Electronic Design and Solution Fair 2009

with FPGA/PLD Design Conference

今さら聞けない高位合成
～一から学ぶ高位合成～

シャープ株式会社
電子デバイス事業本部 副参事
山田　晃久

Design the Future ! — Cutting-edge Technologies Excite the Senses —

www.edsfair.com

ハードウェア設計と抽象度

RTL記述

論理合成

ゲートレベル記述

動作仕様
設計制約（コスト、性能、消費電力）

システムの実現方式を決定
（アーキテクチャ、アルゴリズム）
システム分割（HW/SW）→機能ブロック

ハードウェアの処理を設計
（データ処理回路と制御回路）
クロックレベルのタイミング設計

論理素子（AND/OR/NOTなど）と
記憶素子（ﾌﾘｯﾌﾟﾌﾛｯﾌﾟ）による回路設計

フロアプラン決定
配置配線

ハードウェア設計と抽象度

動作レベル記述

RTL記述

ゲートレベル記述

高位合成
（動作合成）

論理合成

動作仕様
設計制約（コスト、性能、消費電力）

システムの実現方式を決定
（アーキテクチャ、アルゴリズム）
システム分割（HW/SW）→機能ブロック

ハードウェアの処理を設計
（データ処理回路と制御回路）
クロックレベルのタイミング設計

論理素子（AND/OR/NOTなど）と
記憶素子（ﾌﾘｯﾌﾟﾌﾛｯﾌﾟ）による回路設計

フロアプラン決定
配置配線

動作レベル設計とRTL設計

- 動作レベル設計
  - 動作（アルゴリズム）を設計
  - How のみを設計
  - ハードウェアによる実装とは独立

- RTL（Register Transfer Level）設計
  - クロック毎のレジスタ間のデータ転送を設計
  - How と When と by What を設計
  - レジスタや演算回路とそれらの制御を設計
    （ある程度ハードウェアが見えている）

高位合成とは

- 設計対象の回路の動作・アルゴリズムからレジスタ転送レベル（RTL）の回路を自動で合成する技術
- High level synthesisの和訳
- 動作合成(Behavioral synthesis)とも呼ばれる

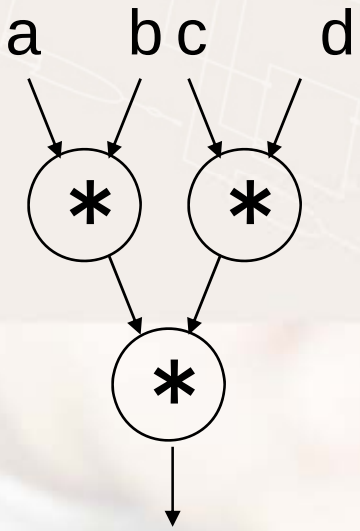
高位合成処理

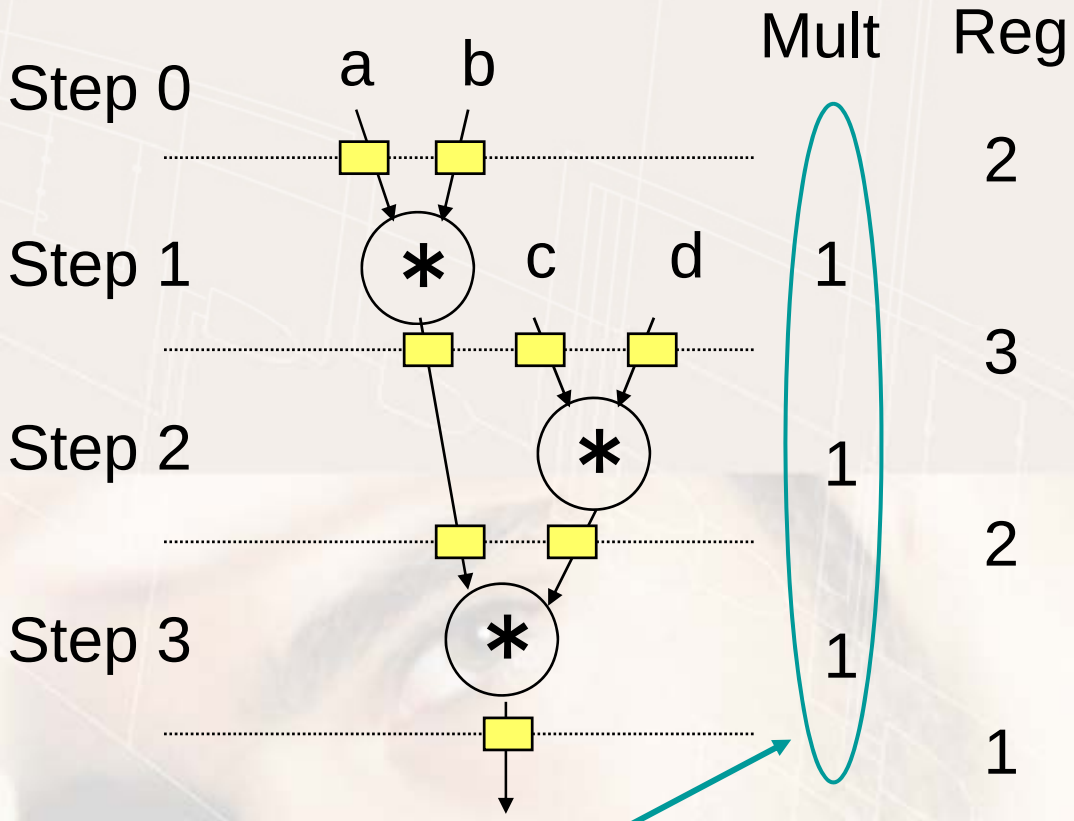
- HDL中の演算の
  1. 実行するサイクルを決定し(スケジューリング)、
  2. 各演算を演算器に割り当て、必要に応じて接続し(アロケーション)、
  3. それらの回路要素を制御する回路を生成する(制御回路生成)。

スケジューリング

Y = a * b * c * d;

