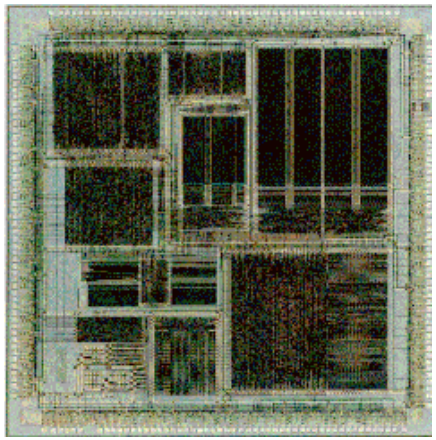
システムLSI設計における形式的手法への期待



安浦寛人

九州大学システムLSI研究センター

2002. 2. 26

システムLSI設計における形式的手法への期待

1．システムLSIとは？

2．システムLSIの設計の流れ

3．形式的検証

4．上流設計と形式的手法

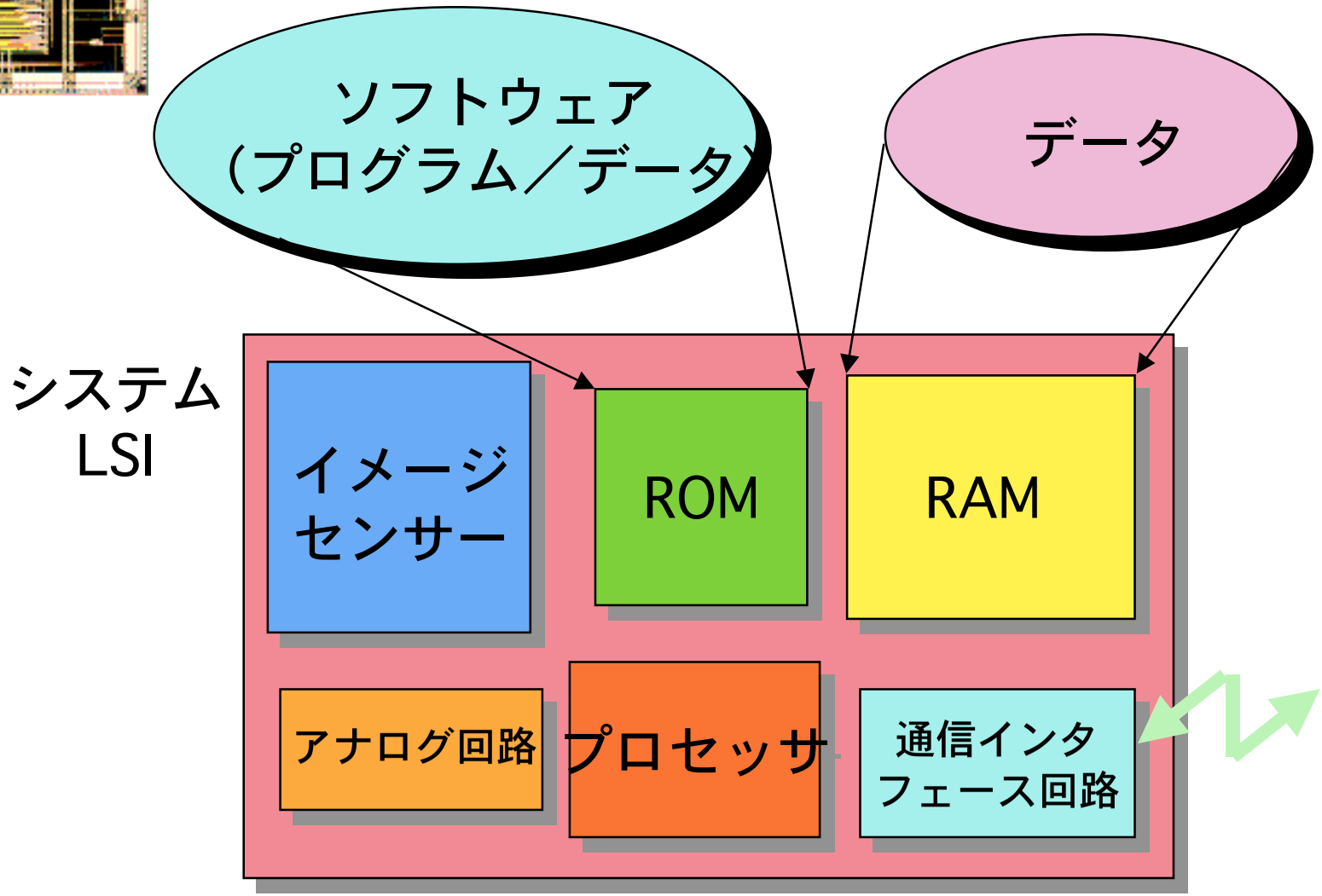
5．システムLSIの経済学

6．Quality, Reliability and Security

システムLSI とは？

- 大規模集積回路　Very Large Scale Integration
  - 1cm^{2}程度のシリコンチップ上に数百万から数億トランジスタを集積
  - 種々の回路を集積
    - デジタル論理回路
      - CPU、演算回路、デジタル信号処理回路、インタフェース回路
    - メモリ回路
      - RAM、ROM、フラッシュメモリ
    - アナログ回路
      - オペアンプ、A･D変換、D･A変換、PLL、入出力回路
    - センサー
      - CCD、フォトダイオード、ガスセンサ

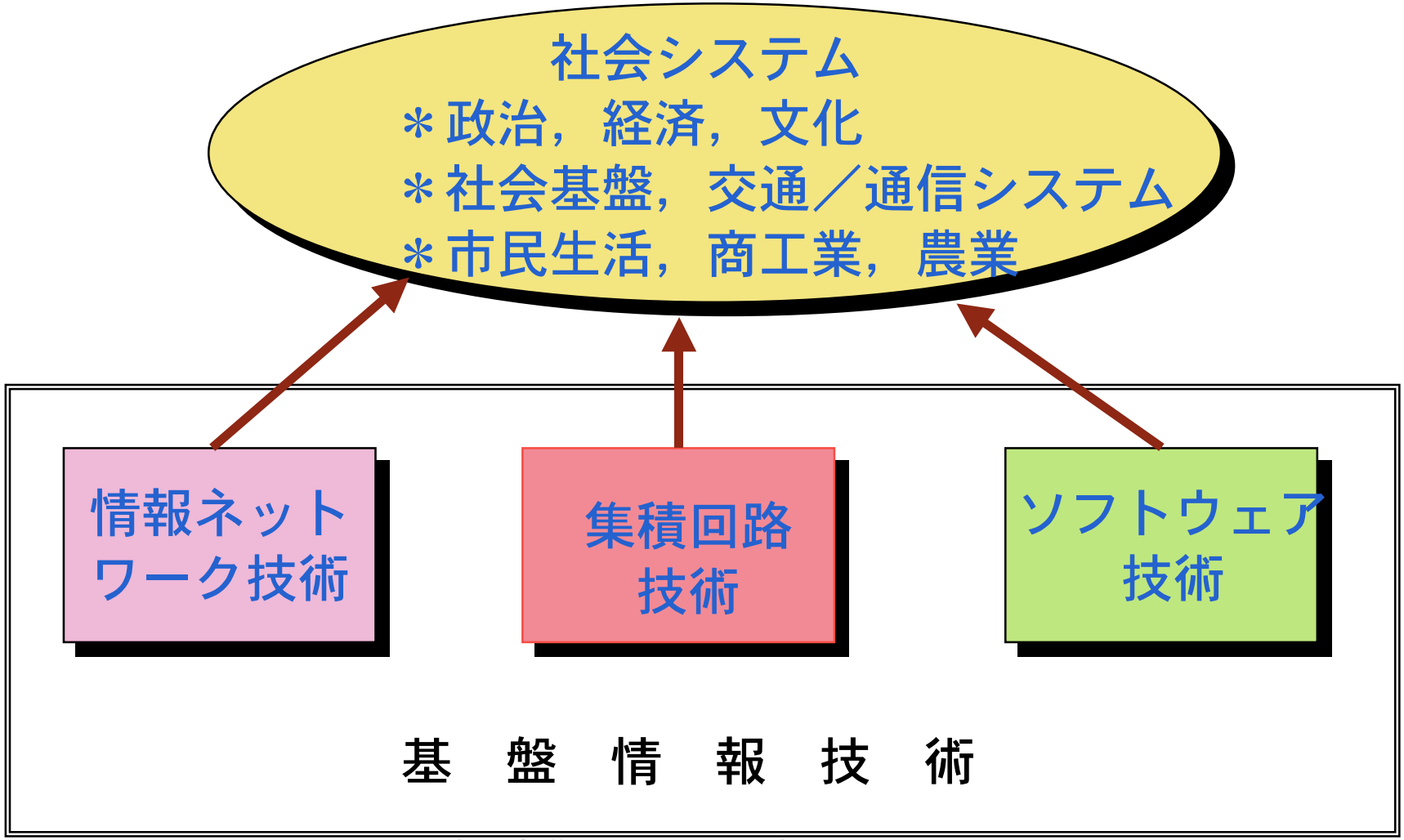
システムLSIの構成


ソフトウェア
（プログラム／データ）
データ
システム
LSI
イメージ
センサー
ROM
RAM
アナログ回路
プロセッサ
通信インタ
フェース回路

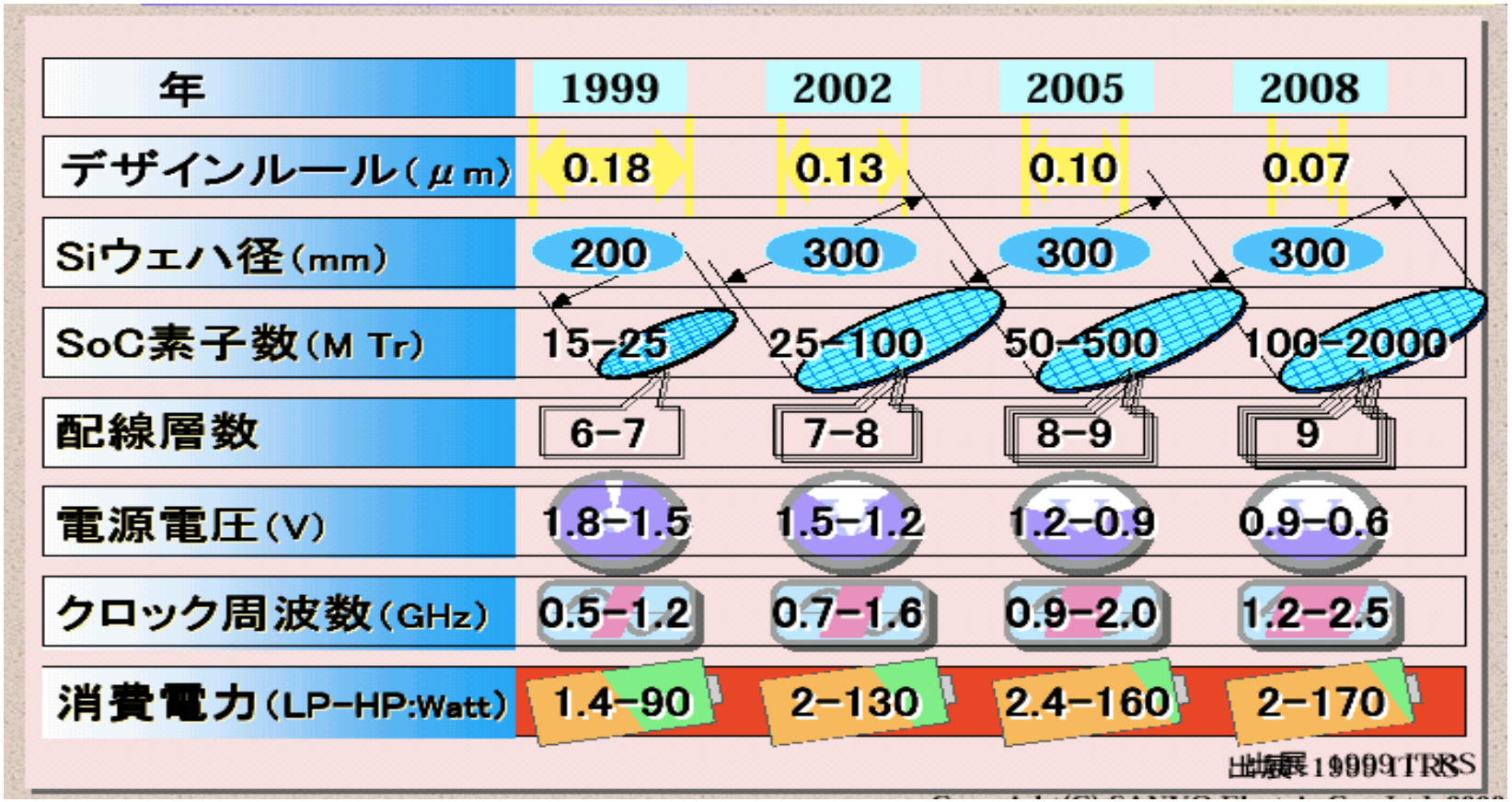
システムLSIの応用分野

- パソコン／ワークステーション／大型計算機／スーパーコンピュータ
- ファミコン／ゲーム機／玩具
- ワープロ／コピー機／プリンタ／電卓／携帯情報機器
- 携帯電話／留守番電話／ファックス
- カメラ／ビデオ／オーディオ／テレビ
- 家電製品（洗濯機，炊飯器，掃除機．．．）
- 自動販売機／自動改札機／自動預金支払機／ロボット
- 自動車／電車／飛行機／船／スペースシャトル
- 役所，会社，駅，航空会社などのシステム，インターネット
- 電子マネー／電子投票／通信・エネルギーシステム

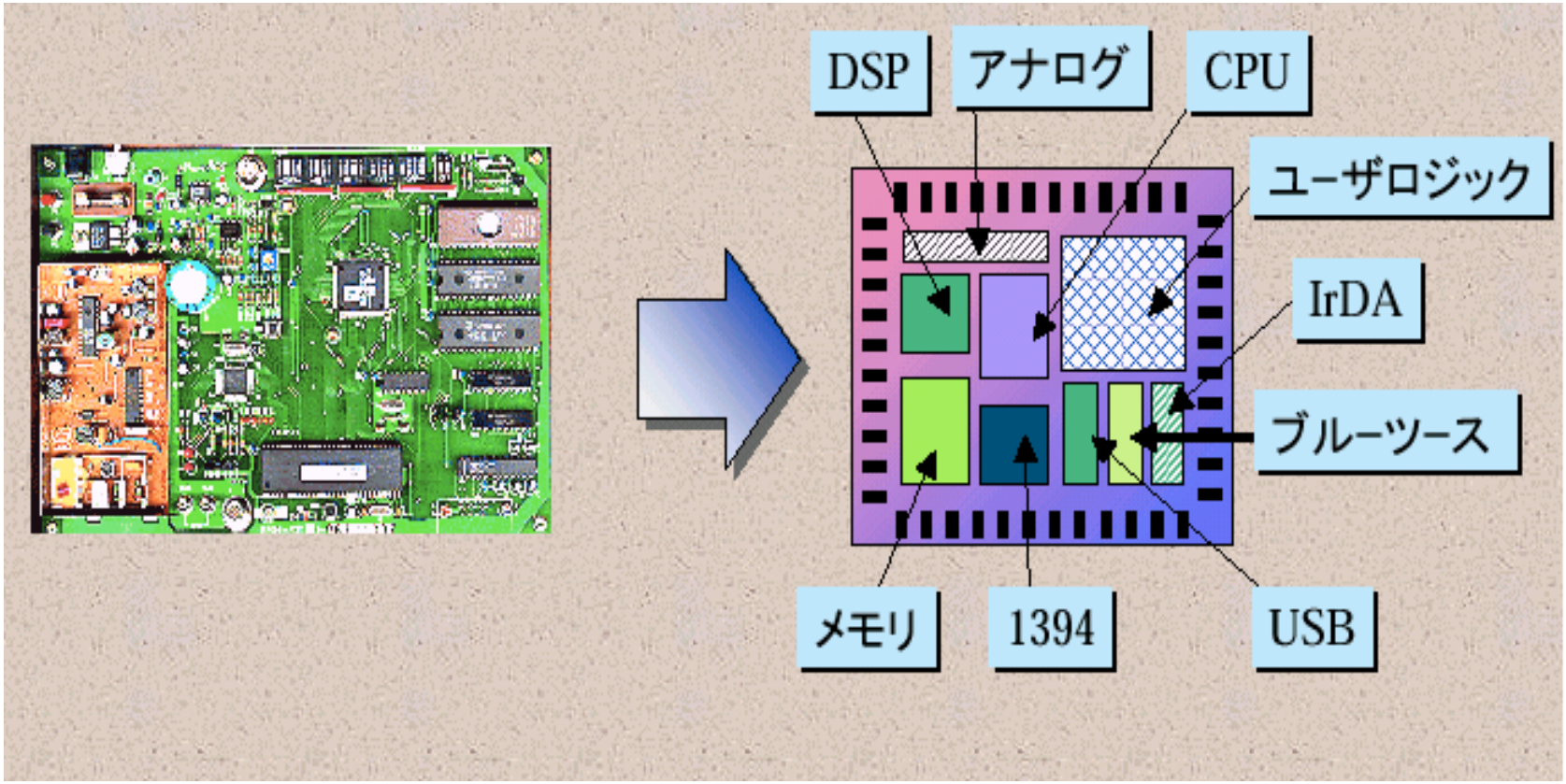
情報技術と社会


社会システム
＊政治，経済，文化
＊社会基盤，交通／通信システム
＊市民生活，商工業，農業
情報ネットワーク技術
集積回路技術
ソフトウェア技術
基　盤　情　報　技　術

