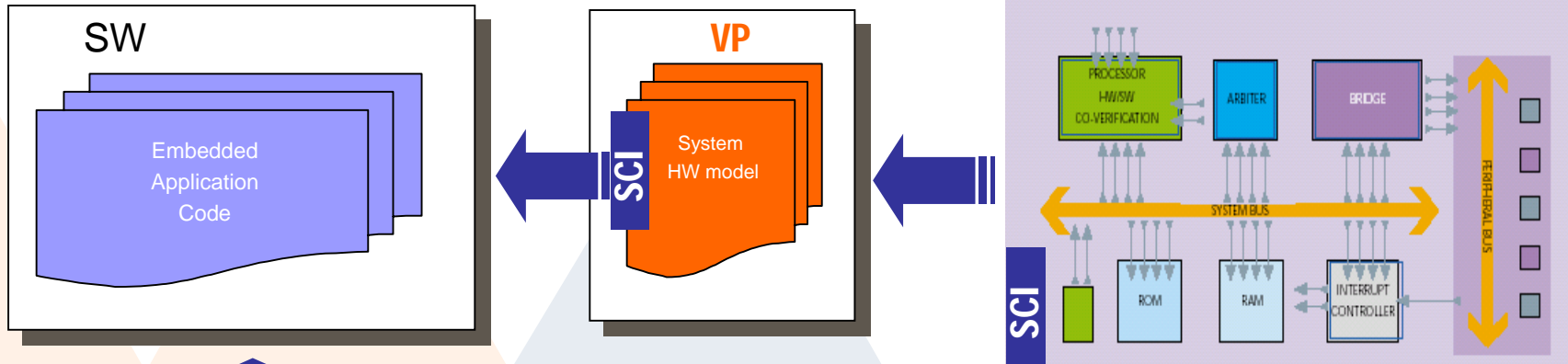


Visual Elite ESC Virtex-II PRO プラットホーム

PowerPC, CoreConnect and IP



C/SystemC based Virtual Prototype



- ◆ 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



Software IDE