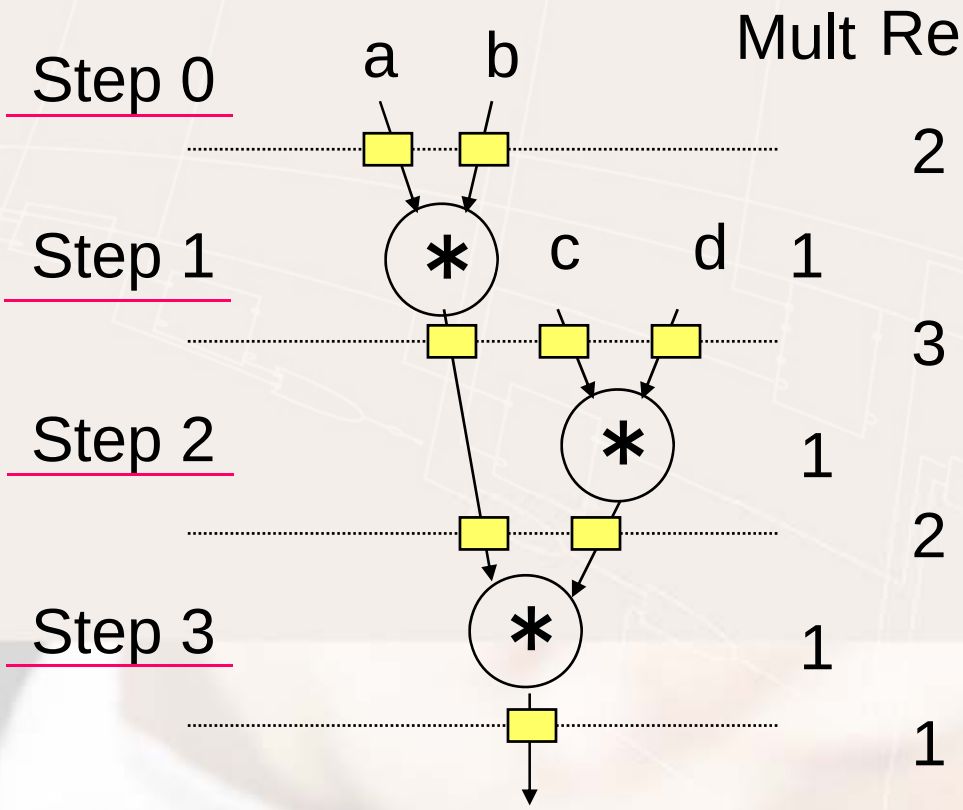
データフローグラフ



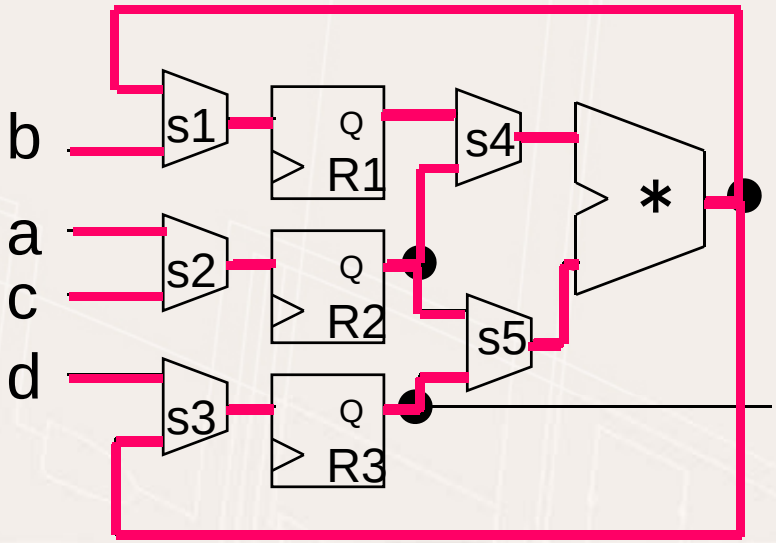


リソース制約： 使用できる乗算器数は1個

アロケーション



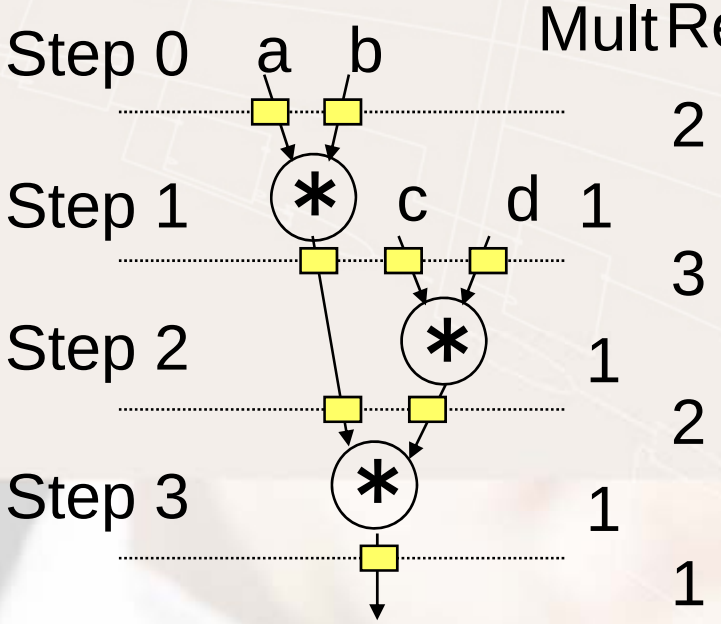
スケジューリング結果



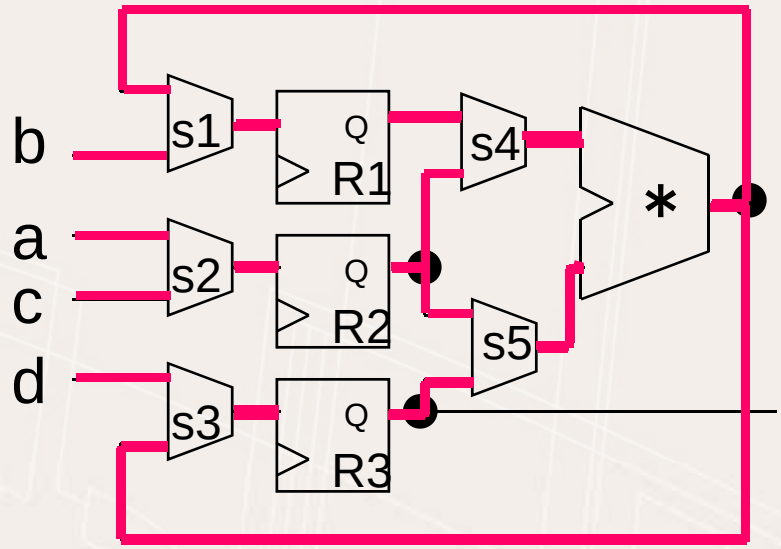
データパスの構成

演算器やレジスタなどの演算資源をどのように割当て、
共有するか（リソースシェアリング）を決定

制御回路生成



スケジューリング結果



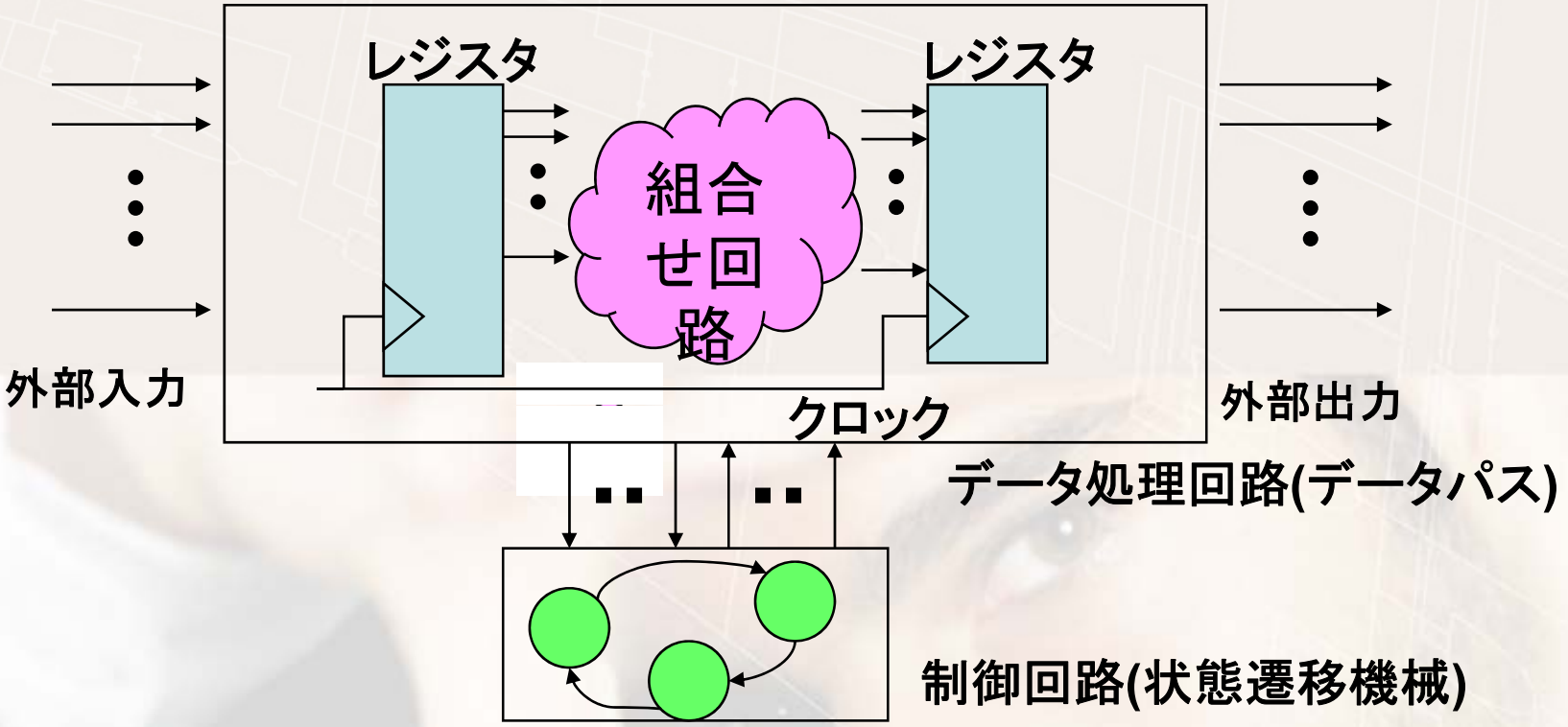
	s1	s2	s3	s4	s5	R1	R2	R3
Step 0	1	0	X	X	X	1	1	0
Step 1	0	1	0	0	0	1	1	1
Step 2	X	X	1	1	1	0	0	1
Step 3	X	X	X	0	1	0	0	1

RTL設計

- 設計の焦点
  - クロック毎のレジスタ間のデータ転送をどのようにするか？（同期設計）
  - レジスタ間のデータ処理と制御を検討
- 設計手法
  - RTレベルのハードウェア記述言語を使用
  - 論理合成を利用
    - コストや性能の制約にもとづいてさまざまな組合せ回路を合成（レジスタ構造は固定）

RTL記述

- レジスタや演算回路とそれらの制御を記述
- クロック毎のレジスタ間のデータ転送を記述



動作レベル設計

- 設計の焦点
  - どのようなアルゴリズムで機能を実現するか？
  - データフローを検討
- 設計手法
  - システム全体をまとまった機能ブロックに分割
  - 動作レベルのハードウェア記述言語を使用
  - 高位合成を利用
    - コストや性能の制約、入出力の制約（順序制約やタイミング制約）を与えてさまざまな構成の回路を合成

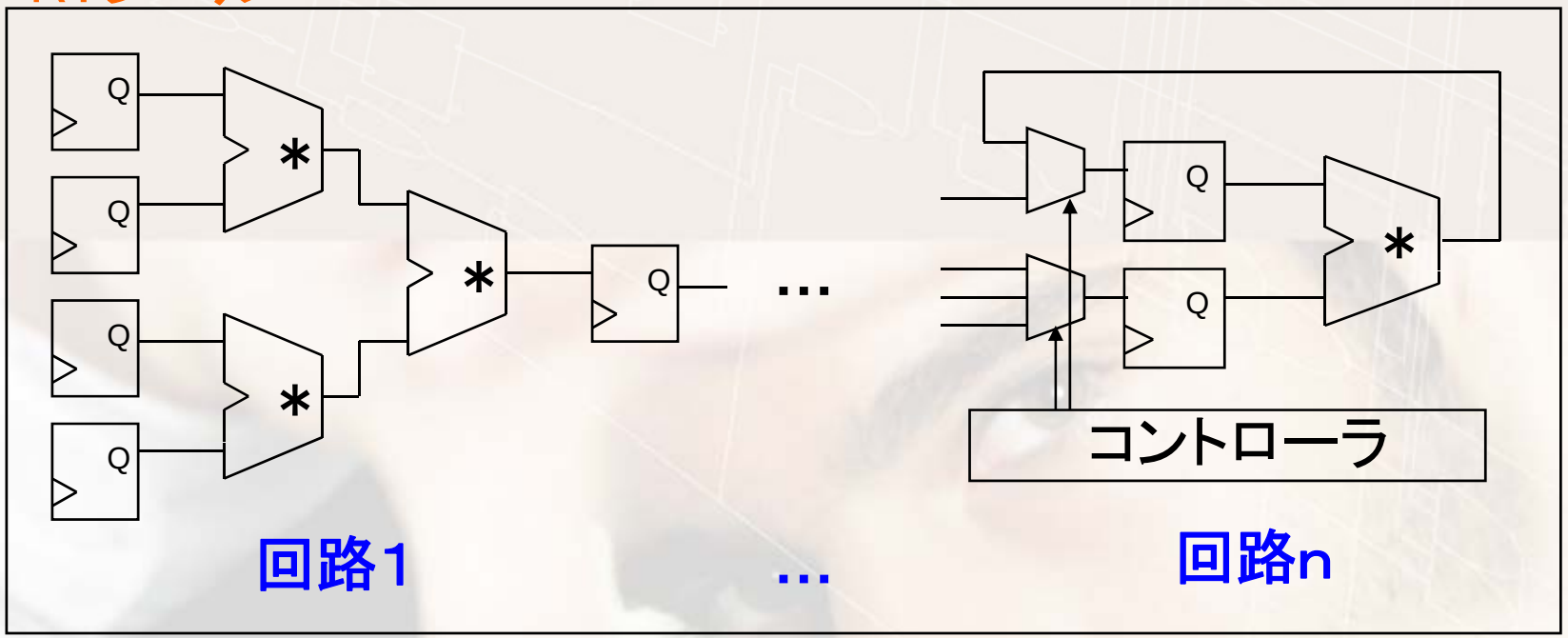
設計抽象度の差(1)

動作レベル

```
Y = a * b * c * d;
```

動作レベルでは
ハードウェア実装を
意識しない

RTレベル

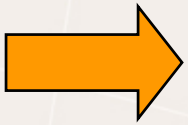


設計抽象度の差(2)