プロセス技術の今後の進歩



部品からシステムへ



システムLSI設計における形式的手法への期待

1．システムLSIとは？

2．システムLSIの設計の流れ

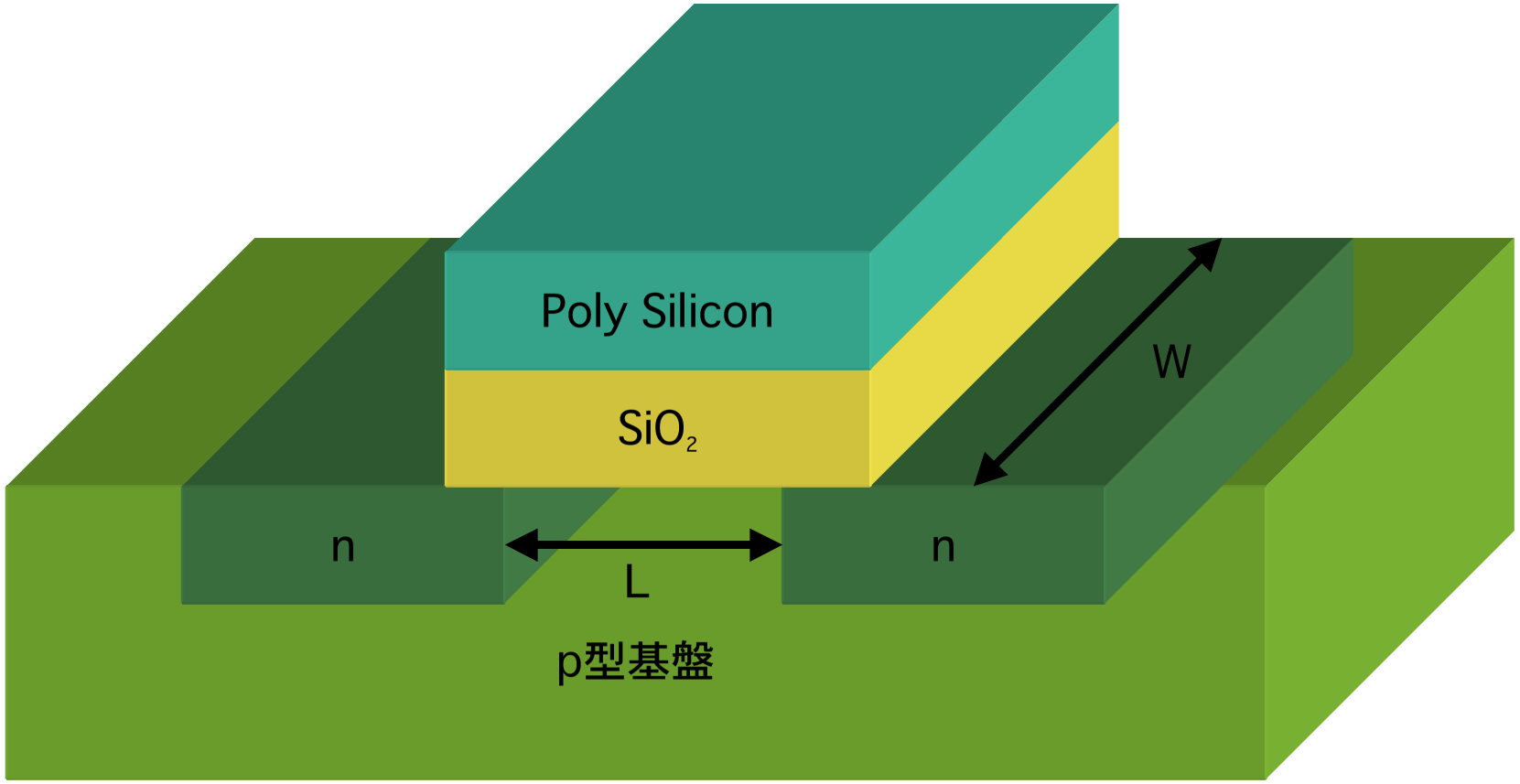
3．形式的検証

4．上流設計と形式的手法

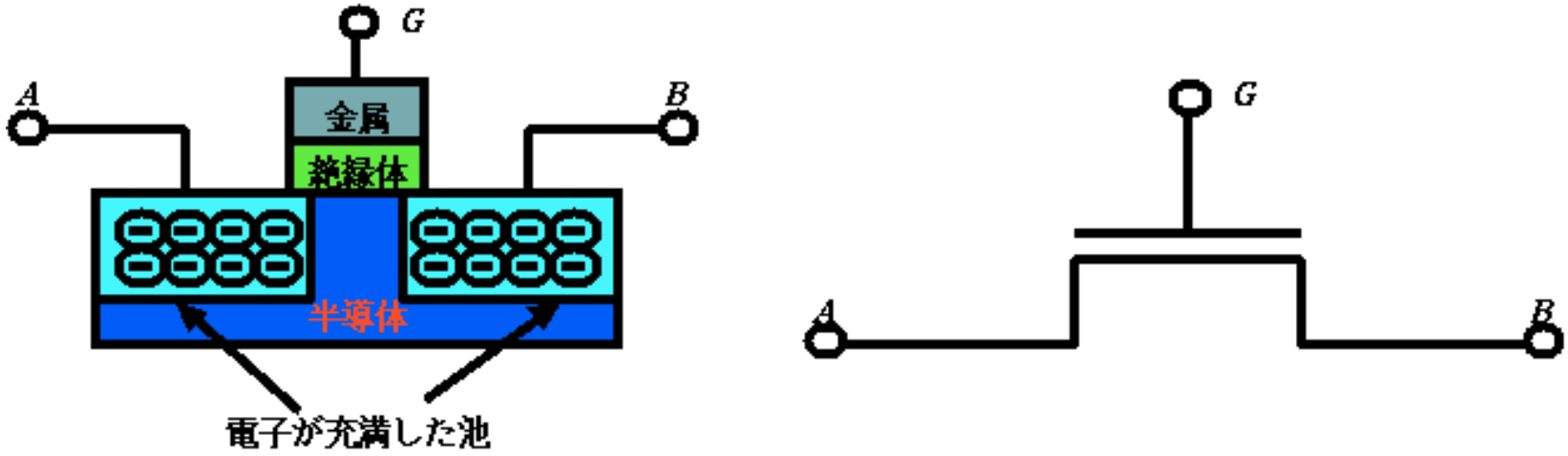
5．システムLSIの経済学

6．Quality, Reliability and Security

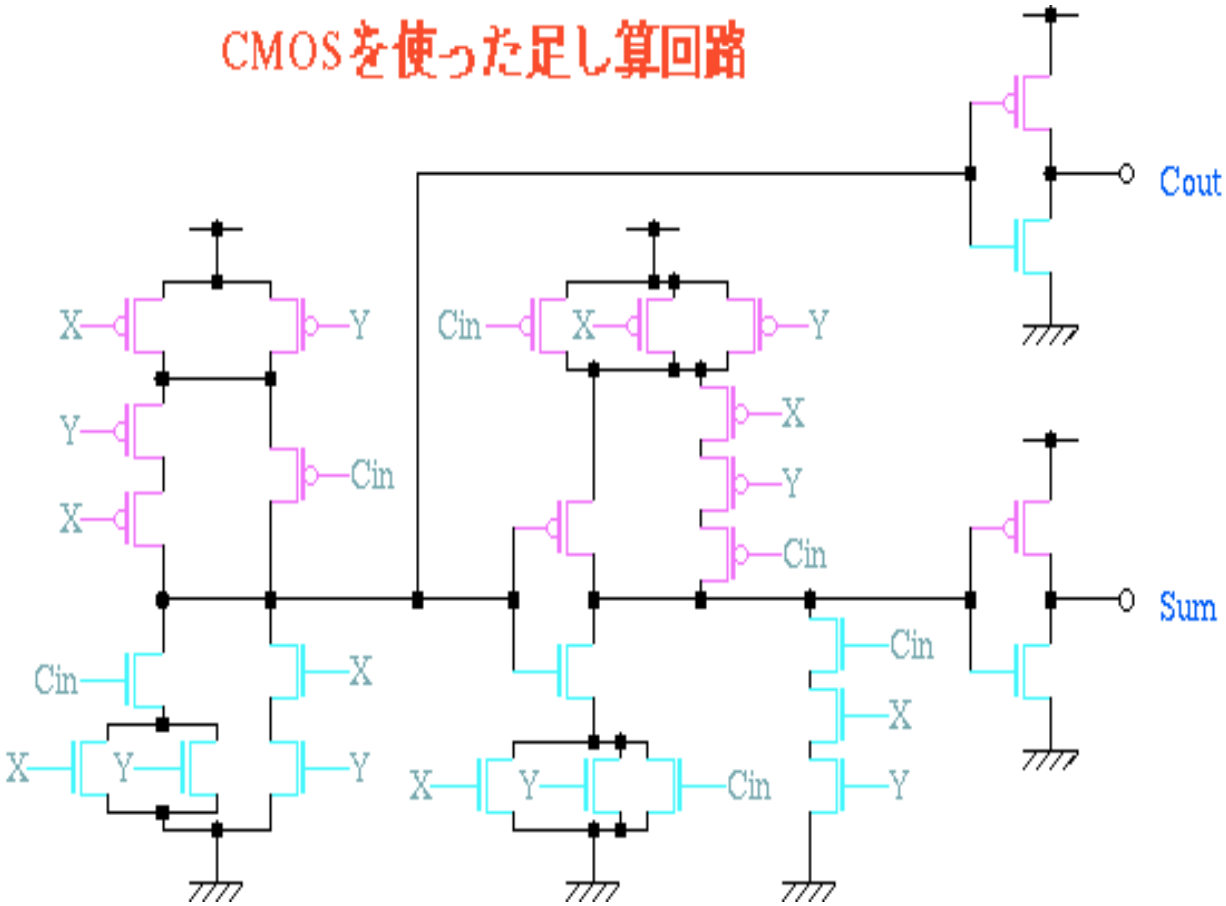
トランジスタの構造



MOSトランジスタ

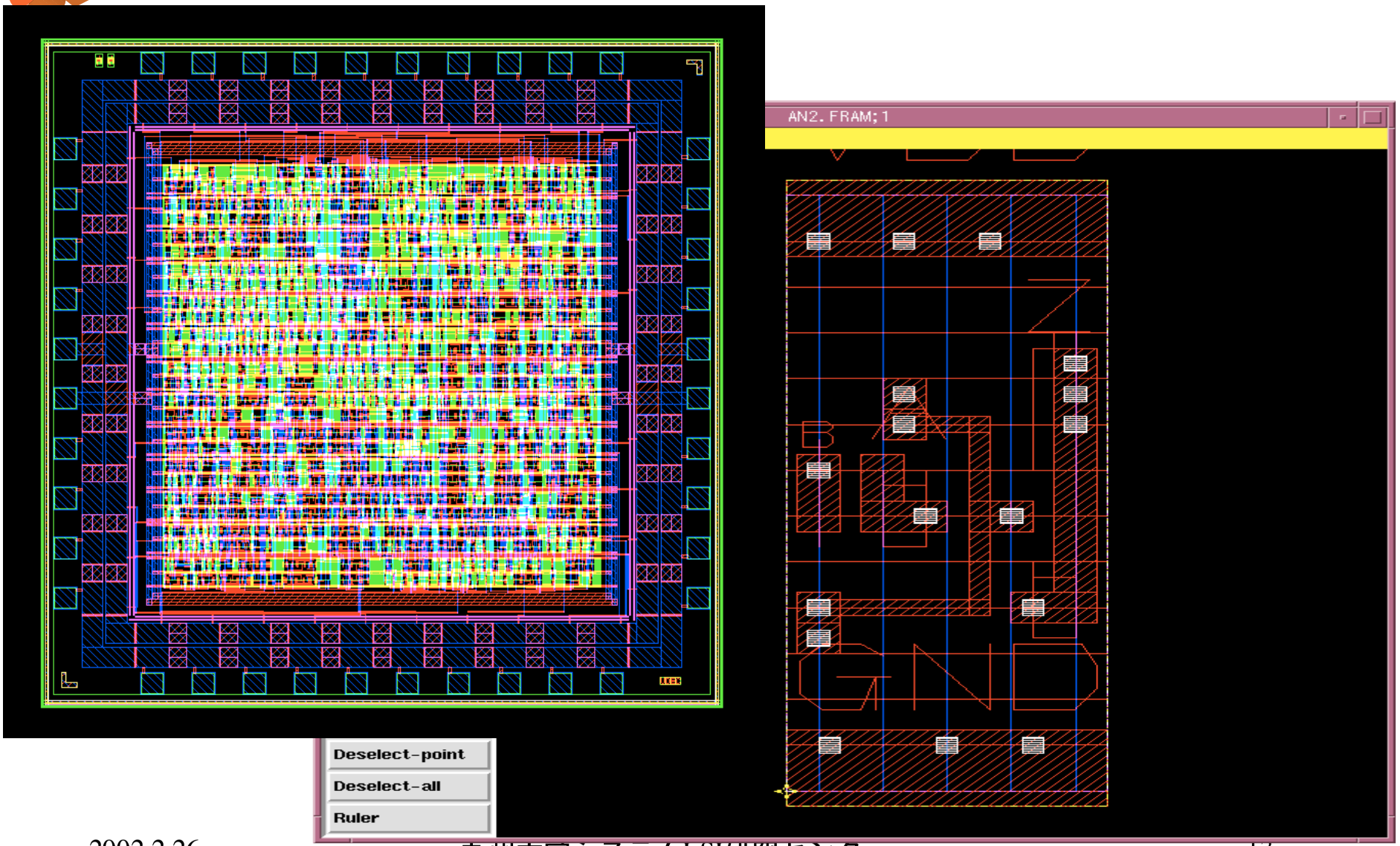


CMOS論理回路

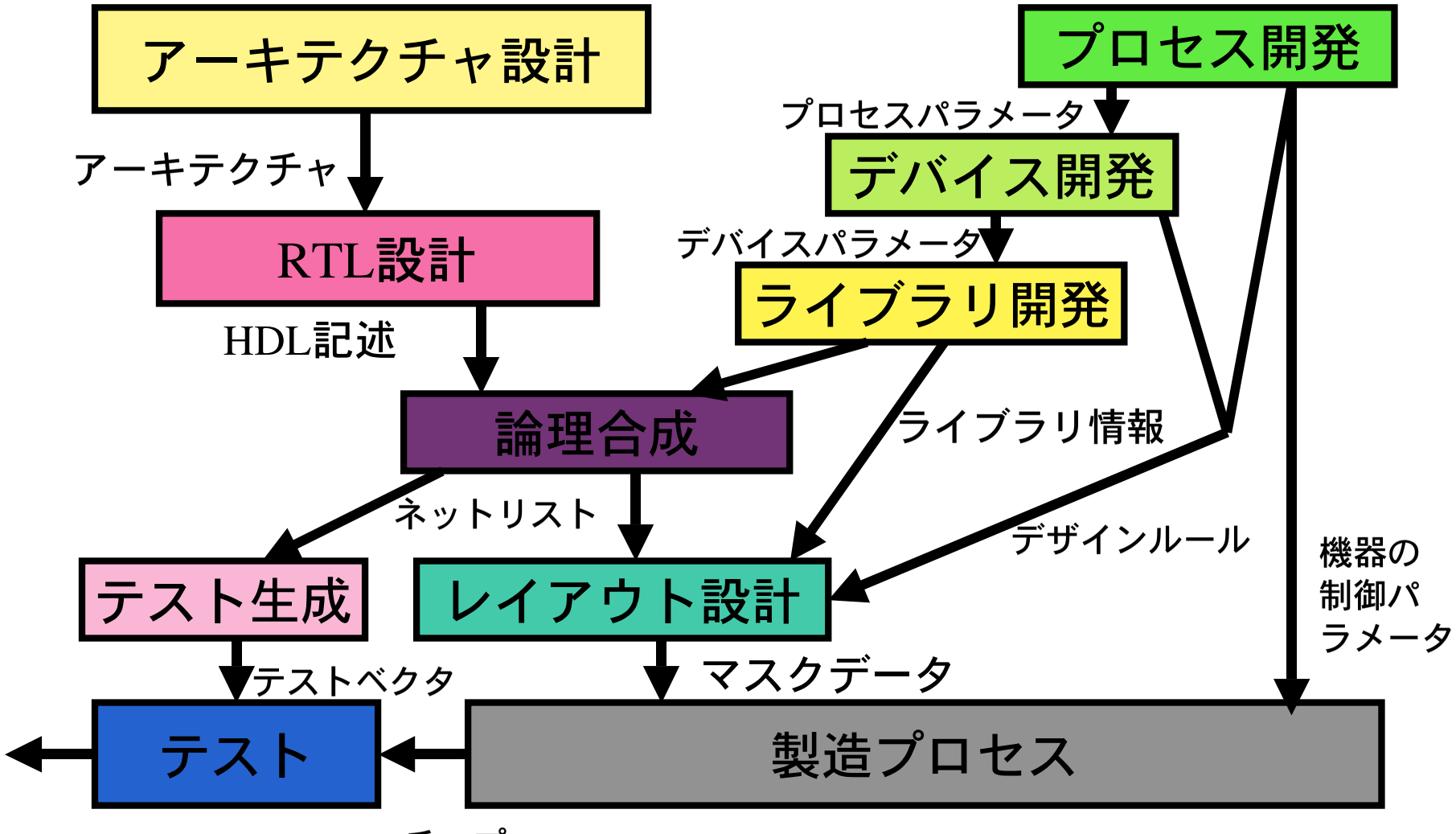


入力			出力	
X	Y	Cin	Cout	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

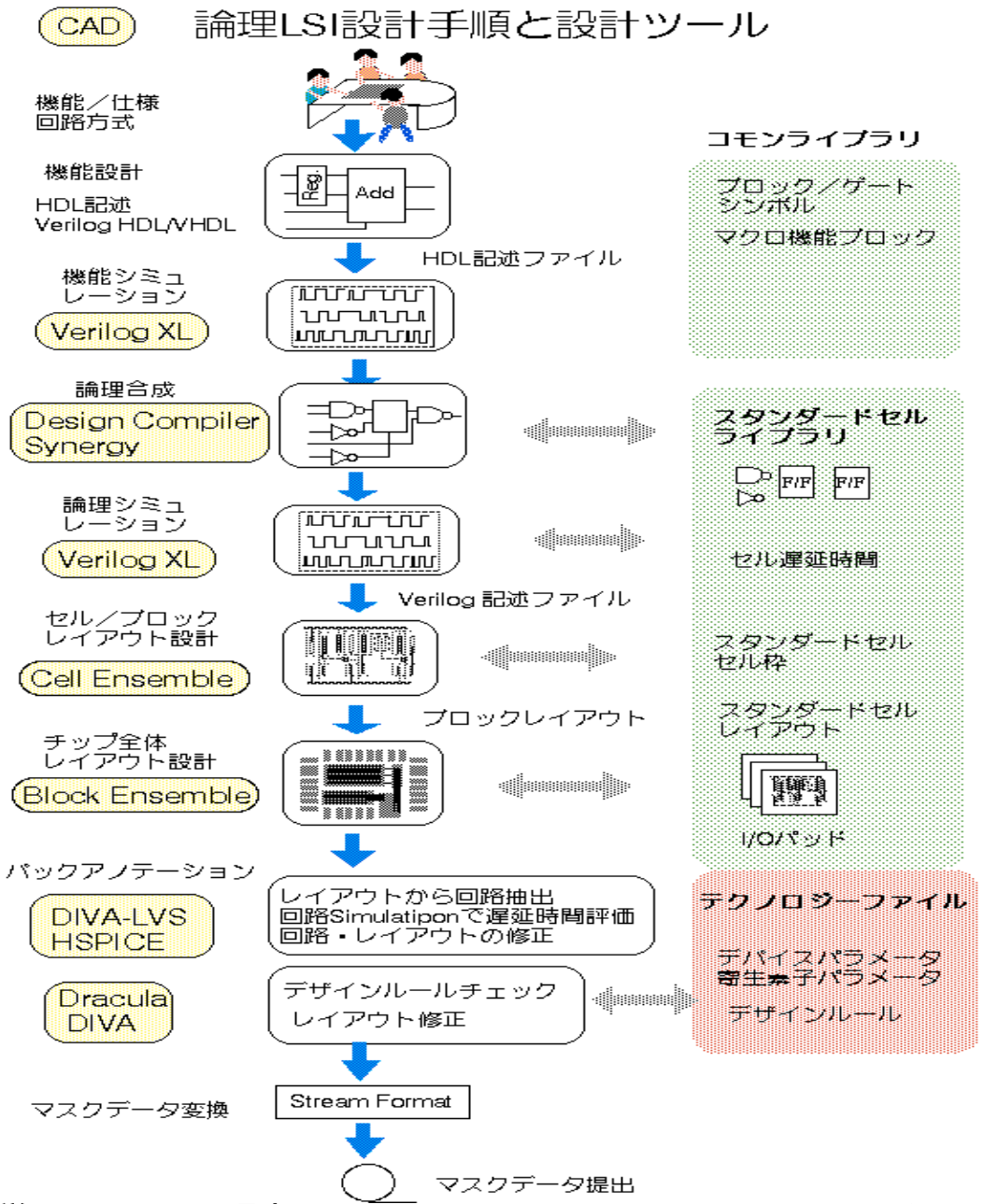
LSIの設計図

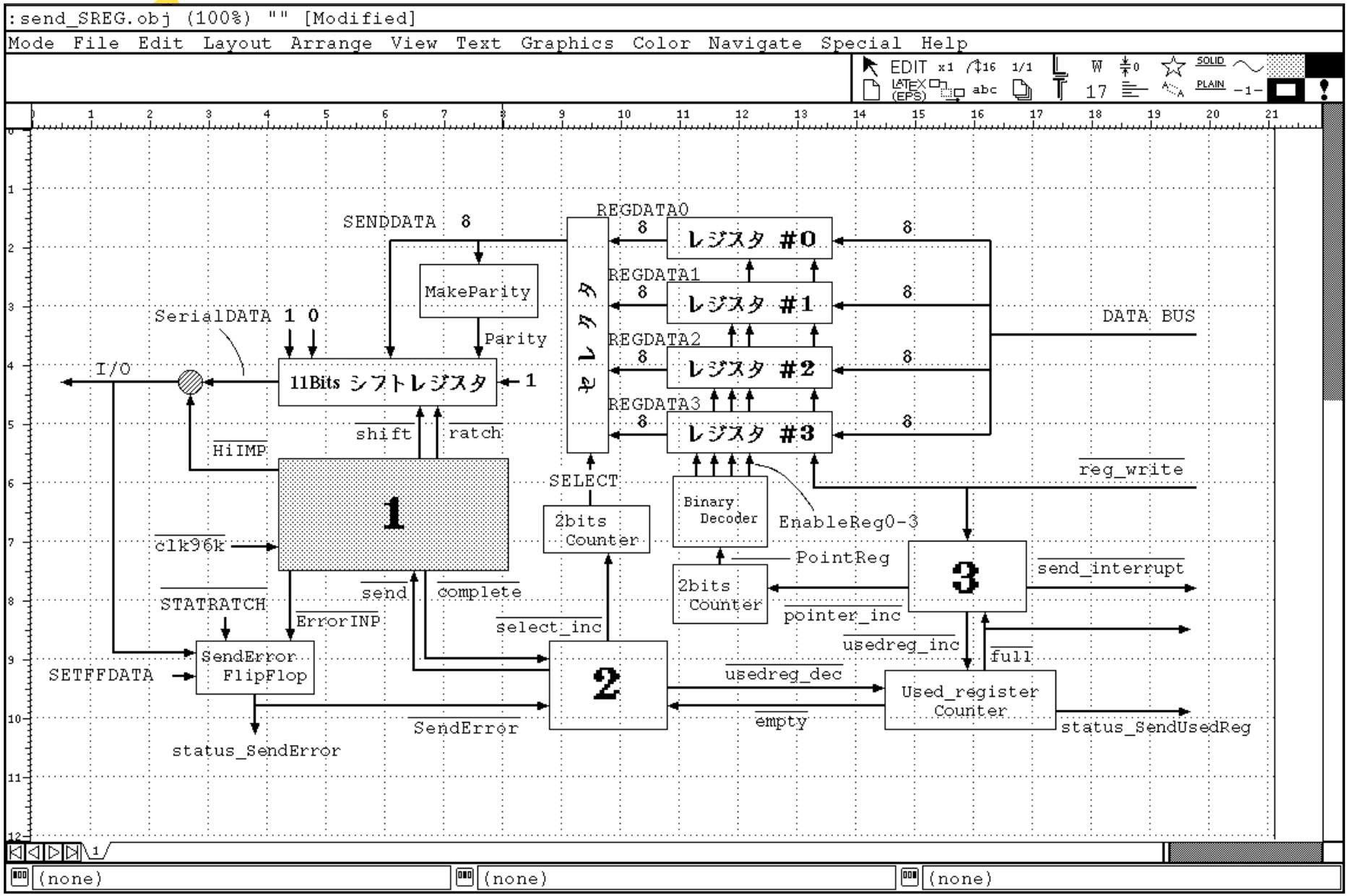


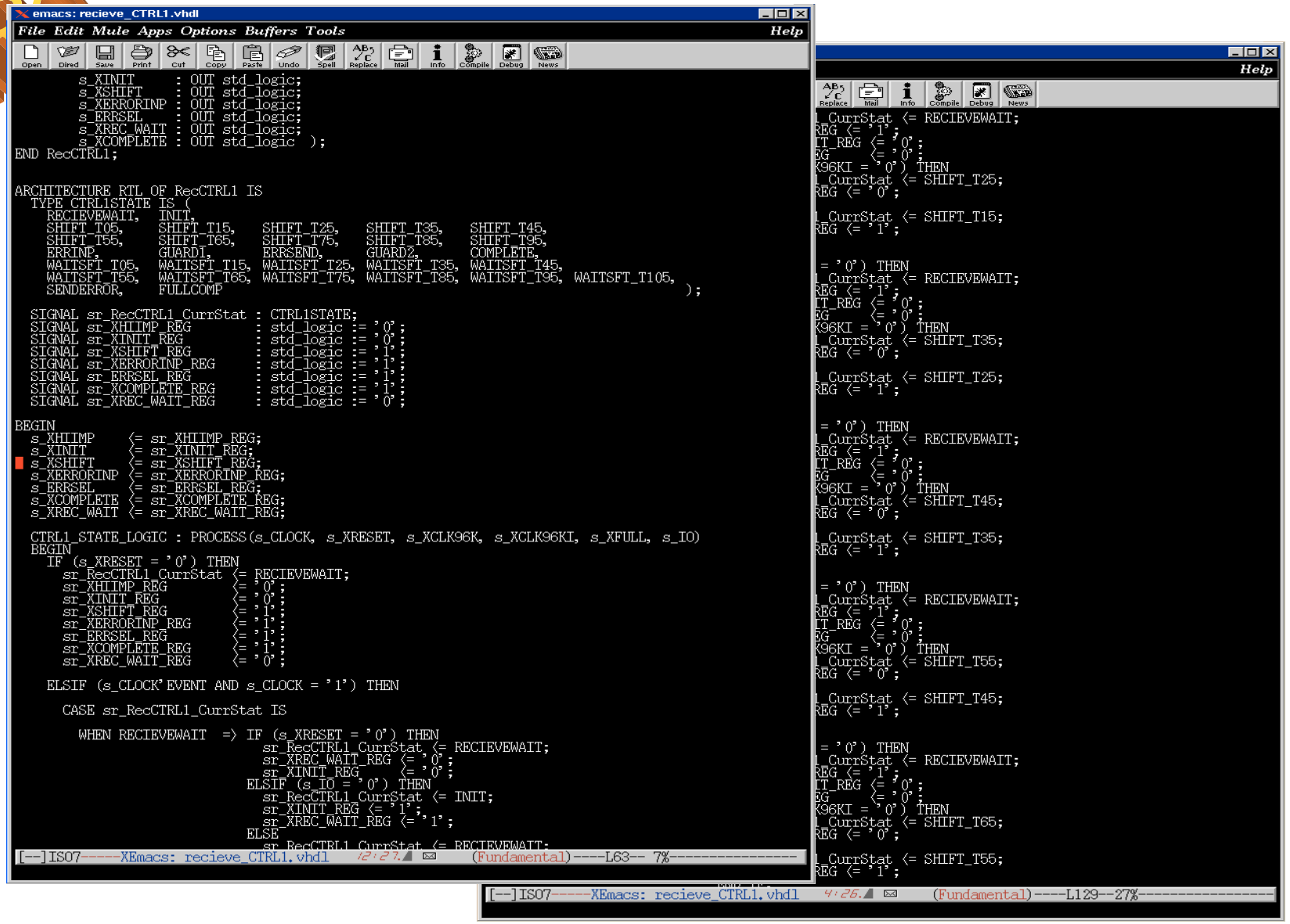
デジタルLSIの設計フロー



デジタル VLSI 設計フロー