動作レベル

```
architecture behavior of MUL4 is
begin
  MULT:process(a,b,c,d)
  begin
      y <= a * b * c * d;
  end process ;
end behavior;
```

7行



行数で数～10倍

RTレベル

```
architecture RTL1 of MUL4 is
  signal aa, bb, cc, dd: unsigned(7 downto 0);
begin
  MULT:process(CLK)
  begin
      if CLK'event and CLK = 1 then
        aa <= a; bb <= b; cc <= c; dd <= d;
        y <= (aa * bb) * (cc * dd);
      end if;
  end process ;
end RTL1;
```

11行

```
architecture RTLn of MUL4 is
  signal mul: unsigned(7 downto 0);
  signal m1, m2: unsigned(7 downto 0);
  .....
begin
  MULT:process(CLK)
  begin
      if CLK'event and CLK = 1 then
        mul <= m1 * m2;
      end if;
  end process ;
  SEL1:process(CLK)
  begin
      if CLK'event and CLK = 1 then
        if stage = 1 then
          m1 <= a;
        else
          m1 <= mul;
        end if;
      end if;
  end process ;
  ....
  y <= mul;
end RTLn;
```

48行

動作レベル設計とRTL設計

- 動作レベル設計
  - How のみを設計
    → どのような処理を行うか？
- RTL設計
  - How と When と by What を設計

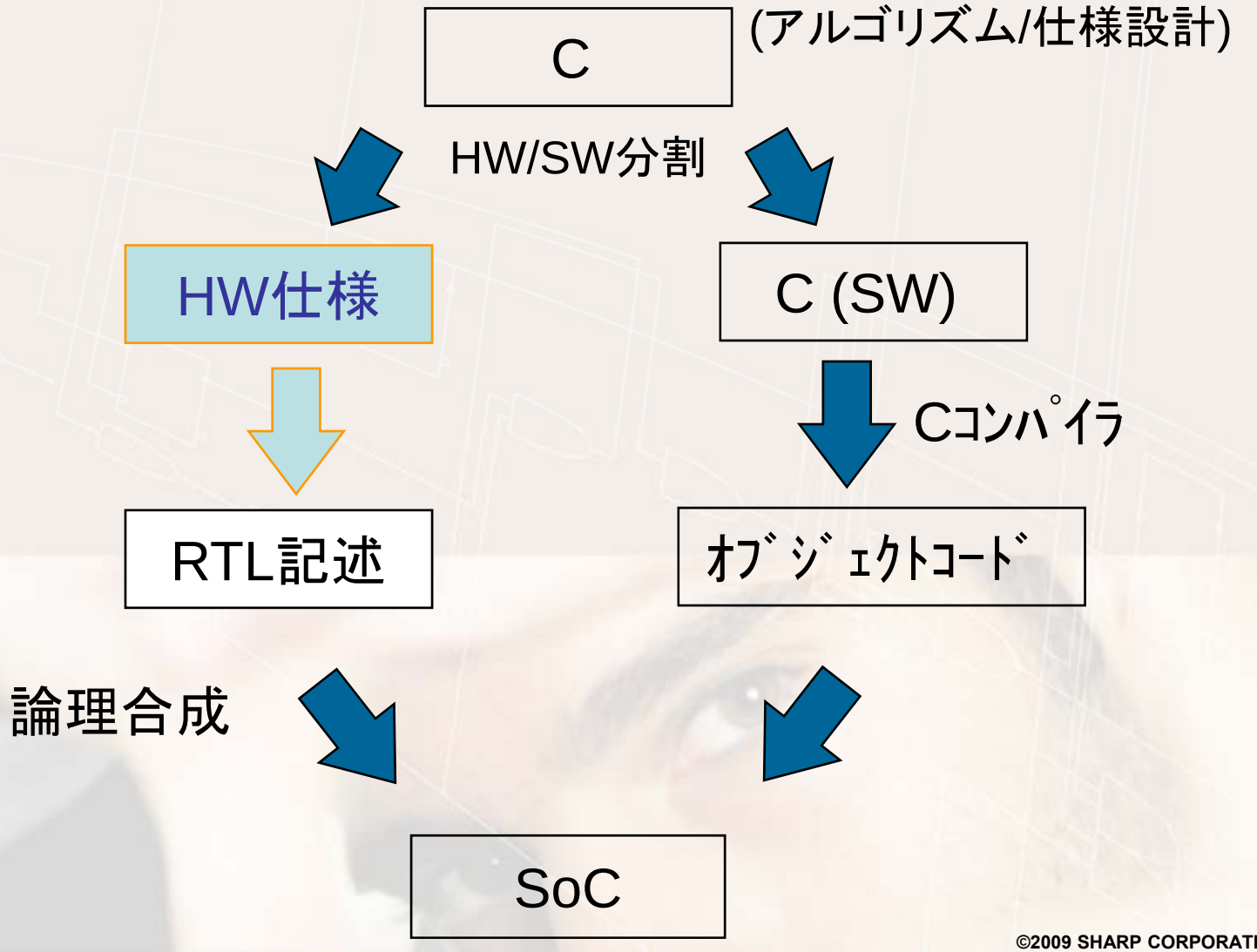
いつその処理を行うか？

スケジューリング

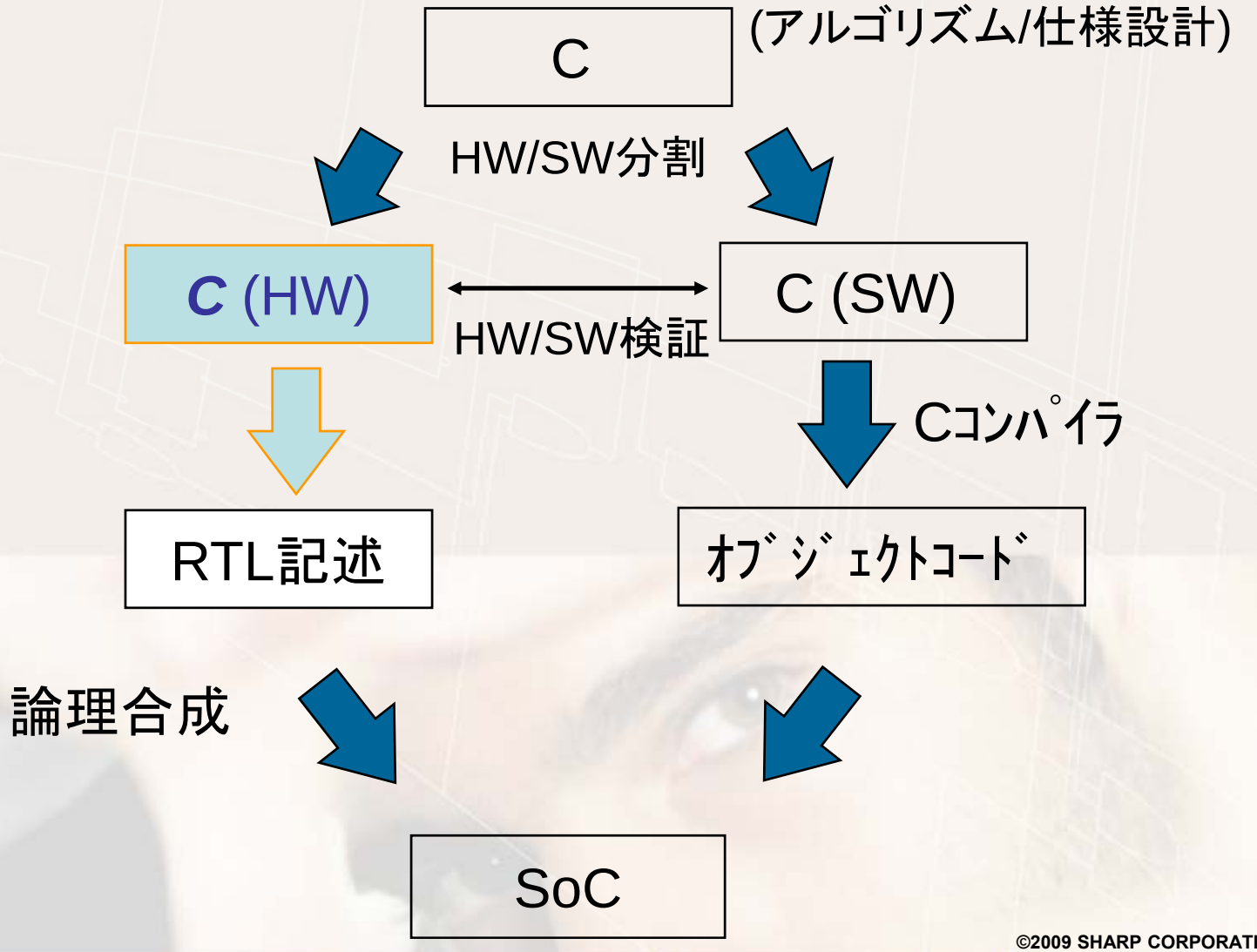
どの演算器がその処理を行うか？

アロケーション

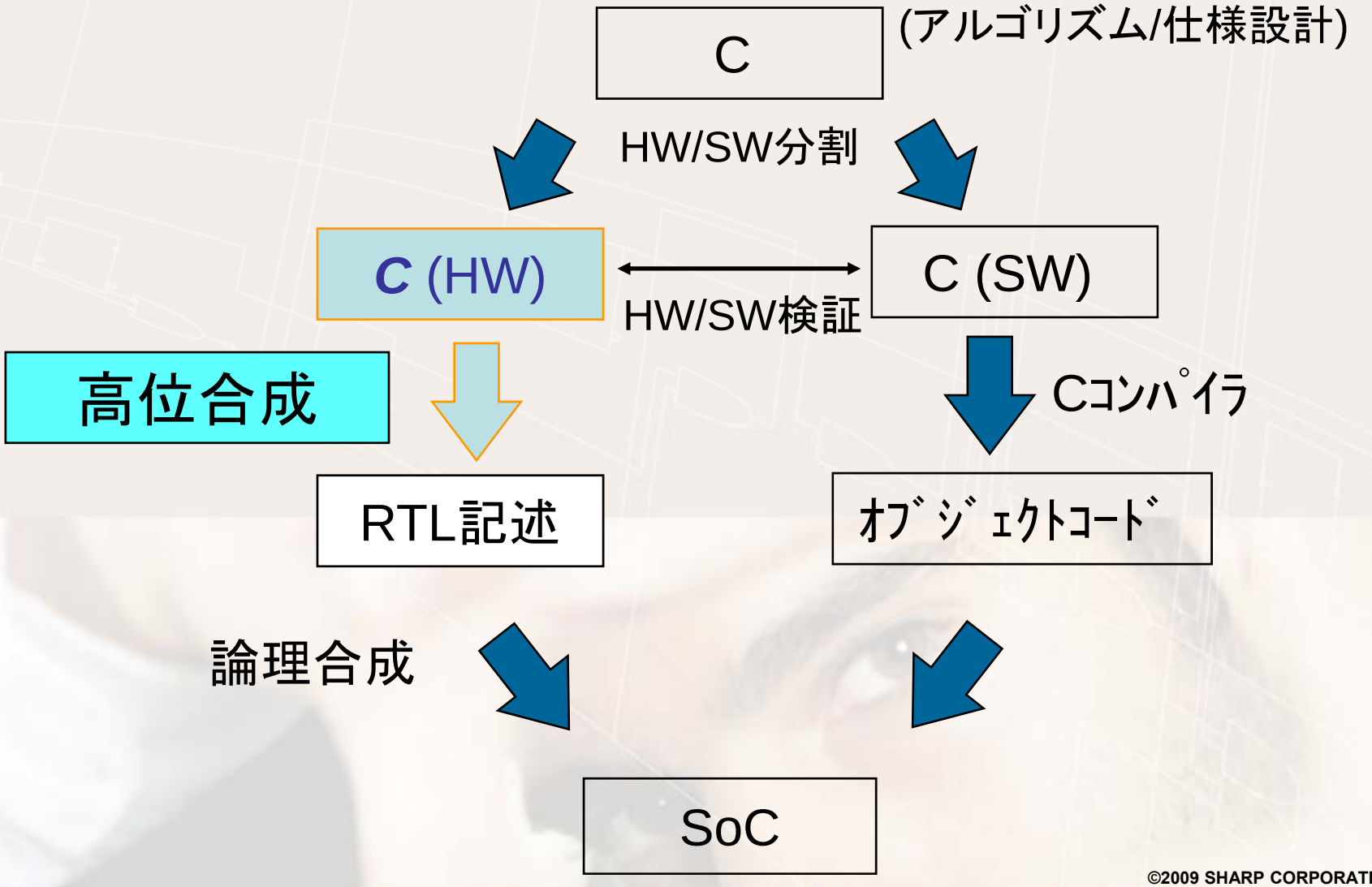
Cベース設計



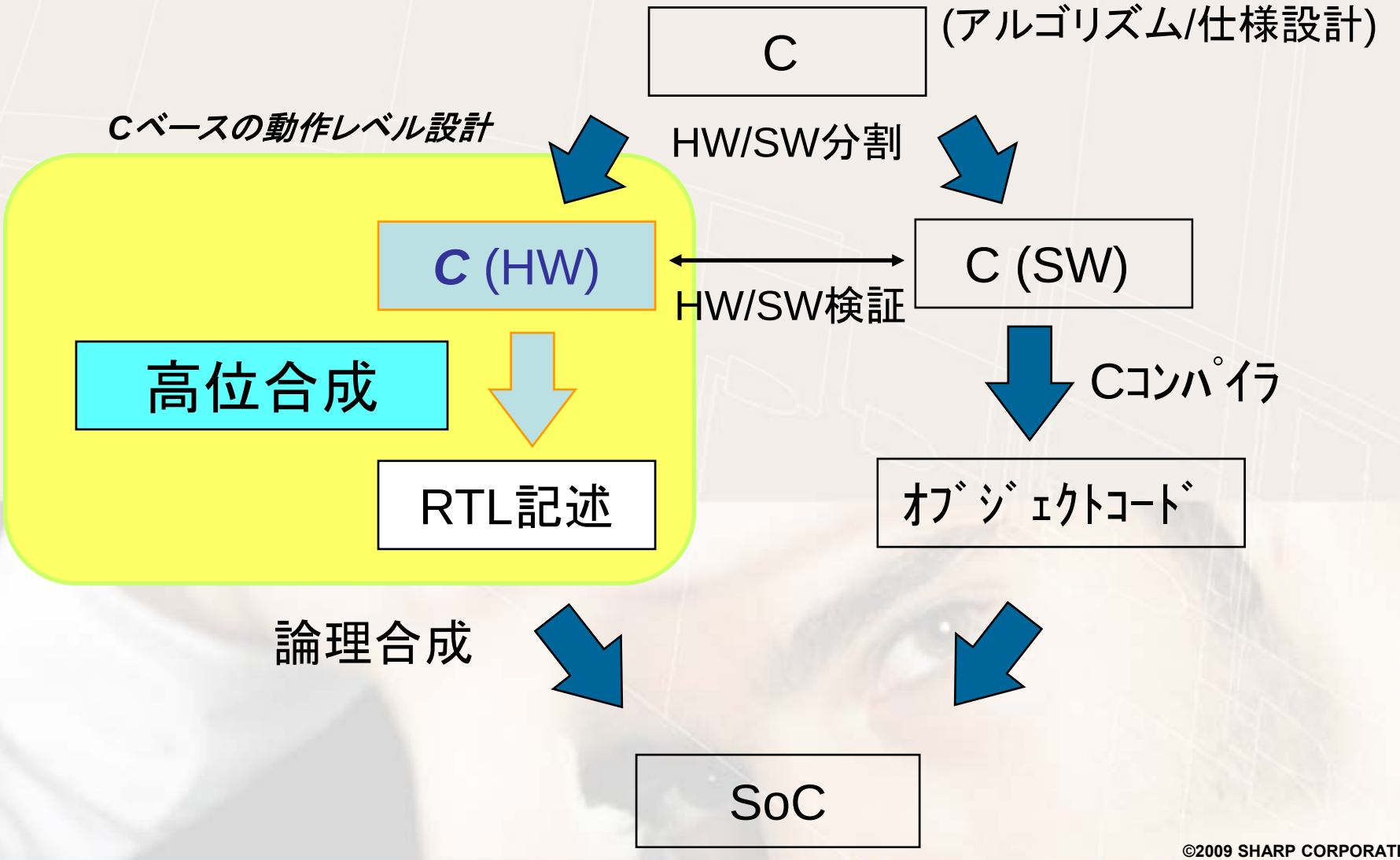
Cベース設計



Cベース設計



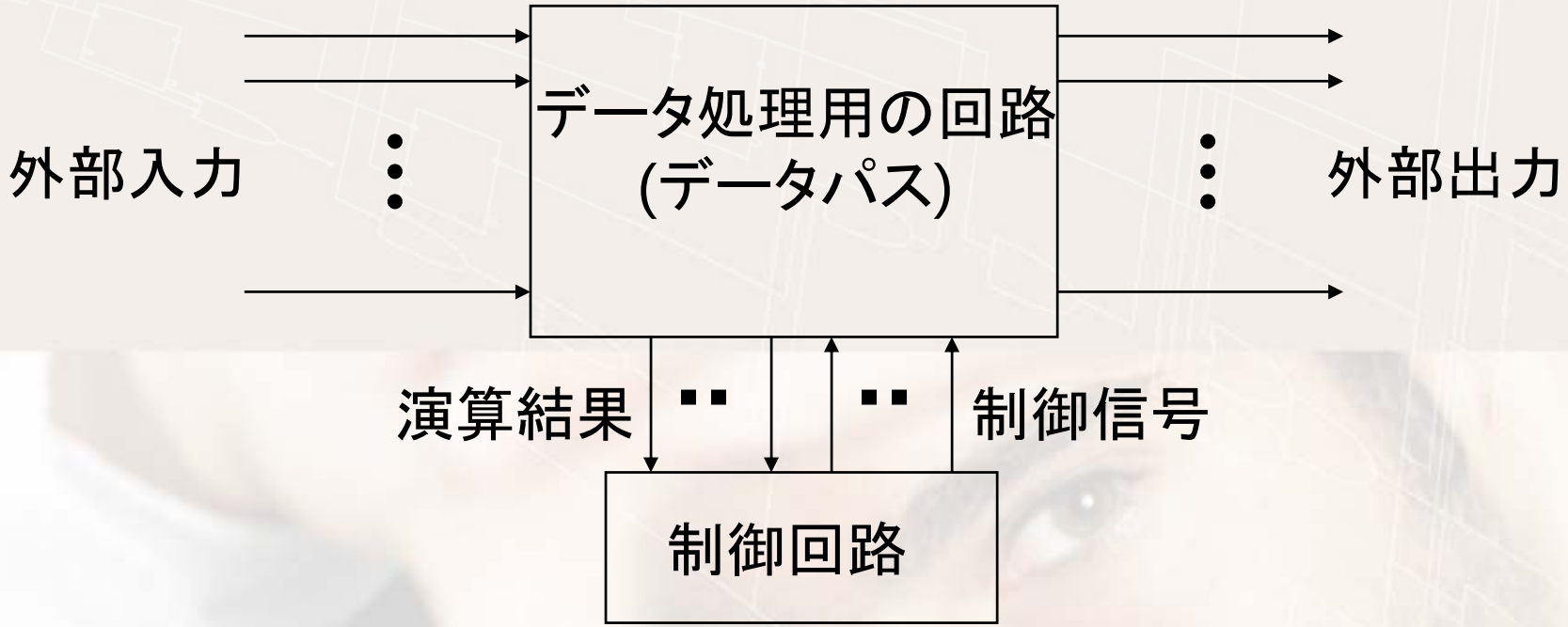
Cベース設計



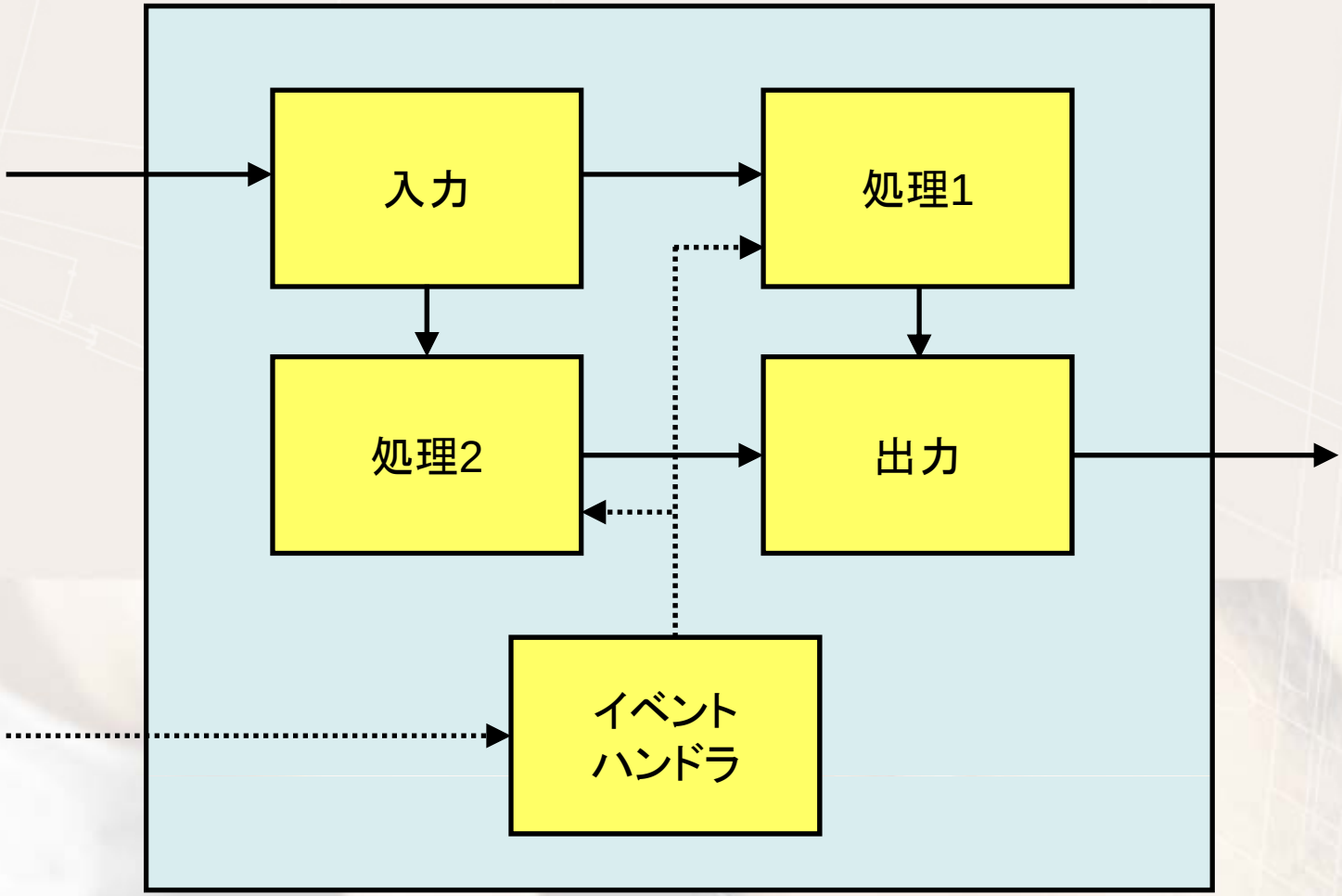
Cベースの動作レベル設計のメリット

- 期待できるメリット
  - 記述量が少なくなる
  - 高速に機能検証が行える
  - 検証した機能がそのまま回路になる
  - 早い段階で回路規模、性能が見積もれる
  - たくさんのCプログラム資産が利用できる

高位合成のターゲット回路

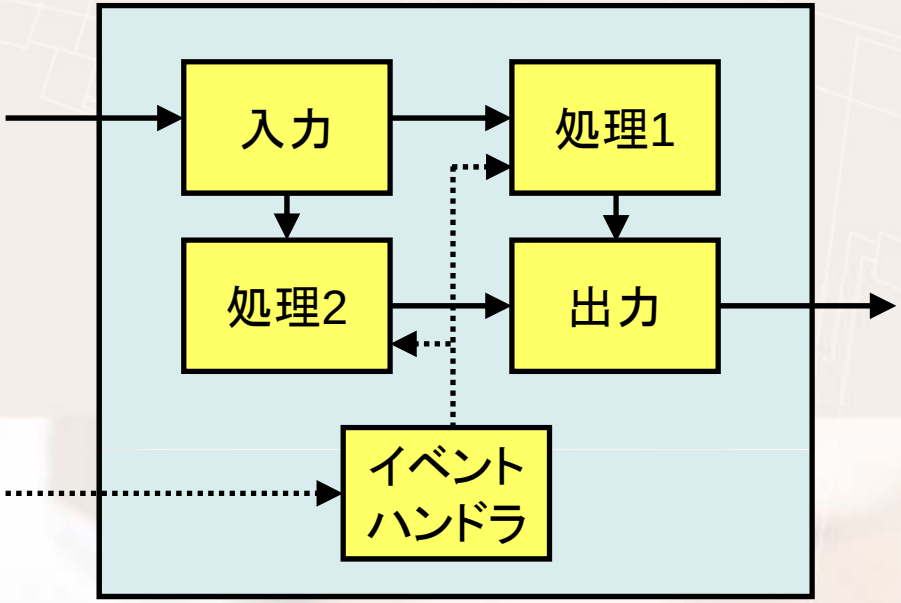


高位合成技術の適用範囲 （1）



疑問：どの部分に高位合成を適用すべきか？

高位合成技術の適用範囲(2)



●各ブロックの処理が簡単な場合

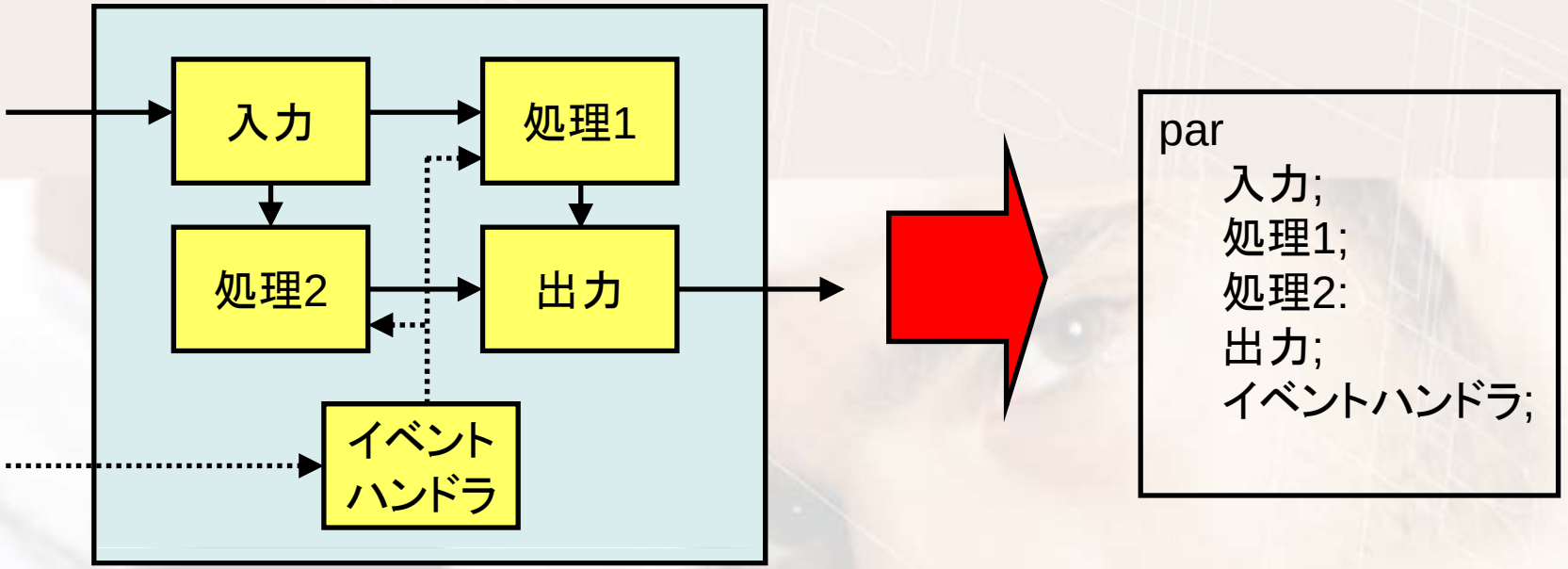
一つの動作で記述して高位合成

●各ブロックの処理が複雑な場合

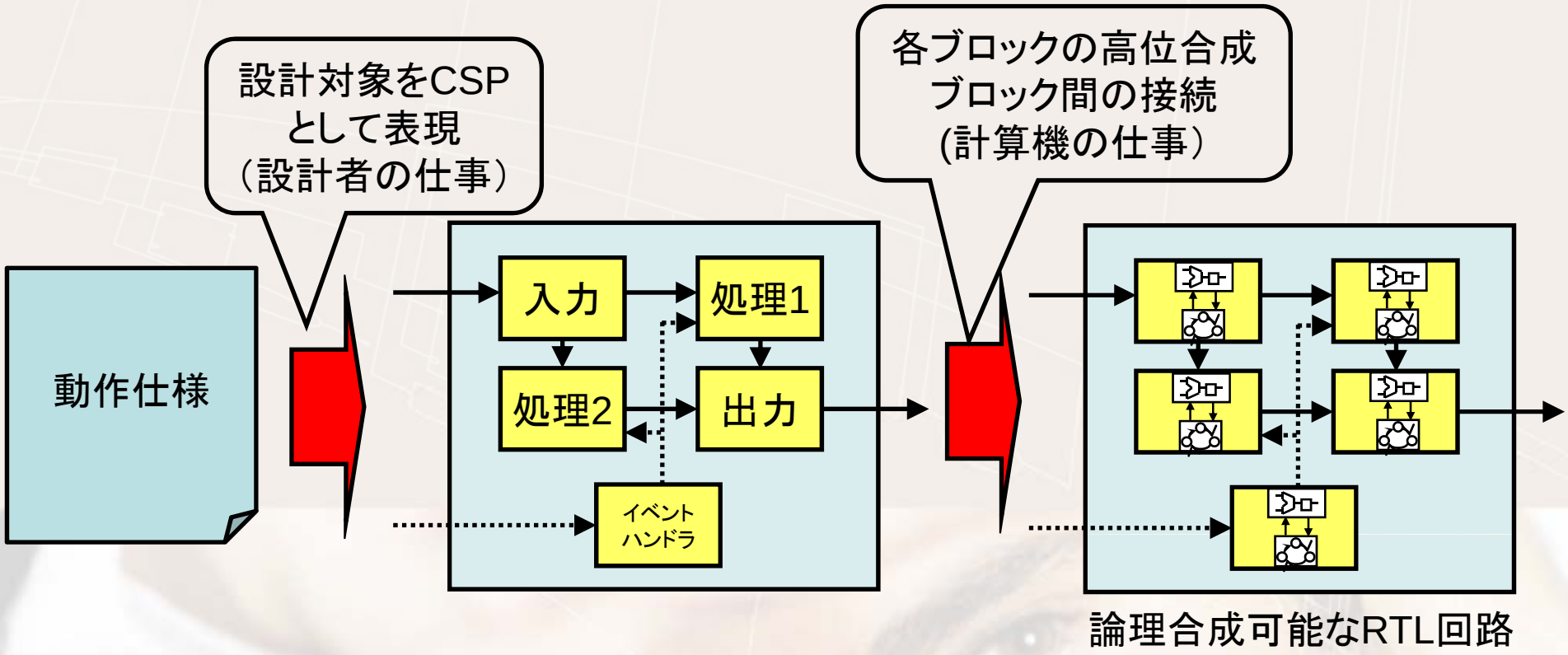
別々のブロックにした方が考えやすい

シャープでの取り組み

- システムの並行性をCSP (Communicating Sequential Processes)として表現
- 各逐次プロセスを高位合成技術で合成

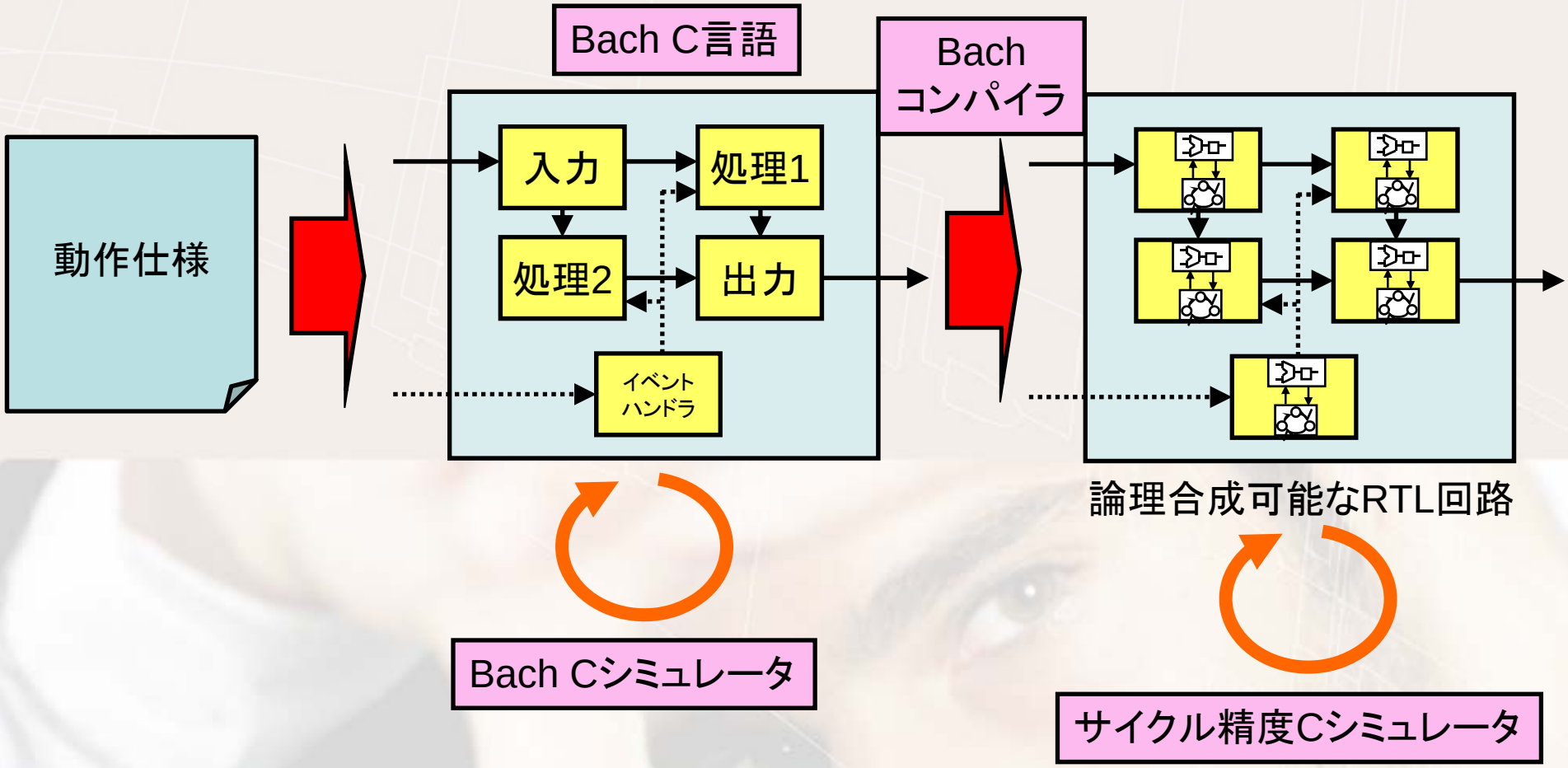