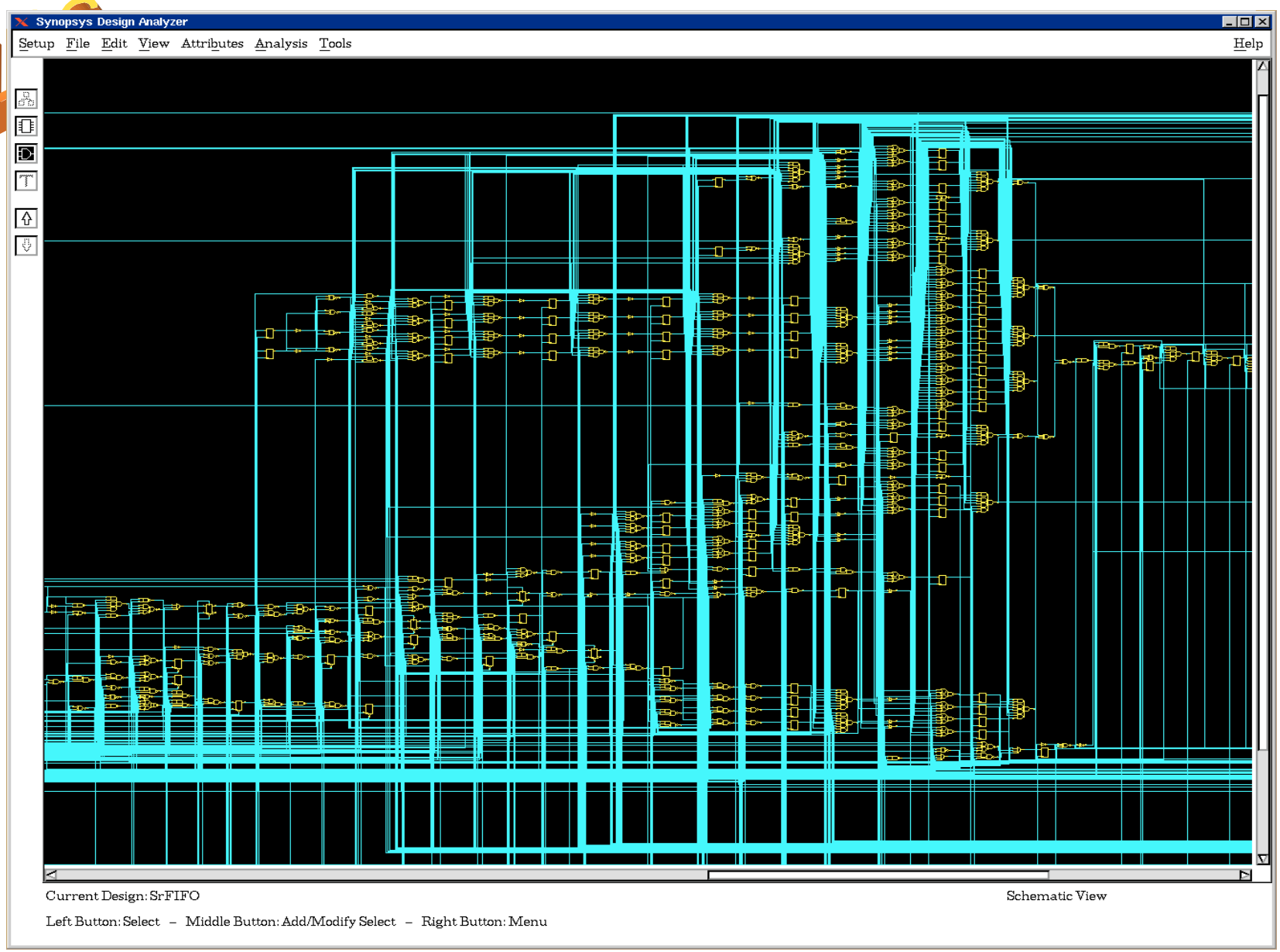

:send_SREG.obj (100%) "" [Modified]
Mode File Edit Layout Arrange View Text Graphics Color Navigate Special Help
SENDDATA 8
MakeParity
Parity
SerialDATA 1 0
I/O
11Bits シフトレジスタ
1
HiIMP
shift
ratch
1
clk96k
STATRATCH
ErrorINP
SendError FlipFlop
SETFFDATA
status_SendError
send
complete
select_inc
2
SendError
SELECT
2bits Counter
REGDATA0
REGDATA1
REGDATA2
REGDATA3
レジスタセレクタ
レジスタ #0
レジスタ #1
レジスタ #2
レジスタ #3
DATA BUS
Binary Decoder
EnableReg0-3
reg_write
PointReg
2bits Counter
pointer_inc
3
send_interrupt
usedreg_inc
full
usedreg_dec
empty
Used_register Counter
status_SendUsedReg
(none)
(none)
(none)

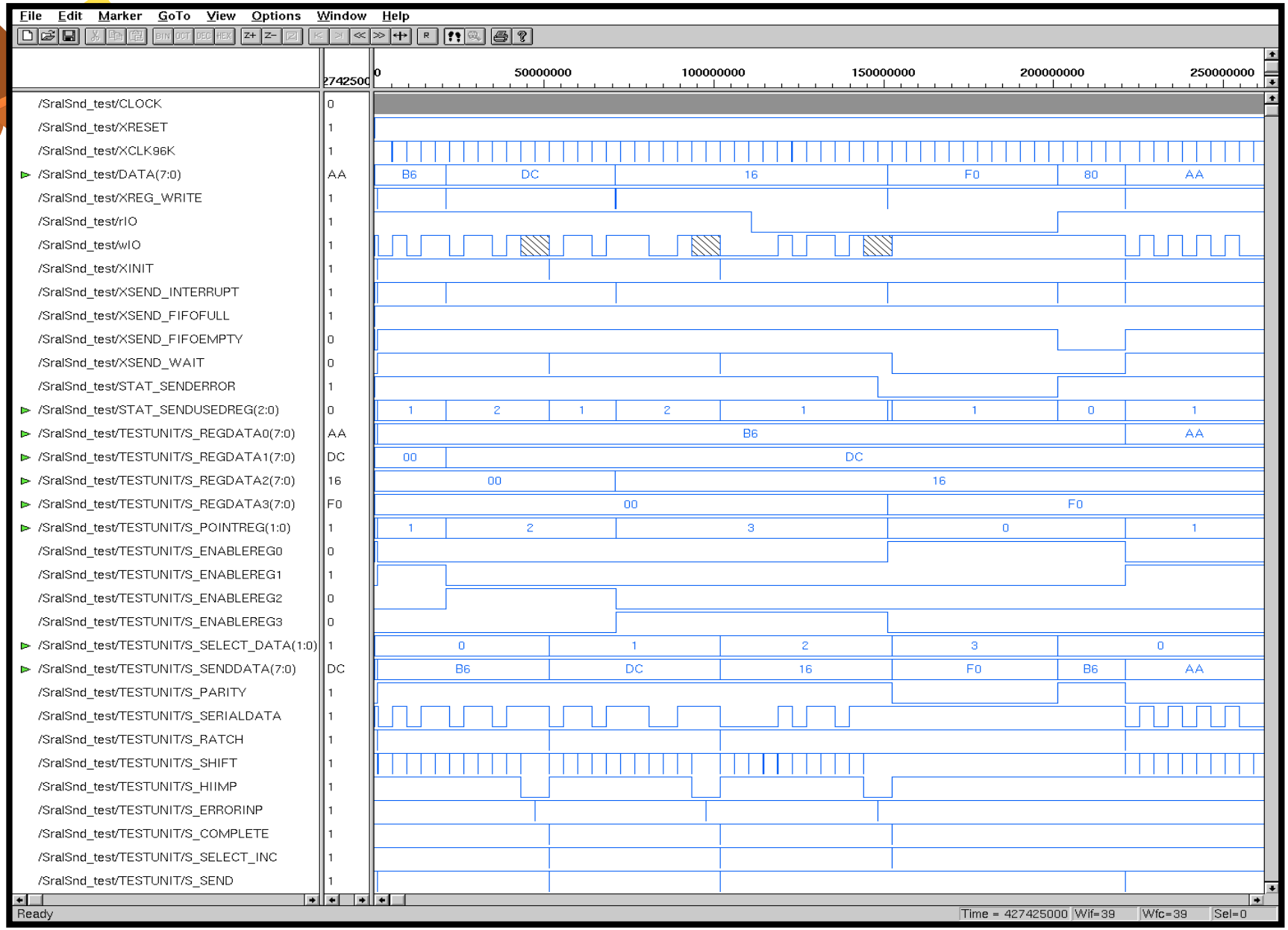

emacs: recieve_CTRL1.vhdl
File Edit Mule Apps Options Buffers Tools Help
Open Dired Save Print Cut Copy Paste Undo Spell Replace Mail Info Compile Debug News
s_XINIT : OUT std_logic;
s_XSHIFT : OUT std_logic;
s_XERRORINP : OUT std_logic;
s_ERRSEL : OUT std_logic;
s_XREC_WAIT : OUT std_logic;
s_XCOMPLETE : OUT std_logic);
END RecCTRL1;
ARCHITECTURE RTL OF RecCTRL1 IS
TYPE CTRL1STATE IS (
RECIEVEWAIT, INIT,
SHIFT_T05, SHIFT_T15, SHIFT_T25, SHIFT_T35, SHIFT_T45,
SHIFT_T55, SHIFT_T65, SHIFT_T75, SHIFT_T85, SHIFT_T95,
ERRINP, GUARD1, ERRSEND, GUARD2, COMPLETE,
WAITSFT_T05, WAITSFT_T15, WAITSFT_T25, WAITSFT_T35, WAITSFT_T45,
WAITSFT_T55, WAITSFT_T65, WAITSFT_T75, WAITSFT_T85, WAITSFT_T95, WAITSFT_T105,
SENDERROR, FULLCOMP);
SIGNAL sr_RecCTRL1_CurrStat : CTRL1STATE;
SIGNAL sr_XHIIMP_REG : std_logic := '0';
SIGNAL sr_XINIT_REG : std_logic := '0';
SIGNAL sr_XSHIFT_REG : std_logic := '1';
SIGNAL sr_XERRORINP_REG : std_logic := '1';
SIGNAL sr_ERRSEL_REG : std_logic := '1';
SIGNAL sr_XCOMPLETE_REG : std_logic := '1';
SIGNAL sr_XREC_WAIT_REG : std_logic := '0';
BEGIN
s_XHIIMP <= sr_XHIIMP_REG;
s_XINIT <= sr_XINIT_REG;
s_XSHIFT <= sr_XSHIFT_REG;
s_XERRORINP <= sr_XERRORINP_REG;
s_ERRSEL <= sr_ERRSEL_REG;
s_XCOMPLETE <= sr_XCOMPLETE_REG;
s_XREC_WAIT <= sr_XREC_WAIT_REG;
CTRL1_STATE_LOGIC : PROCESS(s_CLOCK, s_XRESET, s_XCLK96K, s_XCLK96KI, s_XFULL, s_IO)
BEGIN
IF (s_XRESET = '0') THEN
sr_RecCTRL1_CurrStat <= RECIEVEWAIT;
sr_XHIIMP_REG <= '0';
sr_XINIT_REG <= '0';
sr_XSHIFT_REG <= '1';
sr_XERRORINP_REG <= '1';
sr_ERRSEL_REG <= '1';
sr_XCOMPLETE_REG <= '1';
sr_XREC_WAIT_REG <= '0';
ELSIF (s_CLOCK'EVENT AND s_CLOCK = '1') THEN
CASE sr_RecCTRL1_CurrStat IS
WHEN RECIEVEWAIT => IF (s_XRESET = '0') THEN
sr_RecCTRL1_CurrStat <= RECIEVEWAIT;
sr_XREC_WAIT_REG <= '0';
sr_XINIT_REG <= '0';
ELSIF (s_IO = '0') THEN
sr_RecCTRL1_CurrStat <= INIT;
sr_XINIT_REG <= '1';
sr_XREC_WAIT_REG <='1';
ELSE
sr_RecCTRL1_CurrStat <= RECIEVEWAIT;
[--]ISO7-----XEmacs: recieve_CTRL1.vhdl 12:27 (Fundamental)----L63-- 7%
Help
Replace Mail Info Compile Debug News
[--]ISO7-----XEmacs: recieve_CTRL1.vhdl 4:26 (Fundamental)----L129--27%

動作記述

VHDL記述

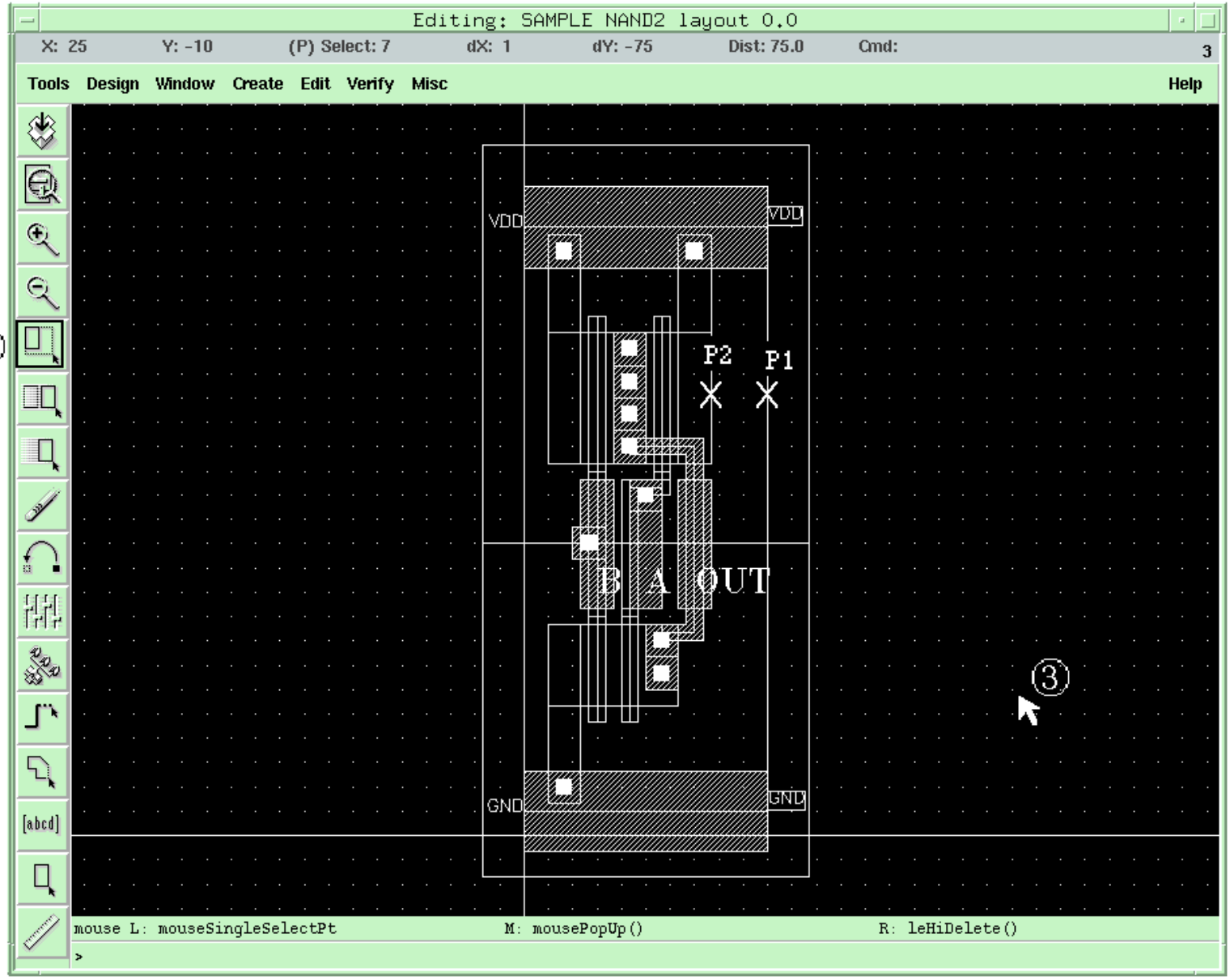
```
-- KUE-CHIP2 RTL Description in VHDL  Version 0.30  1997/Jul./07
-- Copyright 1997 ASTEM RI    Written by H. Kanbara
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_unsigned.all;
use WORK.pkg_kue2.all;
entity kue_chip2 is
   port(CLOCK_p      : in   std_logic;
        RESET_p      : in   std_logic;
        DBI_p        : in   std_logic_vector(7 downto 0);
        SP_p         : in   std_logic;
        SI_p         : in   std_logic;
        SS_p         : in   std_logic;
        SET_p        : in   std_logic;
        ADR_INC_p    : in   std_logic;
        ADR_DEC_p    : in   std_logic;
        OBS_SEL_p    : in   std_logic_vector(3 downto 0);
        IBUF_FLG_IN_p : in   std_logic;
        OBUF_FLG_IN_p : in   std_logic;
        OP_p         : out  std_logic;
        PHASE_p      : out  std_logic_vector(4 downto 0);
        AB_p         : out  std_logic_vector(8 downto 0);
        OB_p         : out  std_logic_vector(7 downto 0);
        DBO_p        : out  std_logic_vector(7 downto 0);
        MEM_OB_p     : out  std_logic;
        MEM_RE_p     : out  std_logic;
        MEM_WE_p     : out  std_logic;
        IBUF_FLG_CLR_p : out  std_logic;
        IBUF_RE_p    : out  std_logic;
        PANEL_RE_p   : out  std_logic;
        OBUF_WE_p    : out  std_logic
   );
end kue_chip2;
```

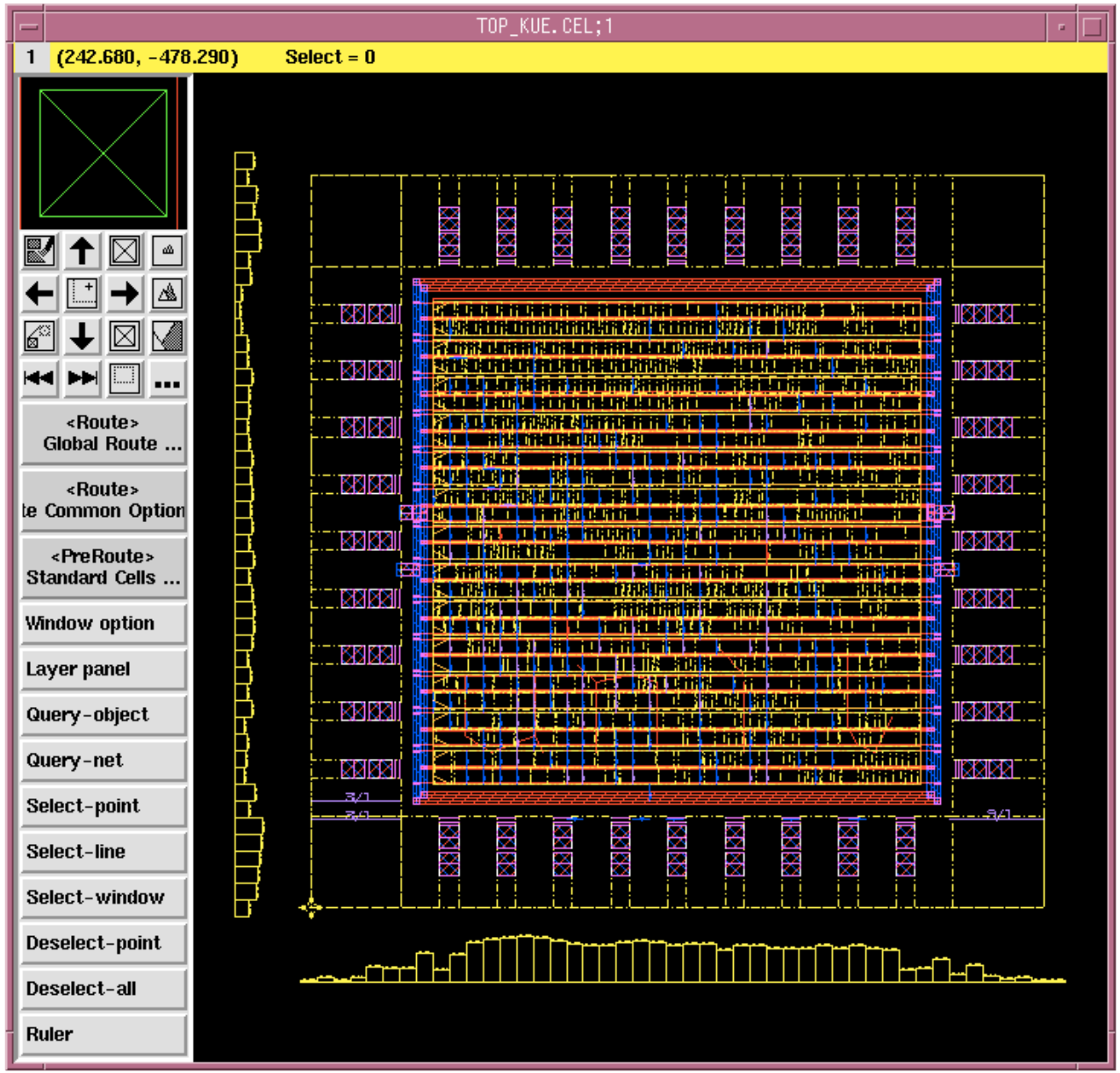

Synopsys Design Analyzer
Setup File Edit View Attributes Analysis Tools Help
Current Design: SrFIFO
Schematic View
Left Button: Select – Middle Button: Add/Modify Select – Right Button: Menu


File Edit Marker GoTo View Options Window Help
2742500
0
50000000
100000000
150000000
200000000
250000000
/SralSnd_test/CLOCK 0
/SralSnd_test/XRESET 1
/SralSnd_test/XCLK96K 1
/SralSnd_test/DATA(7:0) AA B6 DC 16 F0 80 AA
/SralSnd_test/XREG_WRITE 1
/SralSnd_test/rIO 1
/SralSnd_test/wIO 1
/SralSnd_test/XINIT 1
/SralSnd_test/XSEND_INTERRUPT 1
/SralSnd_test/XSEND_FIFOFULL 1
/SralSnd_test/XSEND_FIFOEMPTY 0
/SralSnd_test/XSEND_WAIT 0
/SralSnd_test/STAT_SENDERROR 1
/SralSnd_test/STAT_SENDUSEDREG(2:0) 0 1 2 1 2 1 1 0 1
/SralSnd_test/TESTUNIT/S_REGDATA0(7:0) AA B6 AA
/SralSnd_test/TESTUNIT/S_REGDATA1(7:0) DC 00 DC
/SralSnd_test/TESTUNIT/S_REGDATA2(7:0) 16 00 16
/SralSnd_test/TESTUNIT/S_REGDATA3(7:0) F0 00 F0
/SralSnd_test/TESTUNIT/S_POINTREG(1:0) 1 1 2 3 0 1
/SralSnd_test/TESTUNIT/S_ENABLEREG0 0
/SralSnd_test/TESTUNIT/S_ENABLEREG1 1
/SralSnd_test/TESTUNIT/S_ENABLEREG2 0
/SralSnd_test/TESTUNIT/S_ENABLEREG3 0
/SralSnd_test/TESTUNIT/S_SELECT_DATA(1:0) 1 0 1 2 3 0
/SralSnd_test/TESTUNIT/S_SENDDATA(7:0) DC B6 DC 16 F0 B6 AA
/SralSnd_test/TESTUNIT/S_PARITY 1
/SralSnd_test/TESTUNIT/S_SERIALDATA 1
/SralSnd_test/TESTUNIT/S_RATCH 1
/SralSnd_test/TESTUNIT/S_SHIFT 1
/SralSnd_test/TESTUNIT/S_HIIMP 1
/SralSnd_test/TESTUNIT/S_ERRORINP 1
/SralSnd_test/TESTUNIT/S_COMPLETE 1
/SralSnd_test/TESTUNIT/S_SELECT_INC 1
/SralSnd_test/TESTUNIT/S_SEND 1
Ready
Time = 427425000 Wif=39 Wfc=39 Sel=0

動作の検証

NANDゲートの設計


Editing: SAMPLE NAND2 layout 0.0
X: 25
Y: -10
(P) Select: 7
dX: 1
dY: -75
Dist: 75.0
Cmd:
3
Tools
Design
Window
Create
Edit
Verify
Misc
Help
①
VDD
VDD
P2
P1
B
A
OUT
③
GND
GND
[abcd]
mouse L: mouseSingleSelectPt
M: mousePopUp()
R: leHiDelete()
>


TOP_KUE.CEL;1
1 (242.680, -478.290) Select = 0
<Route>
Global Route ...
<Route>
te Common Option
<PreRoute>
Standard Cells ...
Window option
Layer panel
Query-object
Query-net
Select-point
Select-line
Select-window
Deselect-point
Deselect-all
Ruler