上流設計の考え方

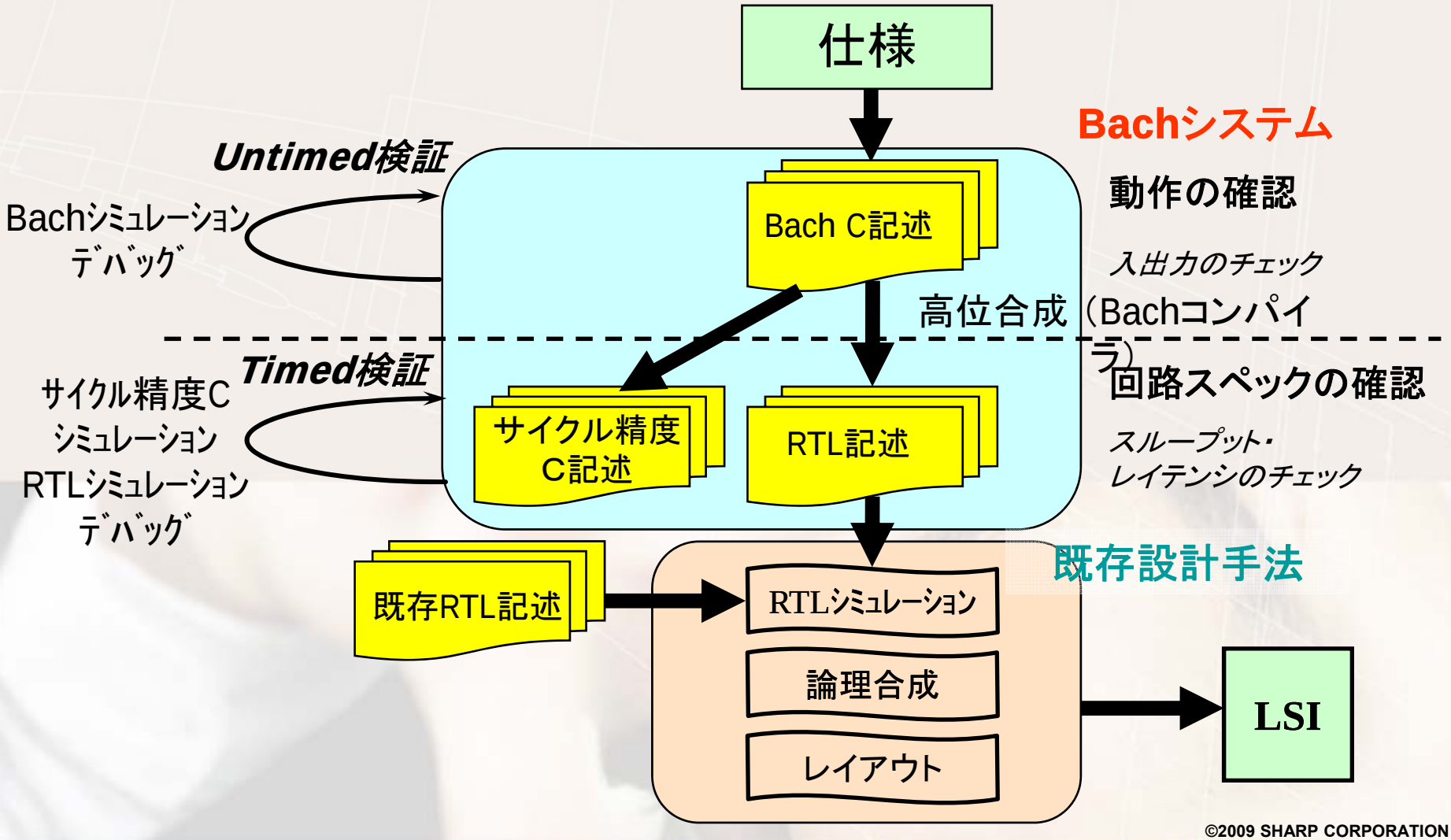


このような上流設計が可能となる設計環境を構築する

Bachシステム



Bachを用いた設計フロー

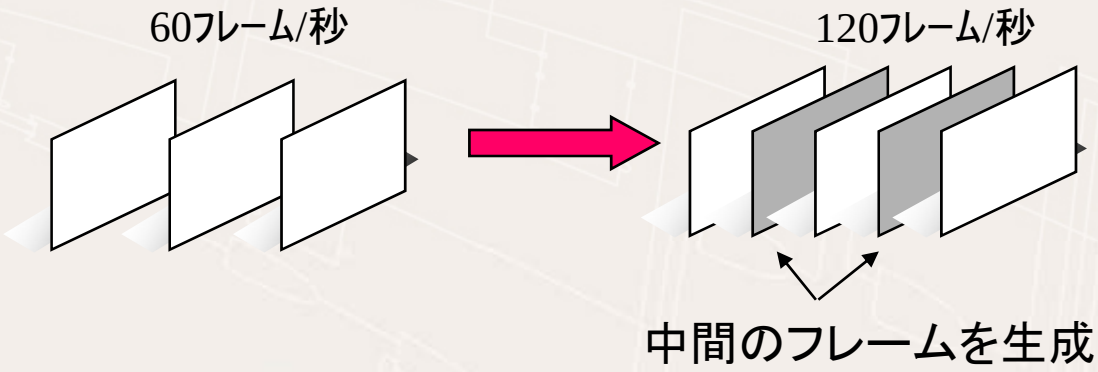


適用事例

- MPEG-4 コーデックLSI
- IrSimpleフォトアダプタ用画像処理エンジン
- CCDカメラ、CMOSセンサカメラ用DSP
- ワンセグ受信用マルチメディア処理LSI