システムLSI設計における形式的手法への期待

1．システムLSIとは？

2．システムLSIの設計の流れ

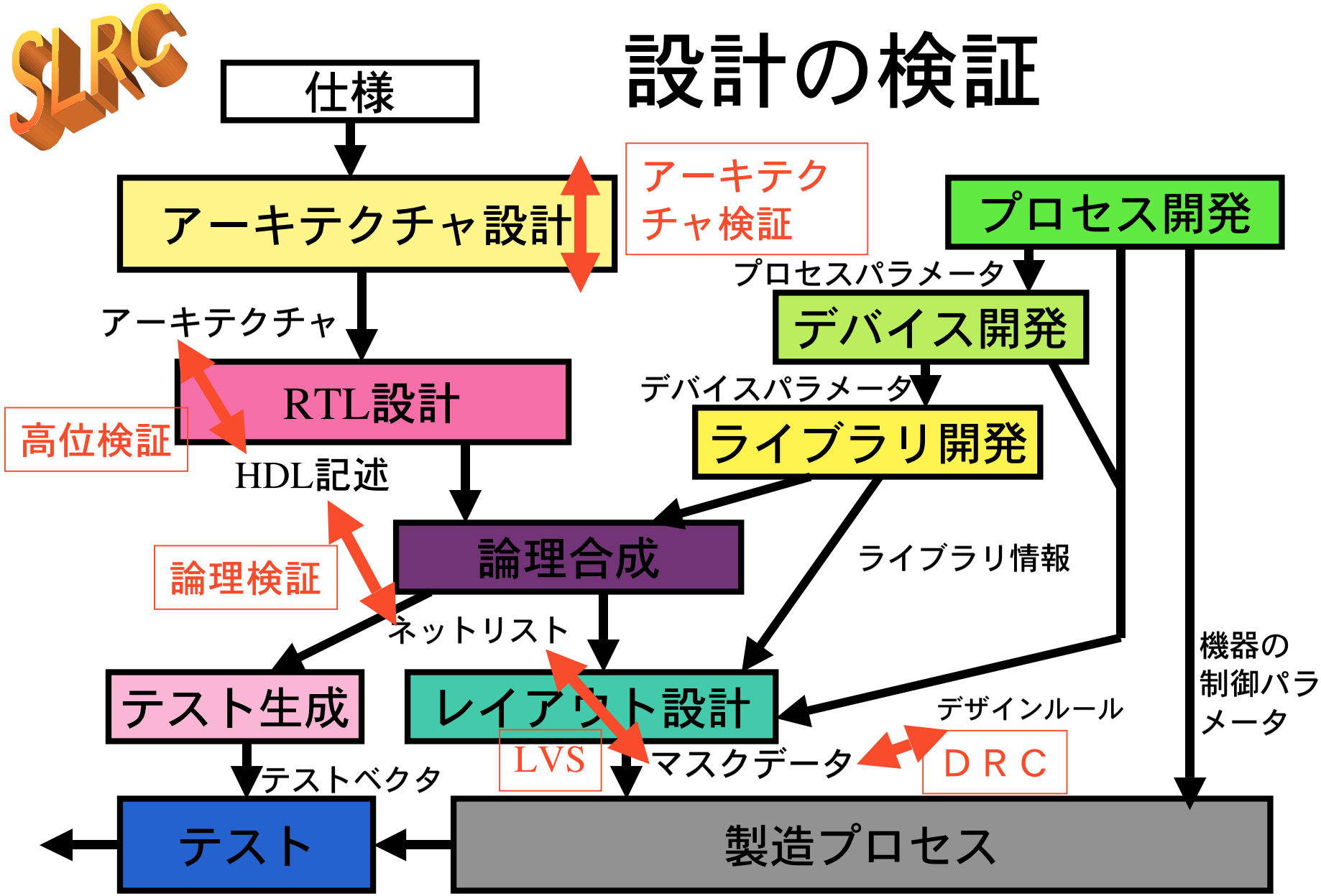
3．形式的検証

4．仕様設計と形式的手法

5．システムLSIの経済学

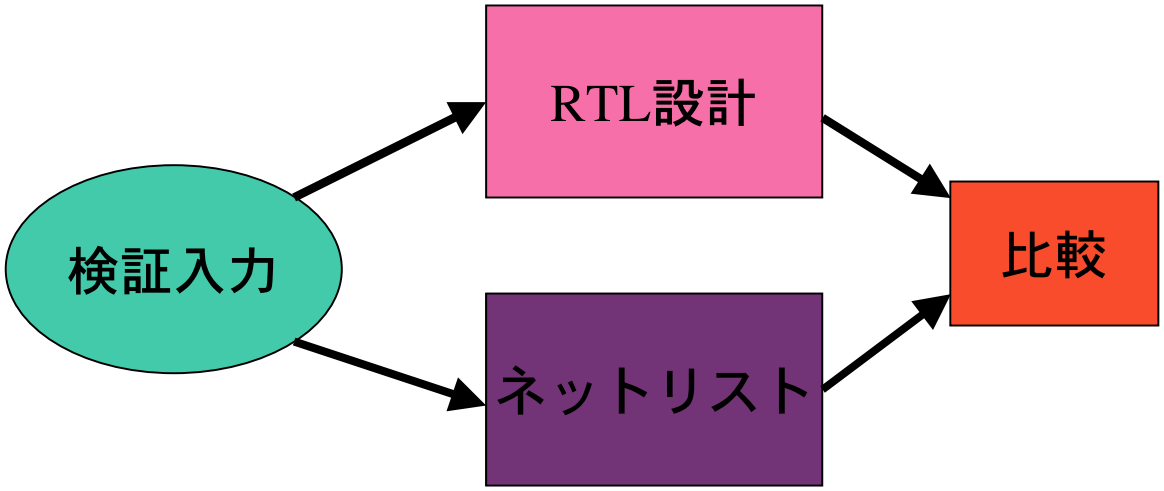
6．Quality, Reliability and Security

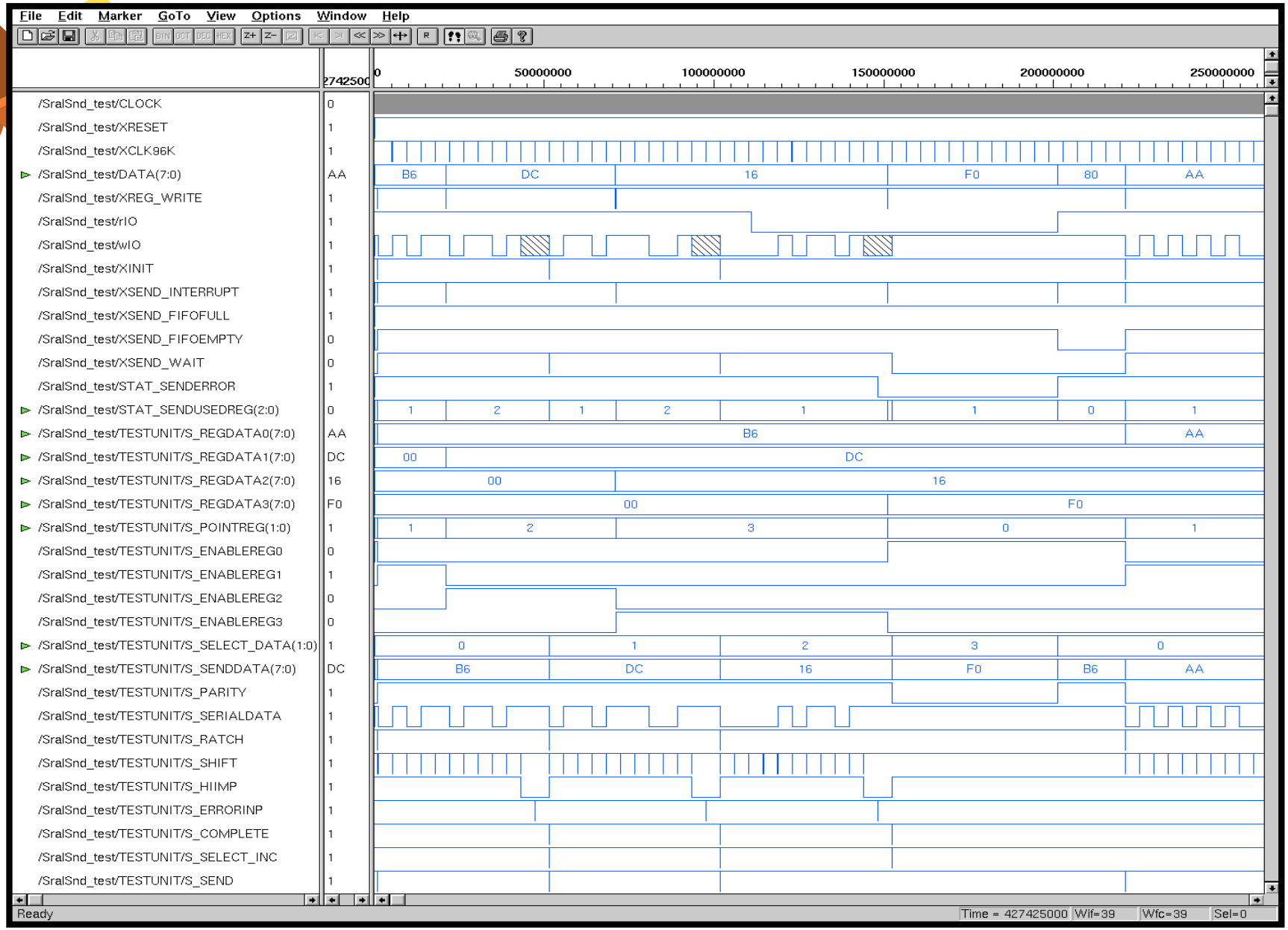
設計の検証


仕様
アーキテクチャ設計
アーキテクチャ検証
プロセス開発
プロセスパラメータ
アーキテクチャ
デバイス開発
RTL設計
高位検証
デバイスパラメータ
ライブラリ開発
HDL記述
論理合成
論理検証
ライブラリ情報
ネットリスト
テスト生成
レイアウト設計
デザインルール
機器の制御パラメータ
LVS
マスクデータ
DRC
テストベクタ
テスト
製造プロセス
チップ

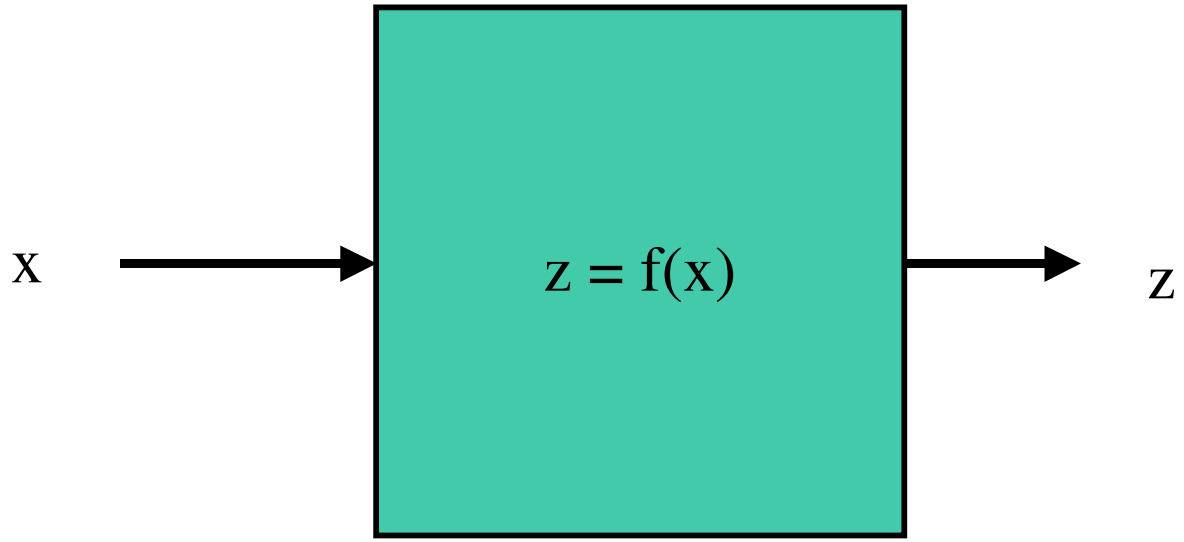
論理検証

- 仕様・アーキテクチャ設計・RTL設計と論理設計結果（ネットリスト）の等価性を確認する。
- シミュレーションと形式的検証




File Edit Marker GoTo View Options Window Help
2742500
0
50000000
100000000
150000000
200000000
250000000
/SralSnd_test/CLOCK 0
/SralSnd_test/XRESET 1
/SralSnd_test/XCLK96K 1
/SralSnd_test/DATA(7:0) AA B6 DC 16 F0 80 AA
/SralSnd_test/XREG_WRITE 1
/SralSnd_test/rIO 1
/SralSnd_test/wIO 1
/SralSnd_test/XINIT 1
/SralSnd_test/XSEND_INTERRUPT 1
/SralSnd_test/XSEND_FIFOFULL 1
/SralSnd_test/XSEND_FIFOEMPTY 0
/SralSnd_test/XSEND_WAIT 0
/SralSnd_test/STAT_SENDERROR 1
/SralSnd_test/STAT_SENDUSEDREG(2:0) 0 1 2 1 2 1 1 0 1
/SralSnd_test/TESTUNIT/S_REGDATA0(7:0) AA B6 AA
/SralSnd_test/TESTUNIT/S_REGDATA1(7:0) DC 00 DC
/SralSnd_test/TESTUNIT/S_REGDATA2(7:0) 16 00 16
/SralSnd_test/TESTUNIT/S_REGDATA3(7:0) F0 00 F0
/SralSnd_test/TESTUNIT/S_POINTREG(1:0) 1 1 2 3 0 1
/SralSnd_test/TESTUNIT/S_ENABLEREG0 0
/SralSnd_test/TESTUNIT/S_ENABLEREG1 1
/SralSnd_test/TESTUNIT/S_ENABLEREG2 0
/SralSnd_test/TESTUNIT/S_ENABLEREG3 0
/SralSnd_test/TESTUNIT/S_SELECT_DATA(1:0) 1 0 1 2 3 0
/SralSnd_test/TESTUNIT/S_SENDDATA(7:0) DC B6 DC 16 F0 B6 AA
/SralSnd_test/TESTUNIT/S_PARITY 1
/SralSnd_test/TESTUNIT/S_SERIALDATA 1
/SralSnd_test/TESTUNIT/S_RATCH 1
/SralSnd_test/TESTUNIT/S_SHIFT 1
/SralSnd_test/TESTUNIT/S_HIIMP 1
/SralSnd_test/TESTUNIT/S_ERRORINP 1
/SralSnd_test/TESTUNIT/S_COMPLETE 1
/SralSnd_test/TESTUNIT/S_SELECT_INC 1
/SralSnd_test/TESTUNIT/S_SEND 1
Ready
Time = 427425000 Wif=39 Wfc=39 Sel=0

形式的論理検証



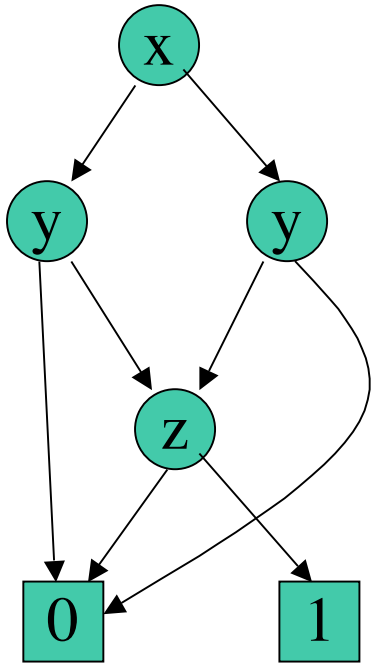
入力定義域の任意の入力xに対して回路はf(x)を計算するか？

組合わせ回路：論理関数$f:\{0,1\}^n \rightarrow \{0,1\}^m$

順序回路：系列の変換$f: \Sigma^n \rightarrow \Gamma^n$

論理関数の検証手法

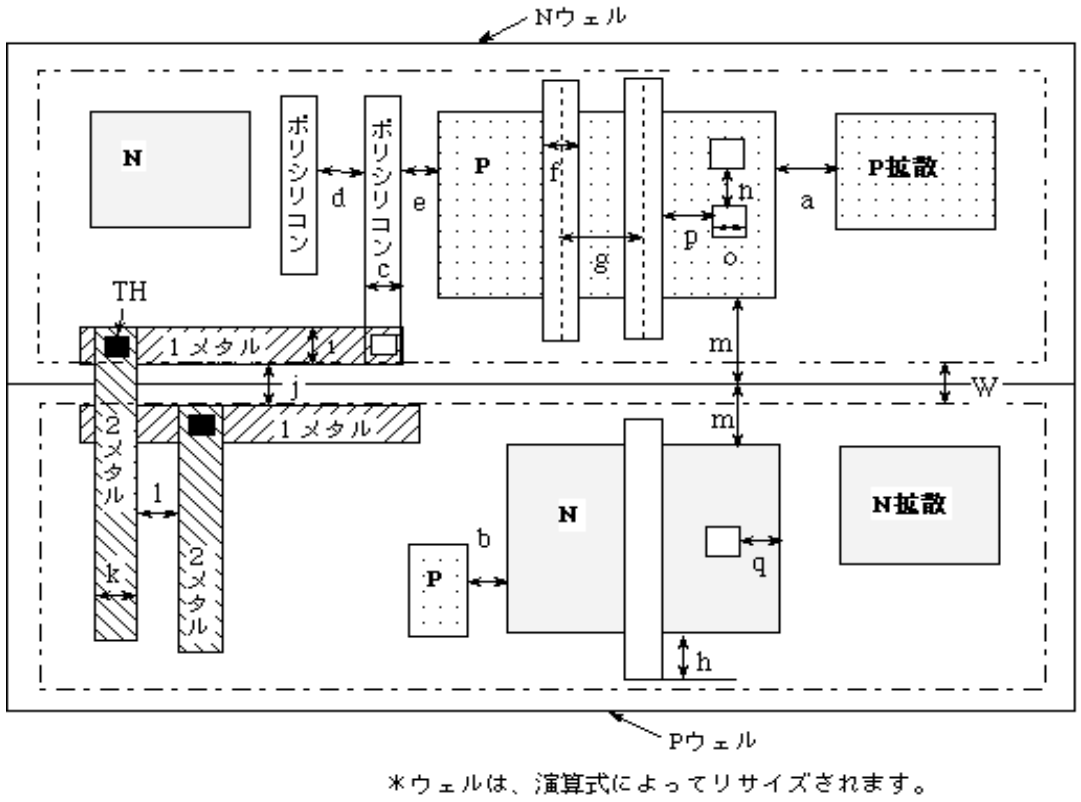
- シンボリックシミュレーション
- データ構造：BDD (Binary Decision Diagram)
  - コンパクトな表現
  - 演算の容易さ
  - 等価性判定の容易さ
    - 変数順を固定すれば簡約形は一意



$$\bar{x}\,y\,z + x\,y\,\bar{z}$$

Design Rule Check

- 製造段階での加工精度の揺らぎに対するマージンの確保
- マスクデータの中に設計ルール違反がないかを検証する。
- 図形処理

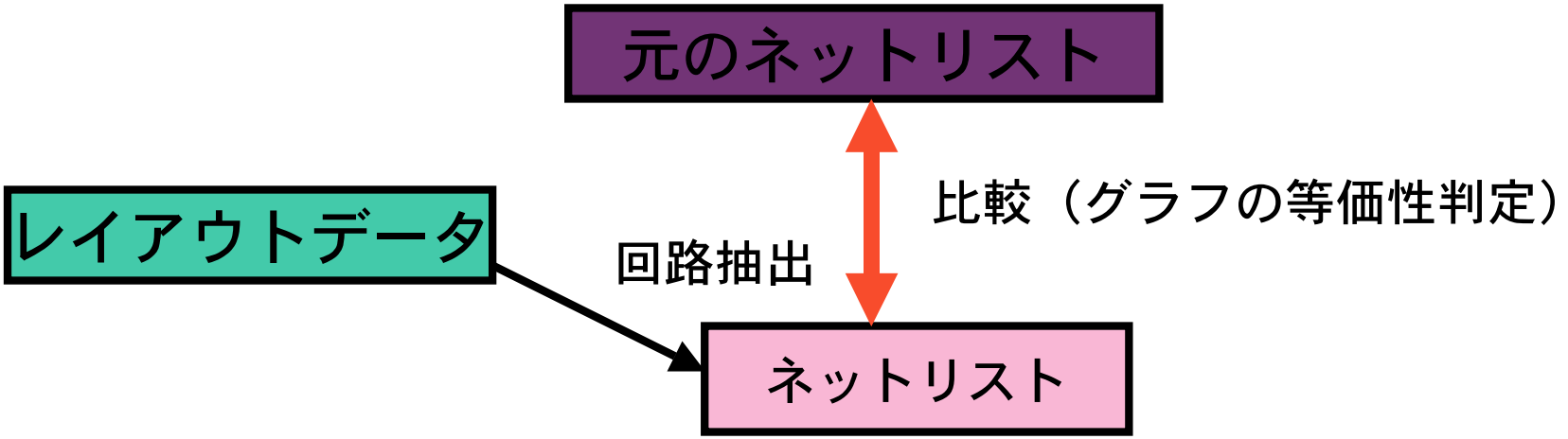


Design Rule Checkの現状

- 階層的設計
  - 各ブロック単位でチェックしておけば，そのブロックを用いた上位の設計ではブロック内のチェックは不要
- 現実
  - 設計の最終段階ですべての階層を展開して全体のチェックを行う．数時間から数十時間かけてDRCを行っている.

LVS (Layout v.s. Schematic)

- ネットリスト通りレイアウトがなされているかをチェックする



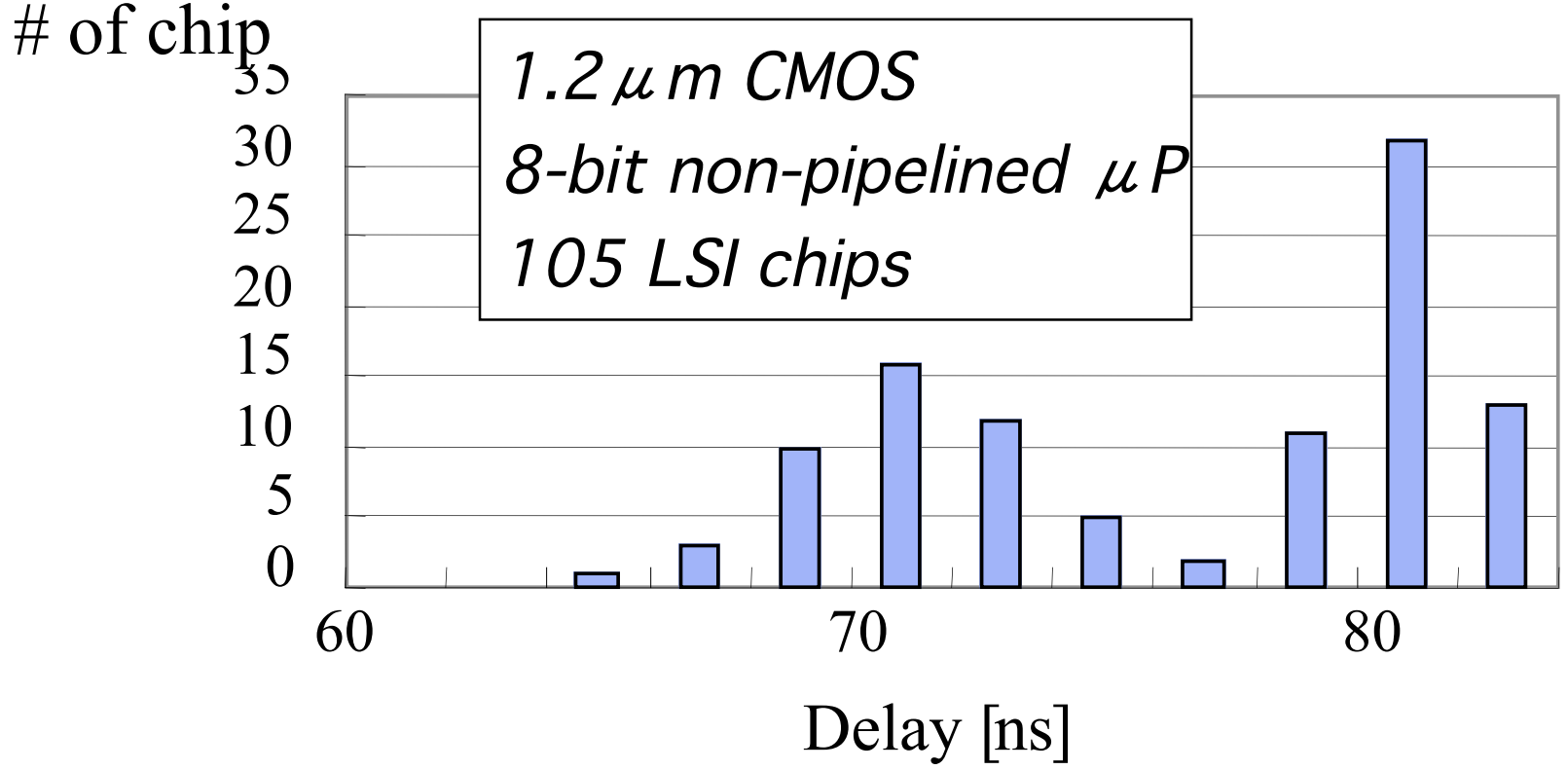
LVSの必要性

- 自動レイアウトツールのバグ
  - アルゴリズムやプログラムのミス
  - 階層間の接続の不具合
  - 数値計算上の誤差
- データのバージョン管理の不備
  - 部分変更への対応

性能の検証

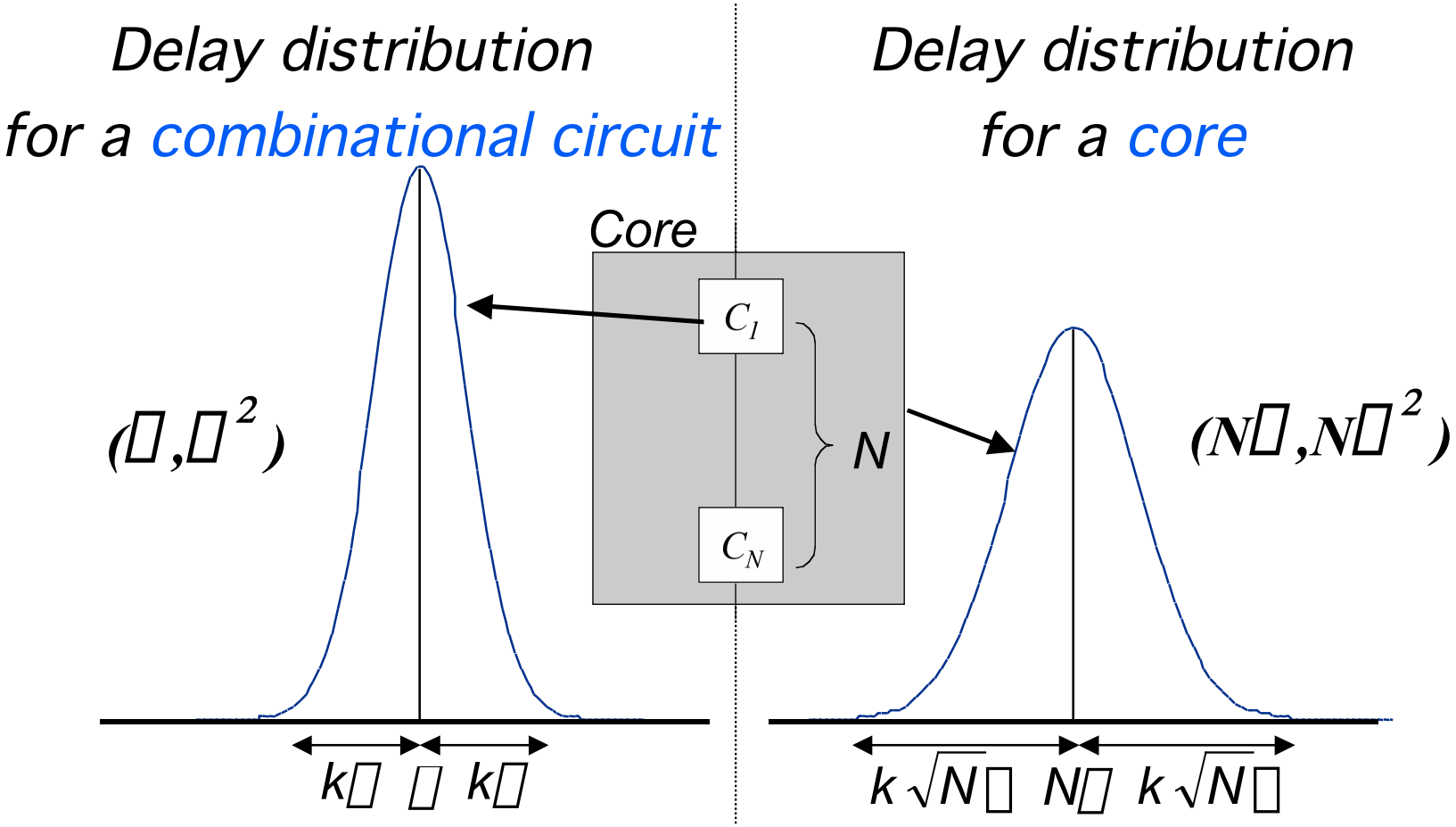
- 同期式回路における時間軸方向の検証
  - 実現すべき機能を与えられたクロック周期内で計算できるか？
- 論理検証ー＞遅延検証
- LVS ー＞回路パラメータ抽出、回路シミュレーション（主にライブラリ開発で行う。）

回路の遅延ばらつき



Source:H.Matsunaga et. al., 55th IPSJ Annual Convention(1)

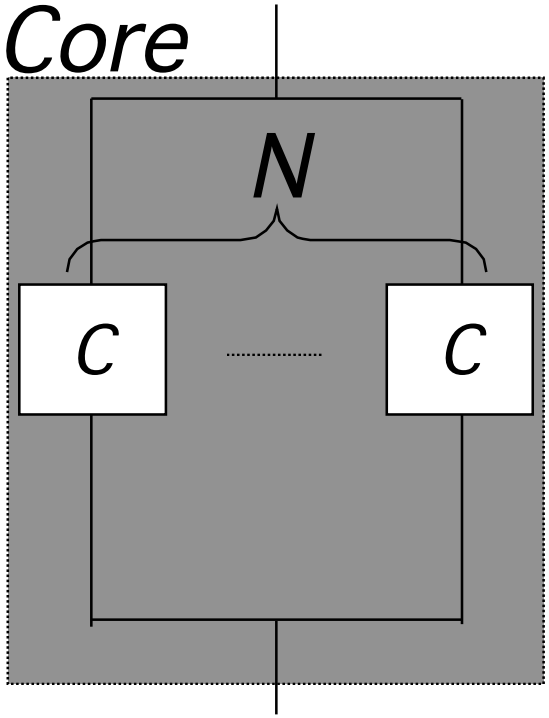
直列接続の遅延ばらつき



並列接続の遅延ばらつき

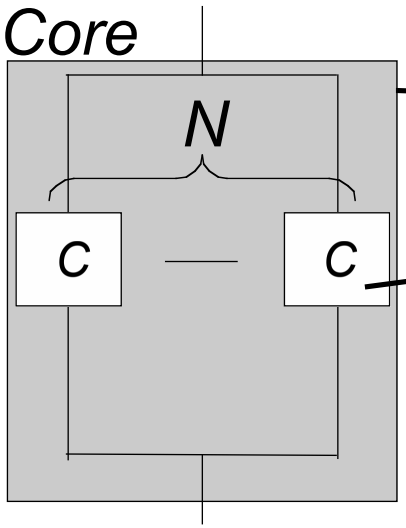
•N combinational circuits of a core are chained **in parallel**, and they obey the same delay distribution D(t). Also, an estimated delay Tc is assumed to be given for a single combinational circuit C. Then, the delay probability of the core is approximated as

$$1 - N \int_{T_c} D(t)\,dt$$



並列接続は過小評価

$$P = 1 - N\int_{Tc}^{\infty} D(t)\,dt \quad - \quad 1 - P = N\left(1 - \int_{0}^{Tc} D(t)\,dt\right)$$



(Prob. of violating delay Tc for a core)
=N (Prob. of violating delay Tc for a comb.)

システムLSI設計における形式的手法への期待

1．システムLSIとは？

2．システムLSIの設計の流れ

3．形式的検証

4．上流設計と形式的手法

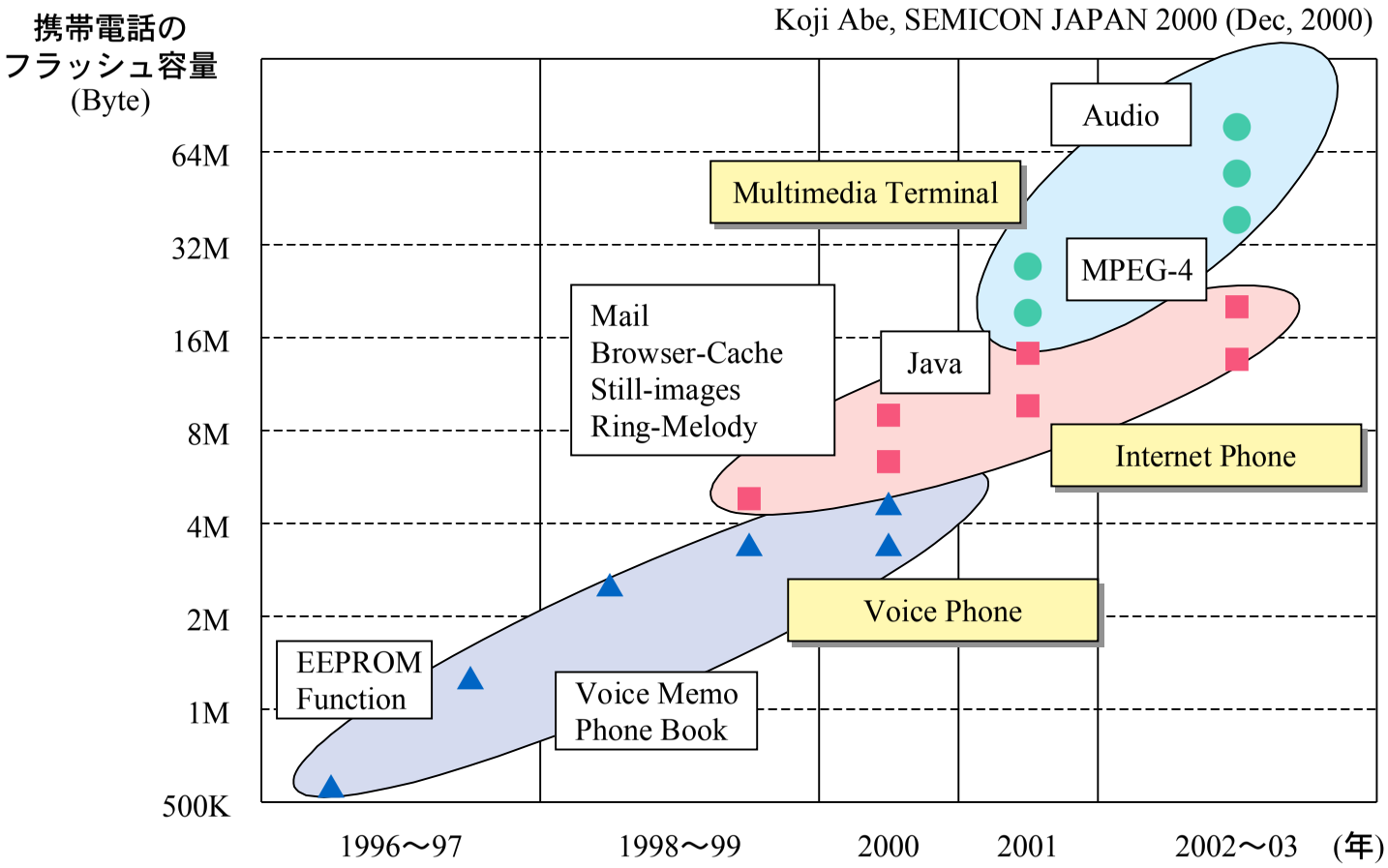
5．システムLSIの経済学

6．Quality, Reliability and Security

携帯電話用SOC開発の危機

- ソフトウェアの爆発的増加
  - 3Mステップのソフトウェア(Cベース)
    - インターネット接続
    - Audioサービス，画像配信
- ソフトウェア開発の生産性
  - C言語で200ステップ／人月（テストまで含む）
  - 開発人員3000人，開発費５００億円
  - デジタル回路部分は200人程度で1st Cutで動作
  - マルチプロセッサ，マルチメモリ環境
  - 付加されるソフトウェア要求の急増