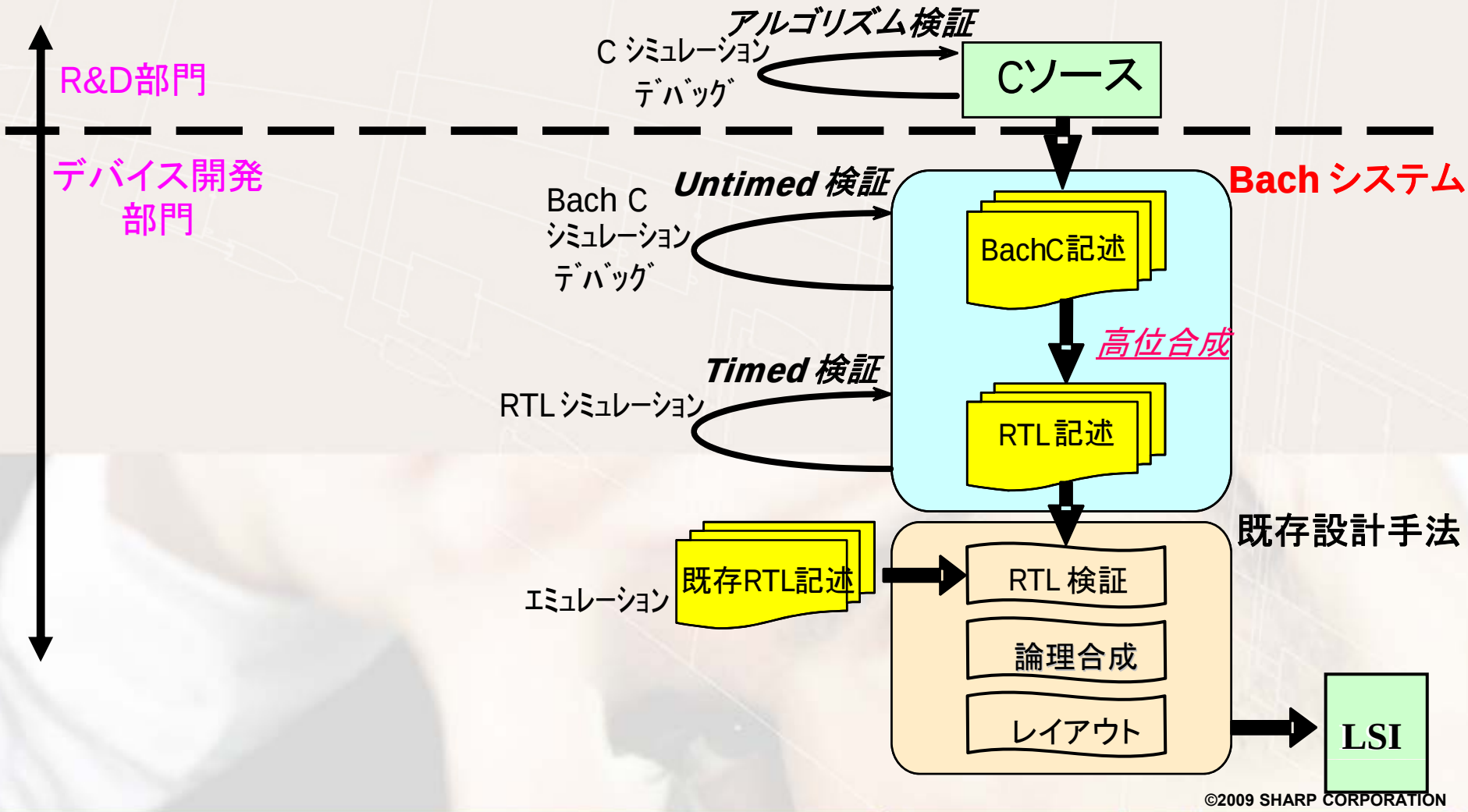
フレームレート変換LSI設計への適用

- 60フレーム/秒の入力動画を120フレーム/秒に変換



- 外部仕様
  - 入力:60 フレーム/秒
  - 出力:120フレーム/秒
  - 画像サイズ : フルスペックHD (1920 x 1080)

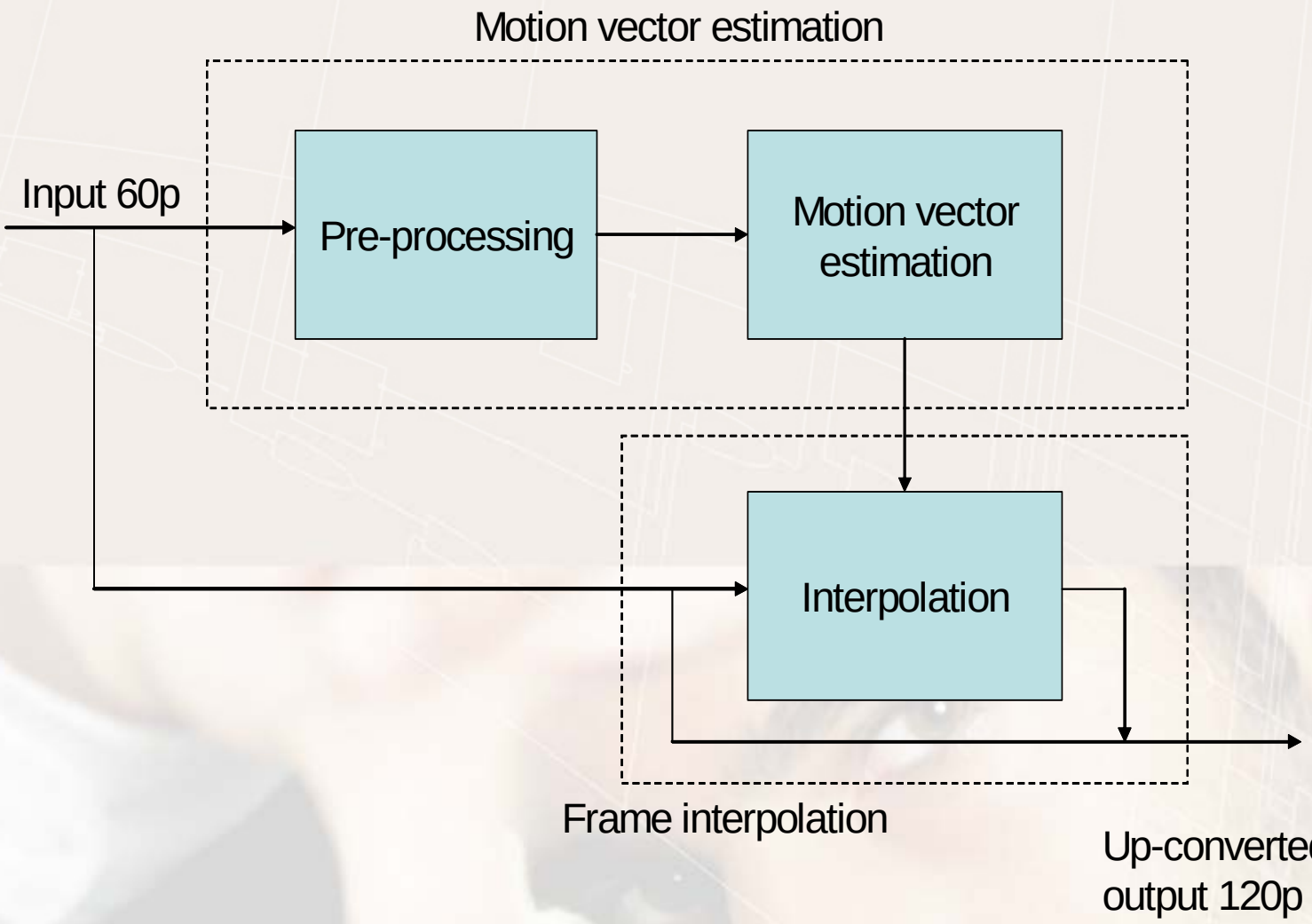
FRC LSIの設計/検証フロー



動作レベル設計（C → Bach C）

1. C言語で設計する部分とそれ以外の切り分け
   - C言語で設計しなかった回路
     - アナログ回路
     - 単相片エッジのクロックで動作しない回路
     - 既設計の回路
     - テスト回路
2. 全体のデータ・制御の流れを決定
   - C言語ベース設計といえどもトップダウンのアプローチは必要

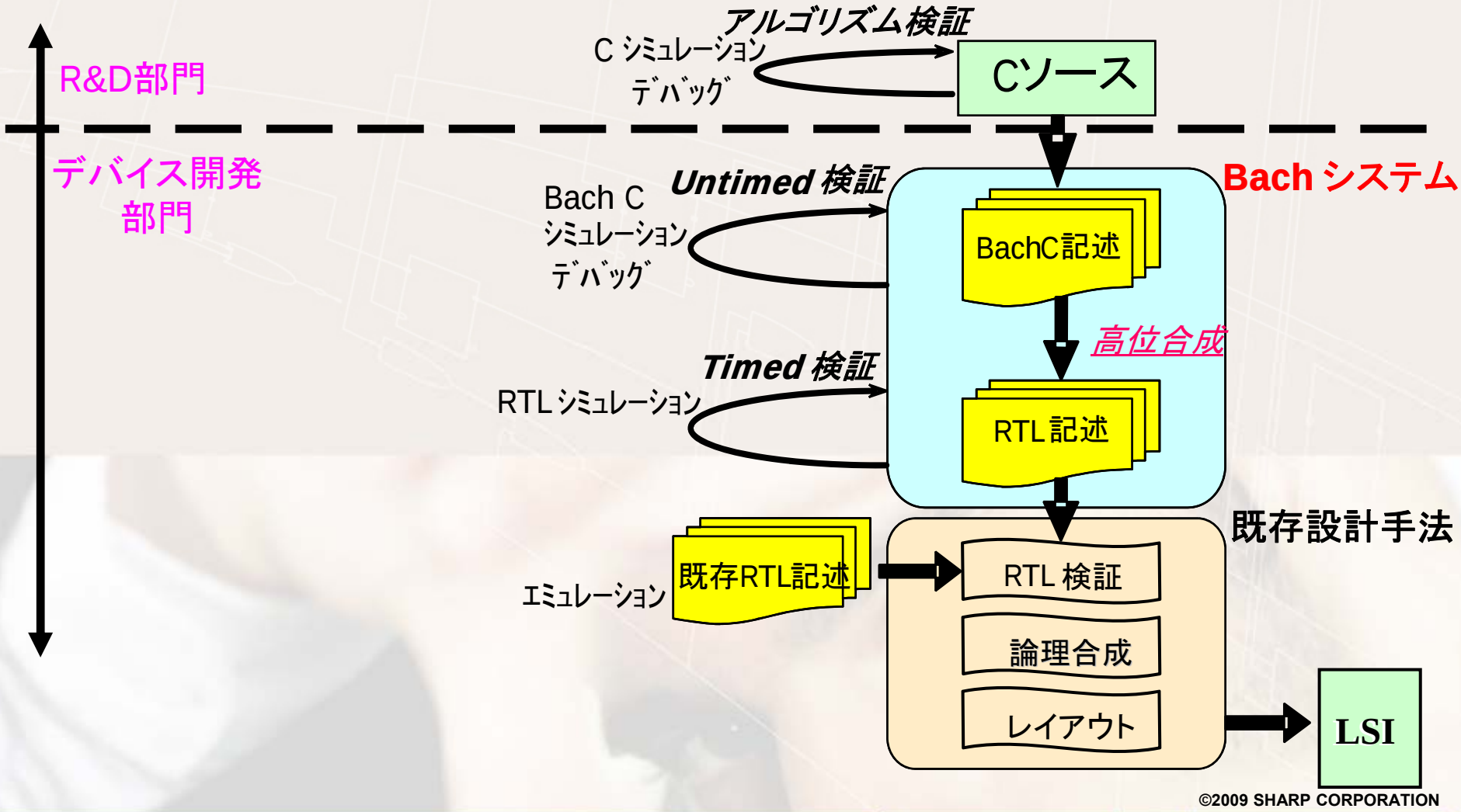
FRCアルゴリズム



動作レベル設計（C → Bach C）

1. C言語で設計する部分とそれ以外の切り分け
   - C言語で設計しなかった回路
     - アナログ回路
     - 単相片エッジのクロックで動作しない回路
     - 既設計の回路
     - テスト回路
2. 全体のデータ・制御の流れを決定
   - C言語ベース設計といえどもトップダウンのアプローチは必要
3. サブモジュールの詳細化
   - 固定小数点化、ビット幅指定、並列化、、、