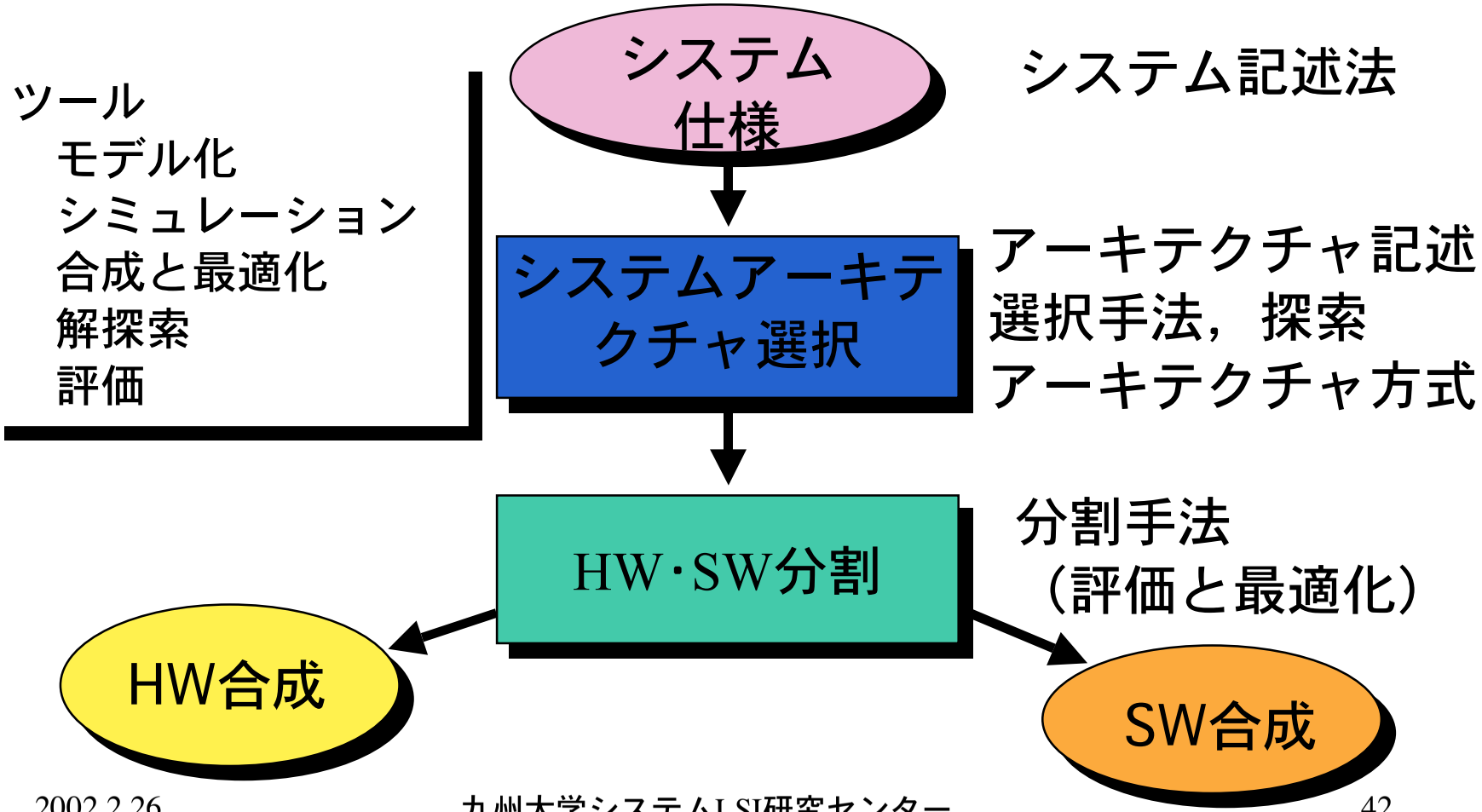
携帯電話の組込ソフトウェアの推移



危機の原因

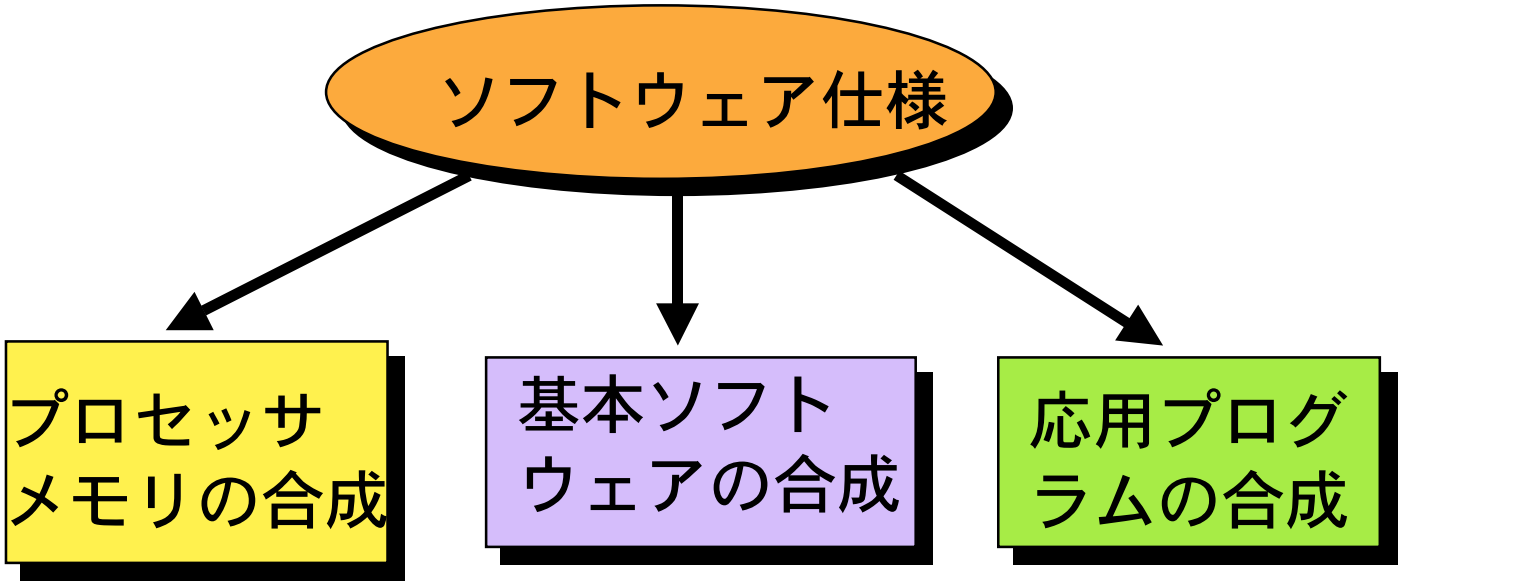
- システム仕様の流動性
  - PC的なadd on型のソフトウェア構造
  - Consumer Electronicsに求められる自己完結性
- 生産側の技術的未熟さ
  - ソフトウェアエンジニアリングの不在
  - ハードウェアとソフトウェアの同時開発
- 製品寿命と販売量（1年で数千万台）
  - 不良品の回収コストの爆発
  - 利用者の数と利用形態の多様性
  - 6ヶ月の製品寿命

システムLSI設計のフロー

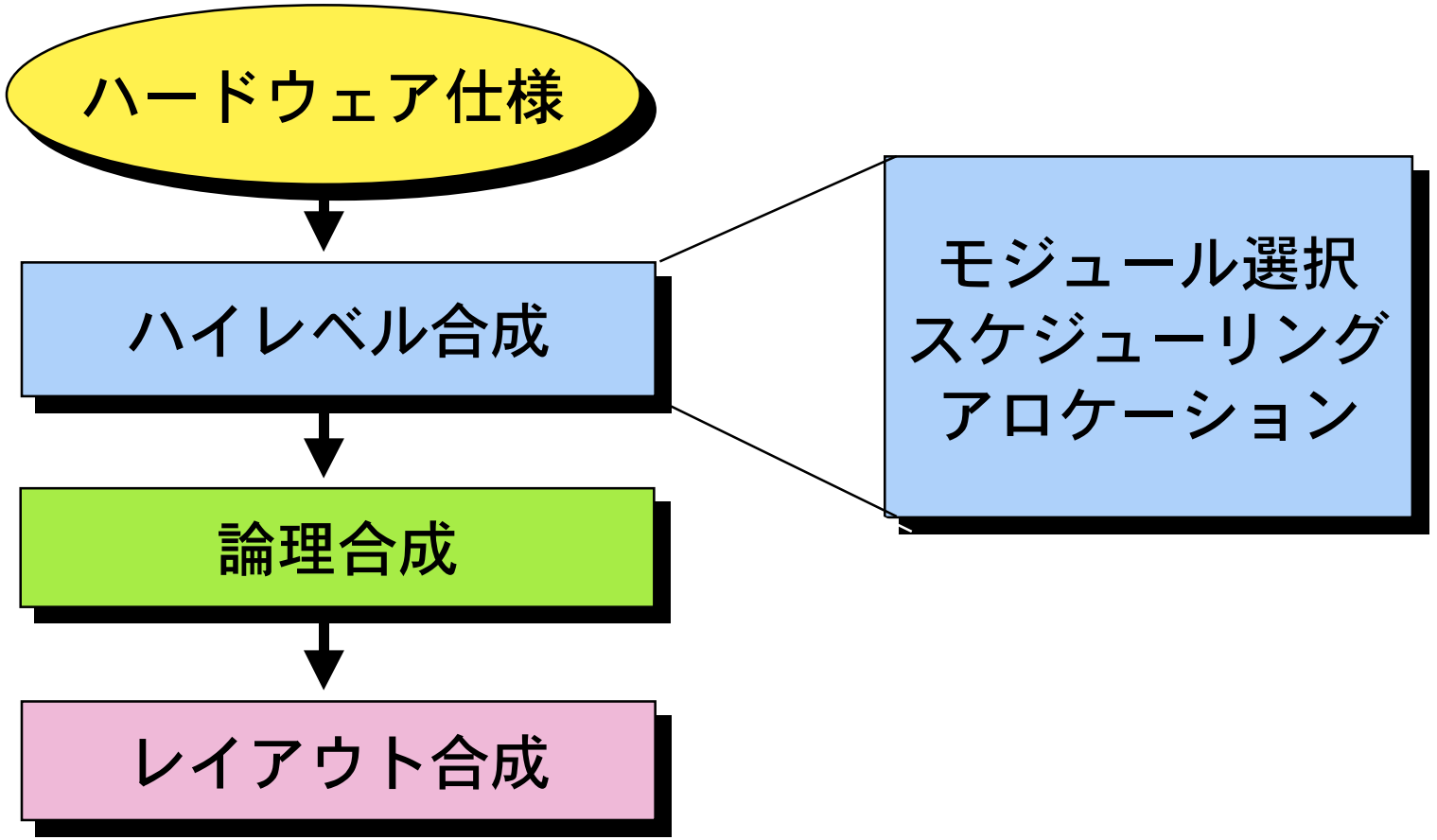



SLRC

ソフトウェア合成


ソフトウェア仕様
プロセッサ
メモリの合成
基本ソフト
ウェアの合成
応用プログ
ラムの合成
汎用・専用
データパス幅
命令体系など
コンパイラ
OS
Debugger
プログラムサイズ
データサイズ
性能，消費電力

ハードウェア合成



仕様記述言語

- VLSIシステム　VHDL, Verilog, SpecC, System C
- DSP　COSSAP,...
- 連続系　MATLAB, MATRIXX,...
- プロトコル　SDL-CLOTOS, ESTELLE,..
- 同期システム　ESTEREL, STATECHART,..
- プログラム　C++, JAVA, C, FORTRAN,
- 高位ソフトウェア　VDM, Z, B, FUNMATH,

各分野でそれぞれの言語がある.

表現力，解析力，汎用性

何が難しいか？

- モデル（Semantics）と文法（Syntax）

  a:=b; b:=a;　同時？逐次？非決定的？

  モデルの多様性（順序機械の定義，通信モデルの違いなど）
- 時間（絶対時間）と順序（因果）関係（相対時間）

  after 500ns;

  after event *b*;
- 抽象化と詳細化

  命令サイクル，クロックサイクル，実時間...
- 計算精度

  ワード，バイト，ビット，アナログ，．．．

問題点１

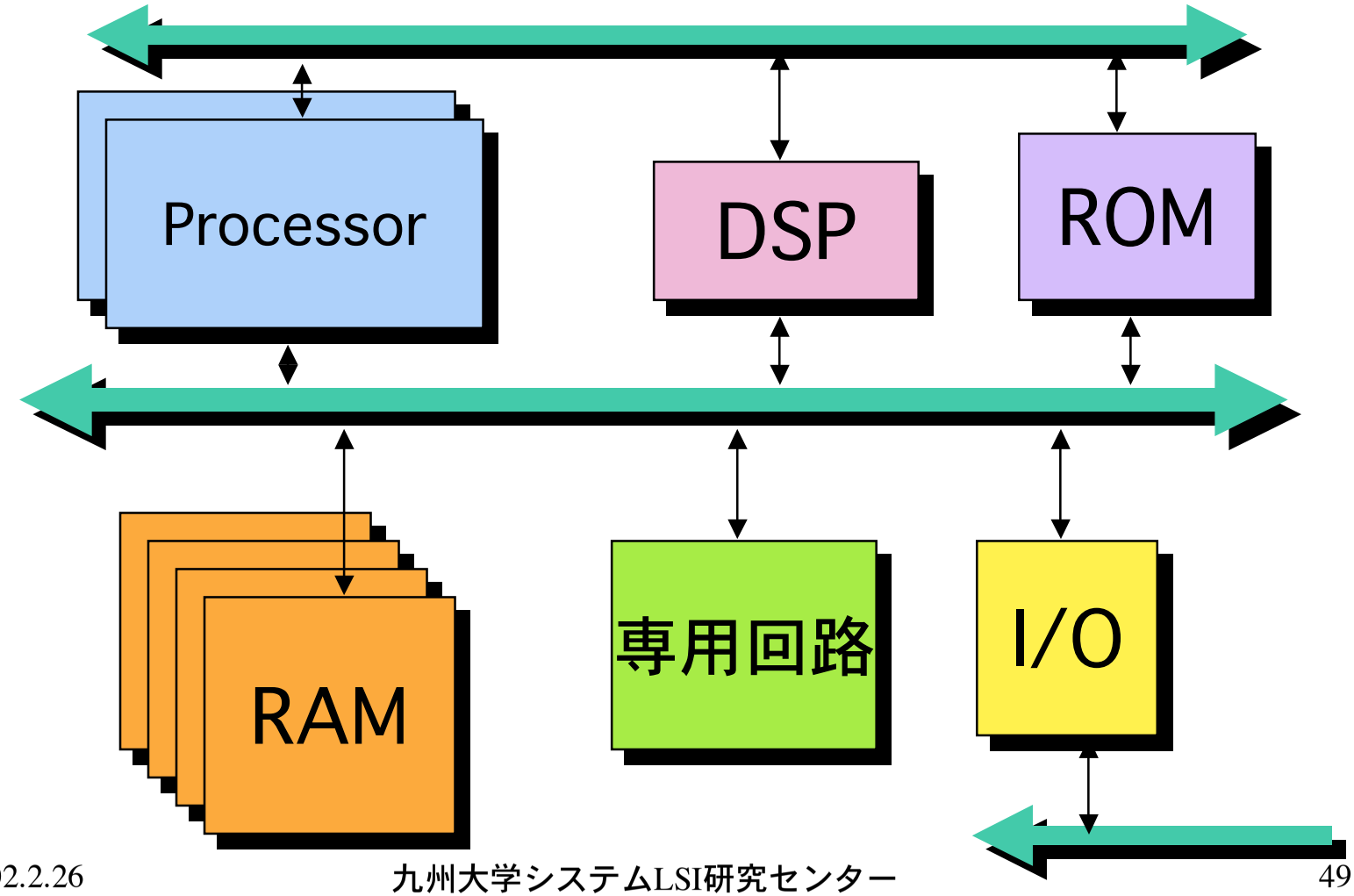
- 仕様をどのように記述するのか？
- 仕様に何を記述するのか？
- 何が仕様なのか？

- 設計制約
  - LSIの物理的制約（性能，規模，電力）
  - 設計時間と設計コスト
  - 既存の規約（通信プロトコル，外部装置）

アーキテクチャの探索

- 部品の選択
  - プロセッサ，メモリ，バスなど
- 動作／変数／チャネルの分割
- バインディング／アロケーション
- 結合の決定
- スケジューリング
- 評価

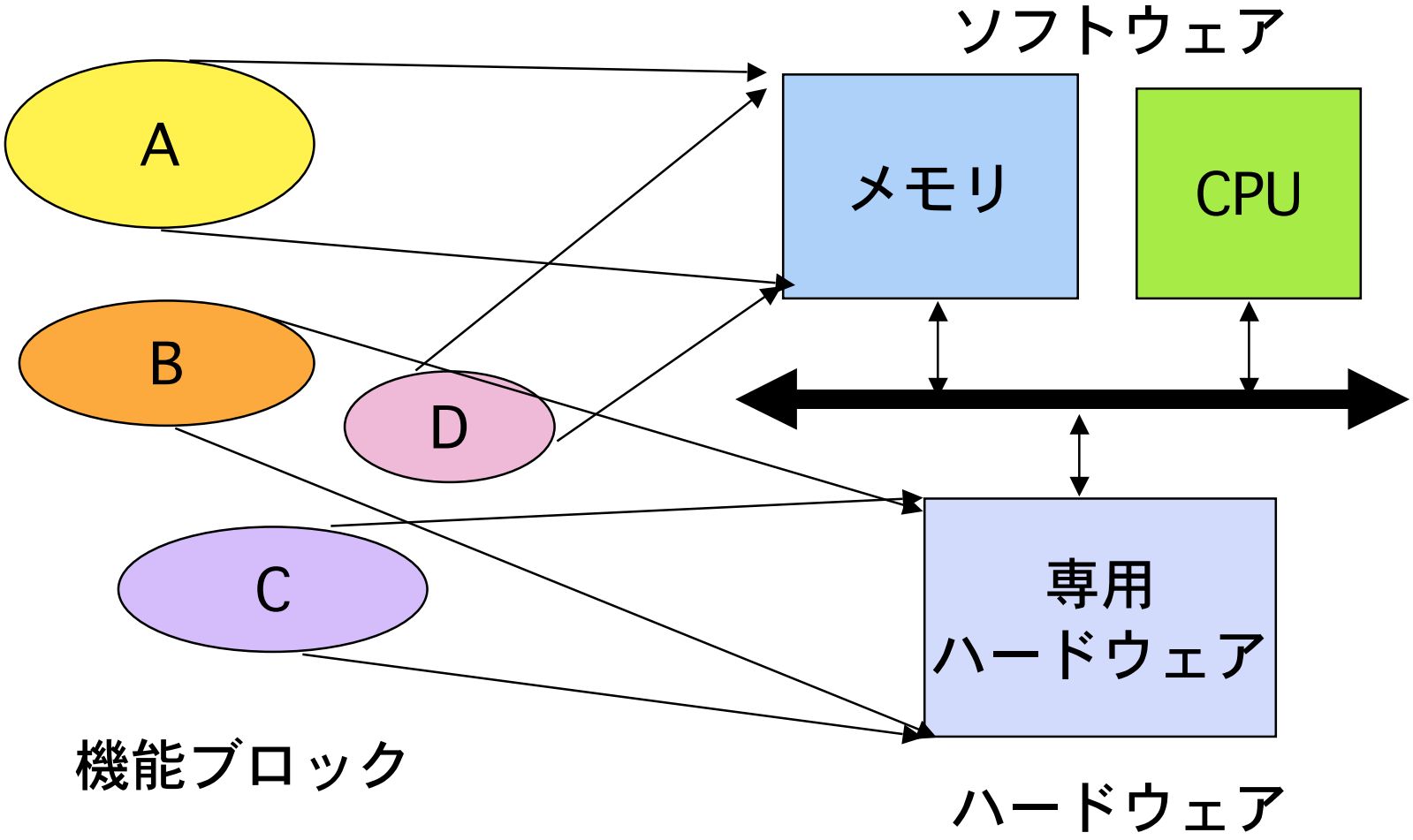
アーキテクチャの探索



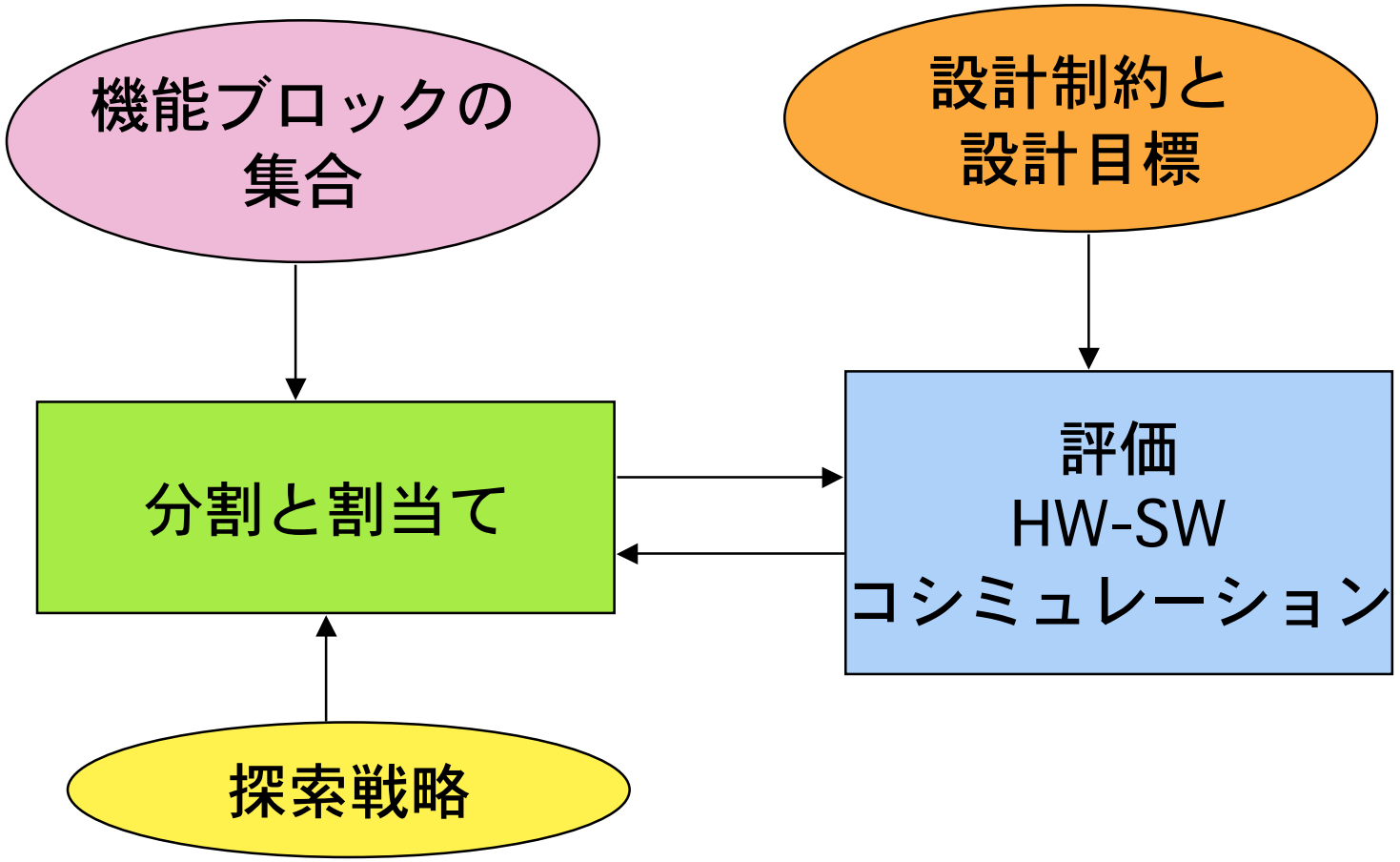
HW-SW分割

- 入力：
  - 機能ブロックの集合
  - 基本アーキテクチャ
  - 設計目標と設計制約
- 出力
  - 機能ブロックの実現形態（HW or SW)