FRC LSIの設計/検証フロー



Bach C → RTレベル回路

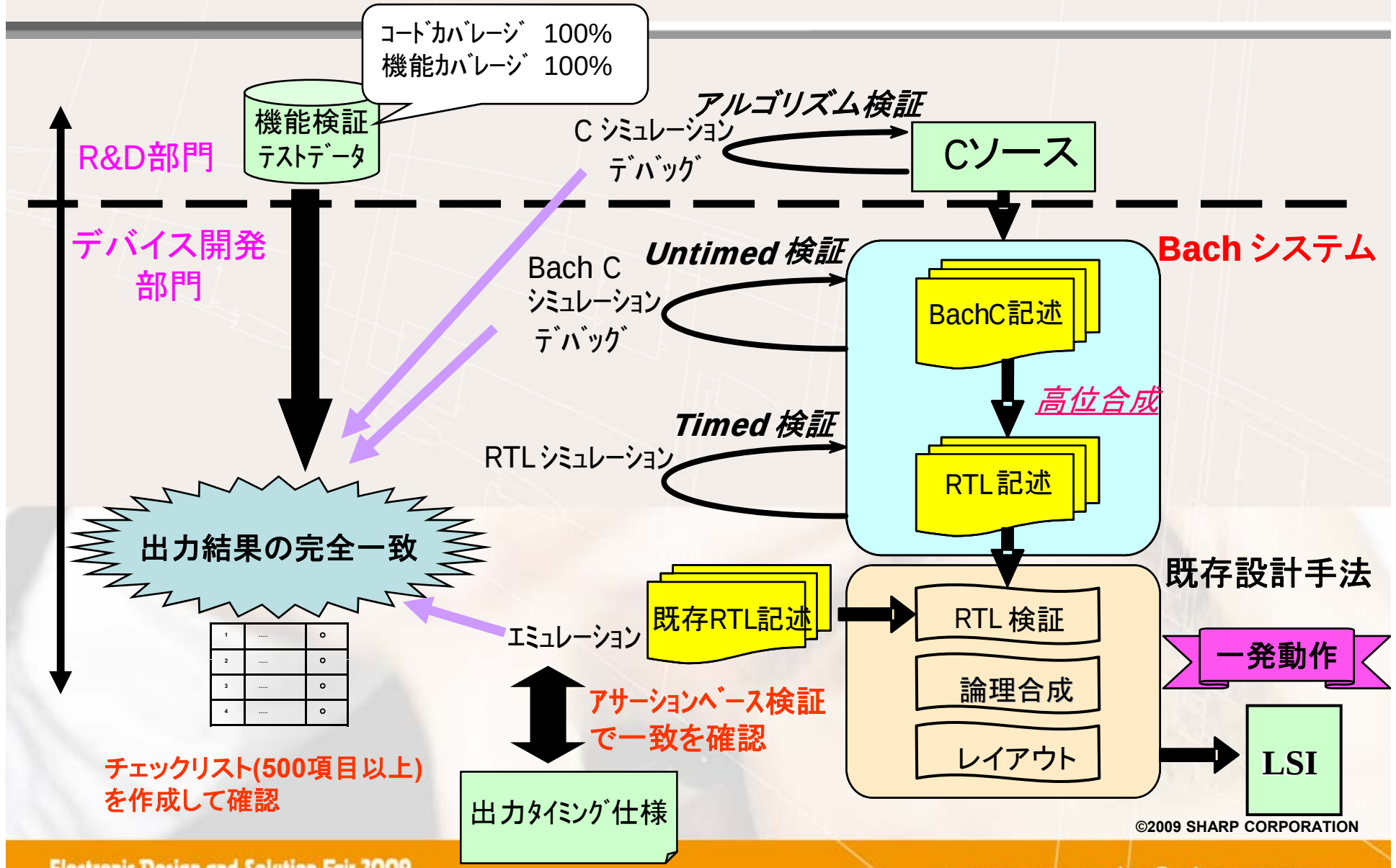
1. Bach Cレベルでuntimedな検証
2. 高位合成
3. 単体でRTレベル検証し、処理サイクルを確認
4. スループット、回路規模のチューニングが必要なら Bach Cコードを修正し1へ。
5. システム全体でRTレベル検証

各工程の検証時間

Original ANSI C	79秒
Bach C	49秒
Emulator	45秒
RTL simulator	228時間 （見積もり）

・入力5フレーム分の検証時間
・CPU: インテル Xeon5160

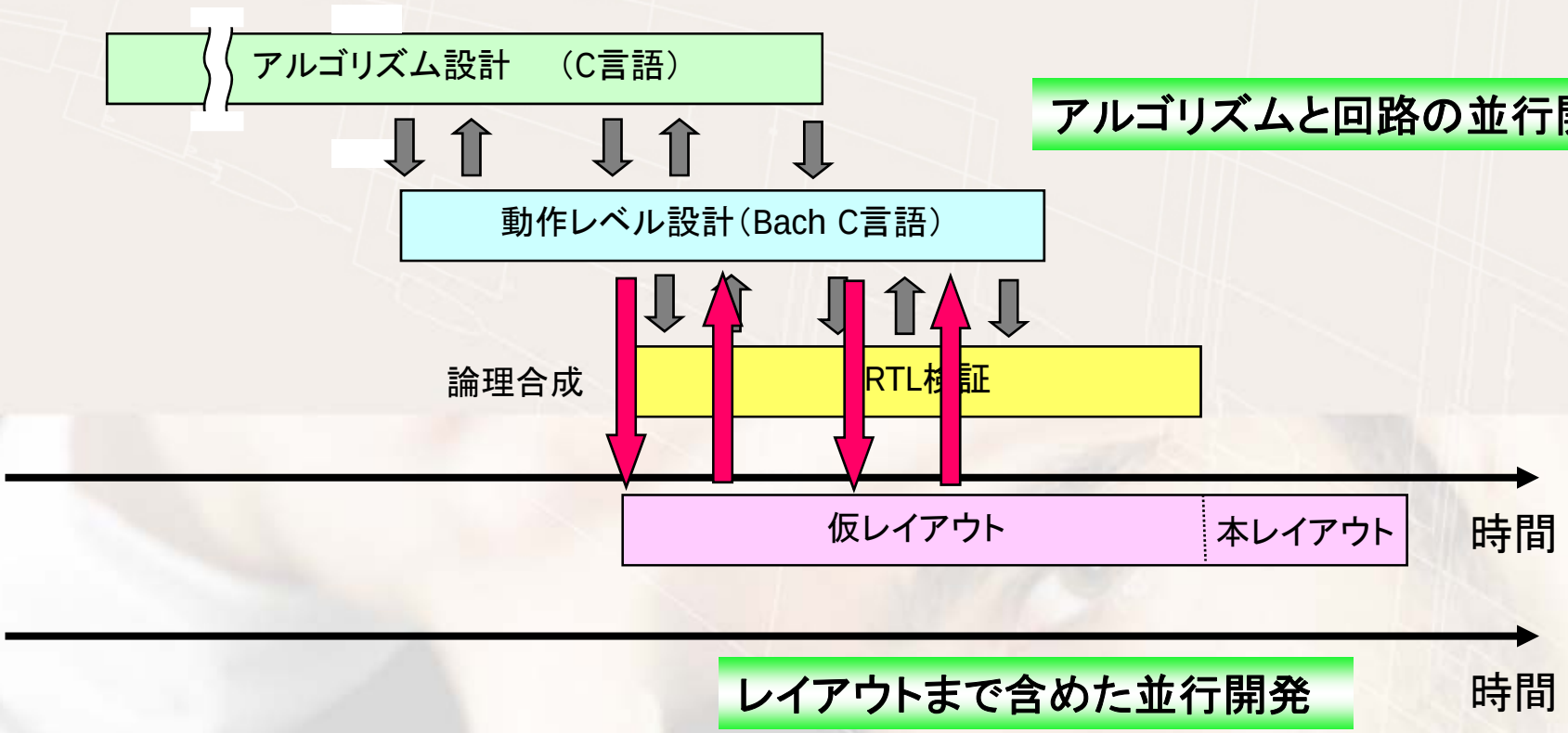
FRC LSIの設計/検証フロー



設計結果

- 回路全体の90%以上をBach C言語で設計
- アルゴリズムがFixしてから約2ヶ月でテープアウト完了

設計工程



アルゴリズムと回路の並行開発

レイアウトまで含めた並行開発

設計結果

- 回路全体の90%以上をBach C言語で設計
- アルゴリズムがFixしてから約2ヶ月でテープアウト完了
- リスピンなしで量産化
- 当社製アクオスに搭載し、
  フレームレート変換機能付きフル
  スペックHD液晶テレビを
  「世界で初めて」商品化。

C+高位合成を使った設計の課題

- タイミング収束
  - 高位合成時の見積り精度
- 静的検証
  - 動作レベルとRTレベル等価性検証
- ECOへの対応
  - 抽象度と解析容易性のトレードオフ
- 記述スタイルの確立

C+高位合成を使った設計での注意点

- すべての回路をC言語で設計するのが良いとは限らない
- 良い合成ツールがあれば良い回路ができるわけではない
- C言語を使えば高速に検証できるわけではない