HW-SW分割の概念



HW-SW分割の仕組



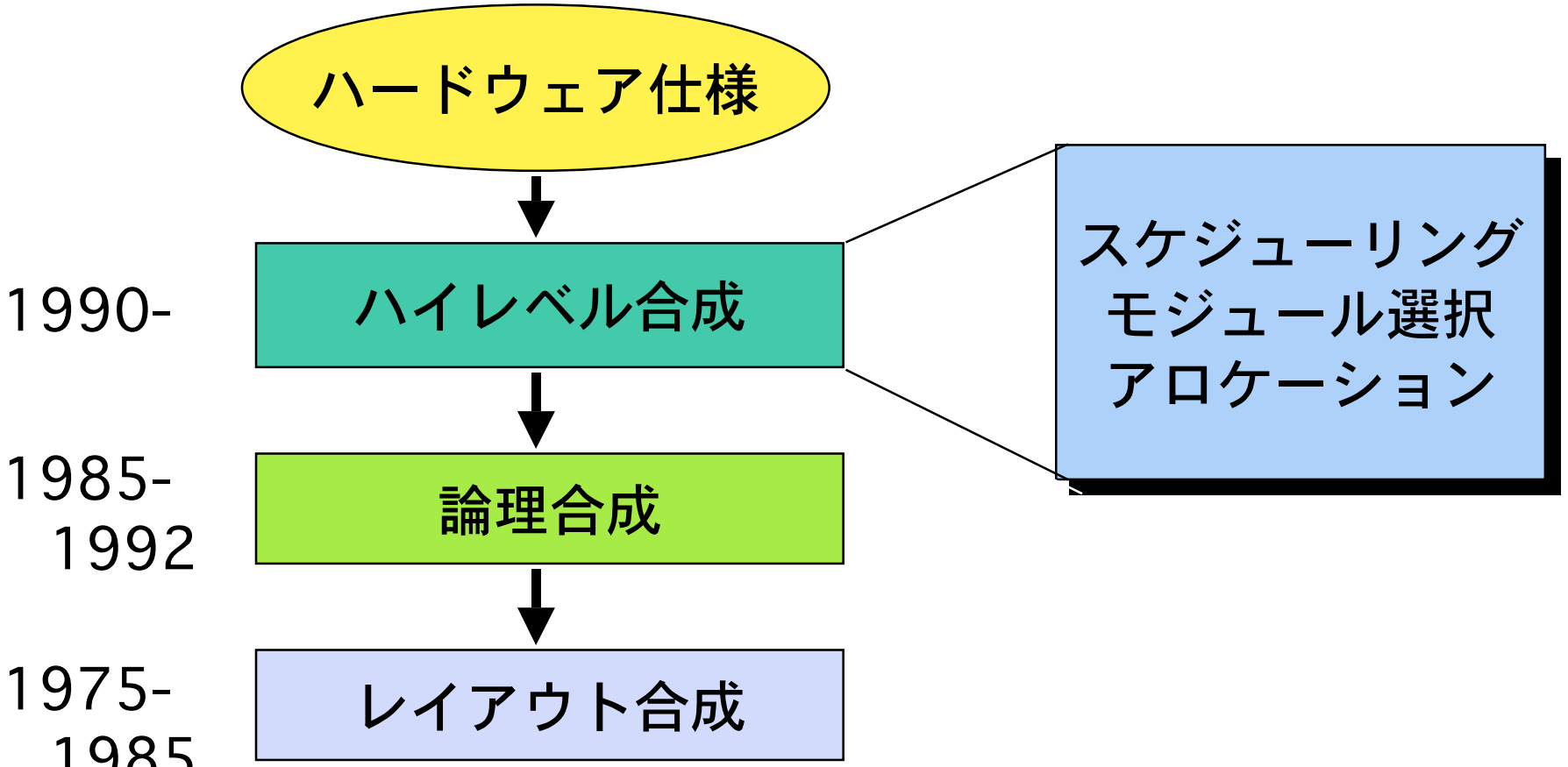
システム設計の要素技術

- システムのモデル化
- 検証技術
  - HW-SW コシミュレーション
  - 検証パタンの生成
  - 形式的検証
- 性能評価技術
- 最適化技術

問題点２

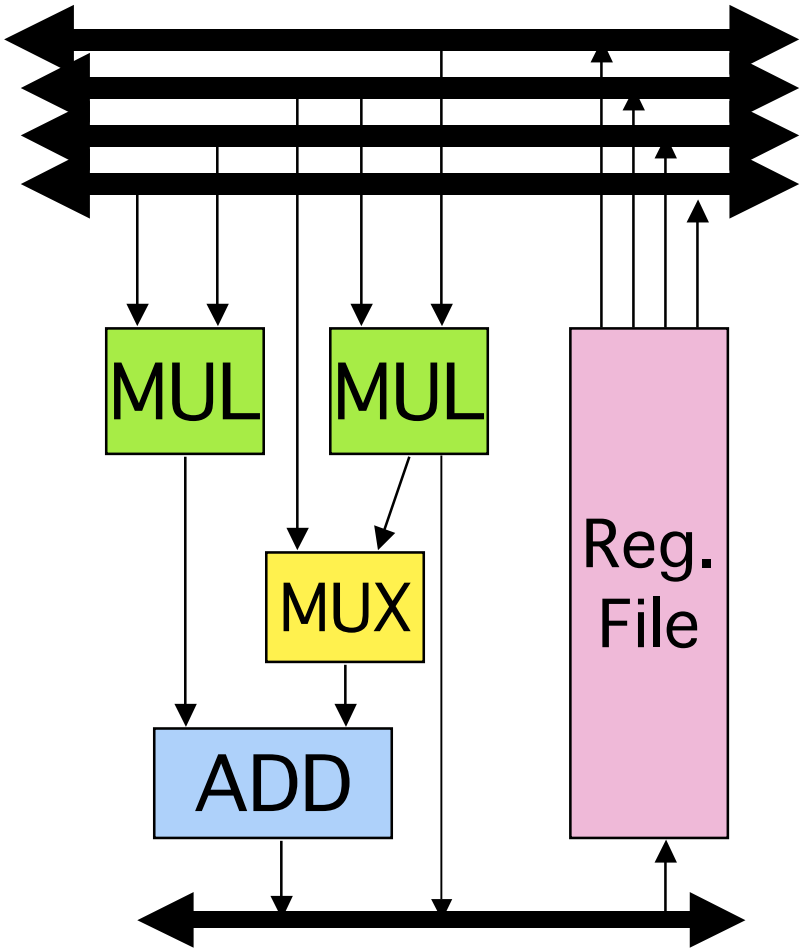
- アーキテクチャ選択とHW/SW分割の順序は？
- 何を固定して，何を設計パラメータとするか？
- 評価関数の選択は？
- 性能を一般的に見積れるか？

ハードウェア合成


ハードウェア仕様
1990-
ハイレベル合成
スケジューリング
モジュール選択
アロケーション
1985-
1992
論理合成
1975-
1985
レイアウト合成

ハイレベル合成

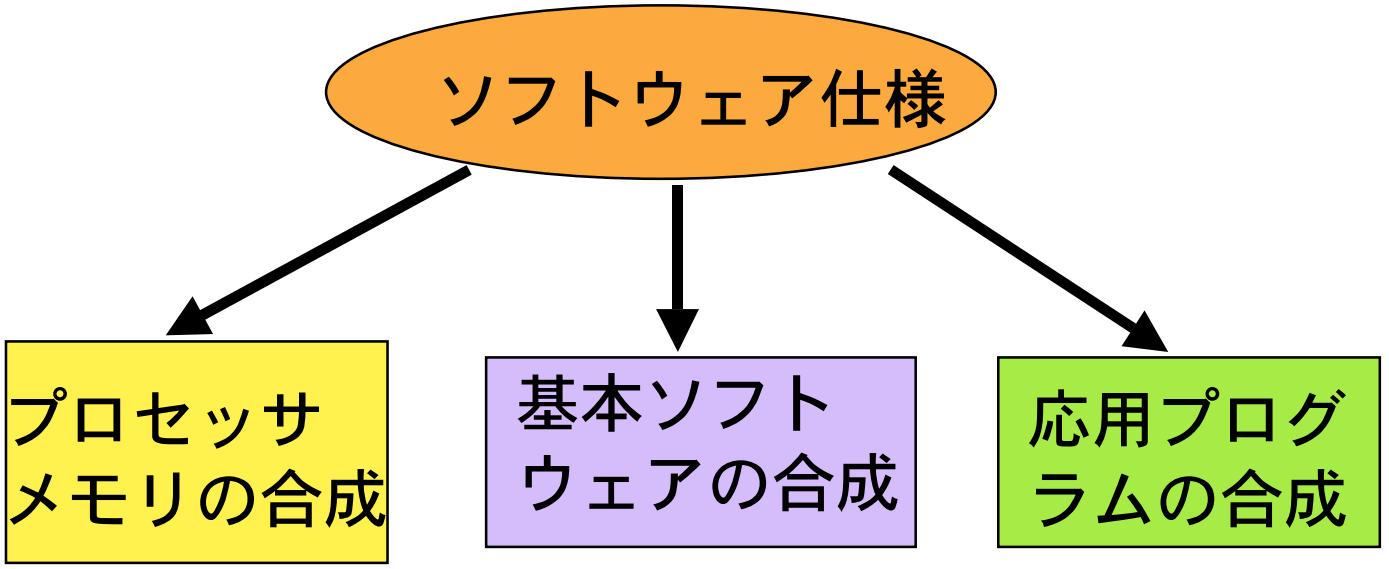
```
for i in 1 to 8 loop
  z:= x(i) * y(i);
  y(i+1) :=a(i) *b(i);
if k=0 then
  s:= z + s;
else
  s:= z + w;
end loop;
```



問題点３

- ハイレベル合成の信頼性
  - 並列化の難しさ
  - セマンティクスのあいまいさ
  - 少ないユーザー
  - ドメイン毎の開発
- 評価関数とその見積り法
  - 面積，性能，消費電力

ソフトウェア合成



汎用／専用
データパス幅
命令体系など

コンパイラ
OS
Debugger

プログラムサイズ
データサイズ
性能，消費電力

基本ソフトウェアの合成

- リターゲッタブルコンパイラ
  - CPU，DSP，専用プロセッサ
    - GCC，Mimola, Valen-C，SUIF,...
- OSの合成

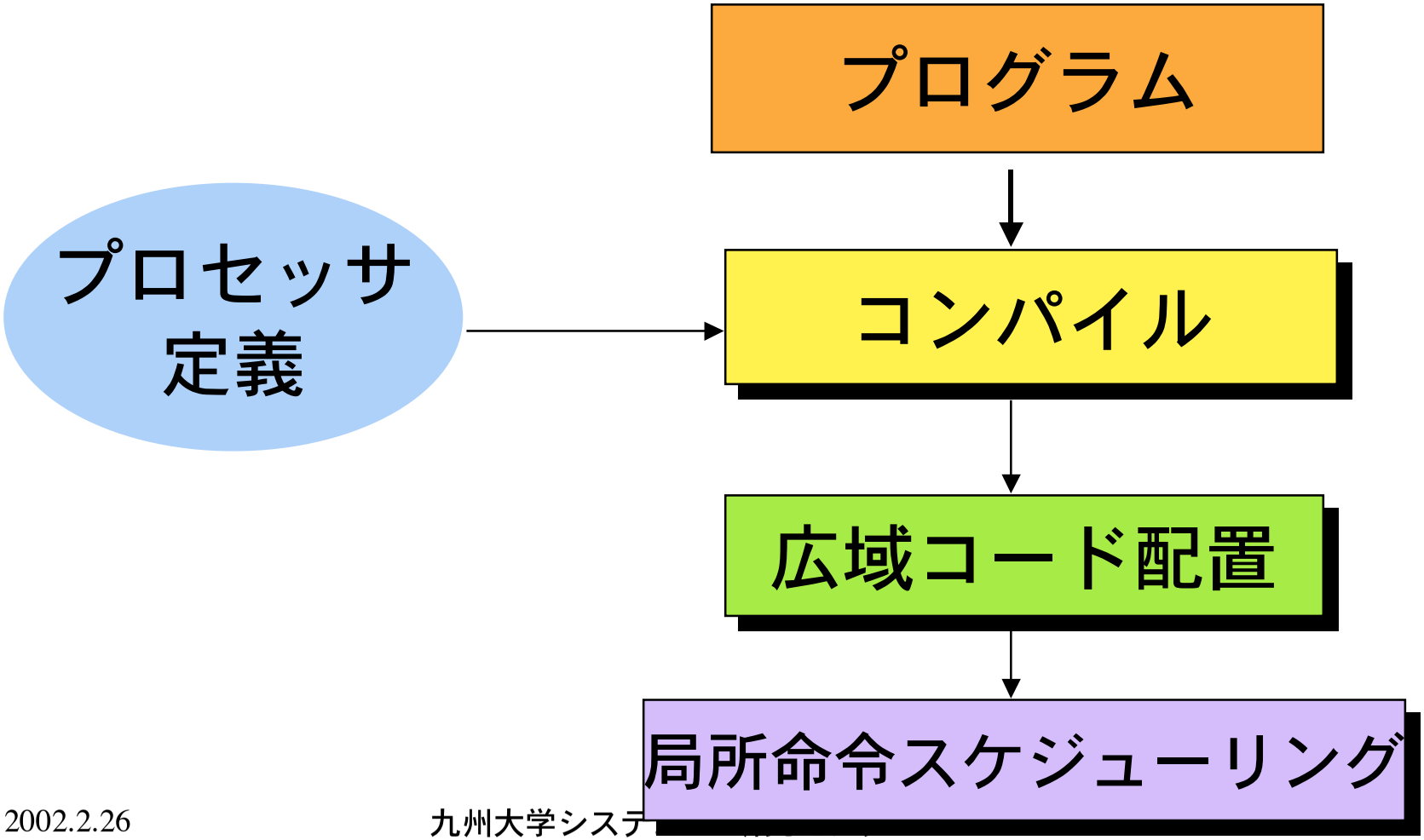
→汎用計算機と異なる条件

- アドレスカウンタの自動インクリメント

組み込みシステムのコンパイル技術

- プログラムはROMの固定される.
- プログラムの実行状況が予測しやすい.
- コンパイルに論理合成やレイアウト合成と同じ程度の最適化努力が許される.
- プログラムに対してハードウェアを変更することも出来る.

コンパイラによるシステム最適化



問題点４

- プロセッサの形式的定義
- 言語の形式的定義
- コンパイラの検証
- OSの検証
- 応用プログラムの最適化の正当性

システムLSI設計における形式的手法への期待

1．システムLSIとは？

2．システムLSIの設計の流れ

3．形式的検証

4．上流設計と形式的手法

5．システムLSIの経済学

6．Quality, Reliability and Security

集積回路のチップコスト

$$COSTchip = (COSTdesign + m\ COSTmask) / Ngood + COSTfab / (Y\ Nwefer) + TESTchip + Pack + IP$$

COSTchip： １チップあたりのコスト
Nwefer： １ウェハ上のチップ数
COSTdesign： 設計費(HW+SW)
TESTchip： １チップあたりのテスト費
COSTmask： マスク１枚の価格
m：新たに必要となるマスクの数
Ngood： 良品の数
Pack： パッケージのコスト
COSTfab： １ウェハあたりの製造コスト
Y： 歩留まり（良品率）
IP：購入IPコスト(HW+SW)

$$Y = (1 - q)^{Achip}$$

$$Nwefer = Awefer / Achip$$

$$TESTchip = TESTdesign / Ngood + V_T Ttest / Y$$

チップコストの概算

COSTchip ＝ (COSTdesign ← 10^6-10^9 （単位円）

20-50 10^5-10^7

＋ m * COSTmask) ／Ngood ← 10^5-10^8

10^4-10^6

＋ COSTfab／（Y Nwefer) ← 10^2-10^4

＋ TESTchip ← 10-10^3

＋ Pack ← 10-10^5 ＋ IP ← 10-10^4

チップコストの例

COSTdesign = 85,000,000円　　m=30枚

COSTmask = 500,000円

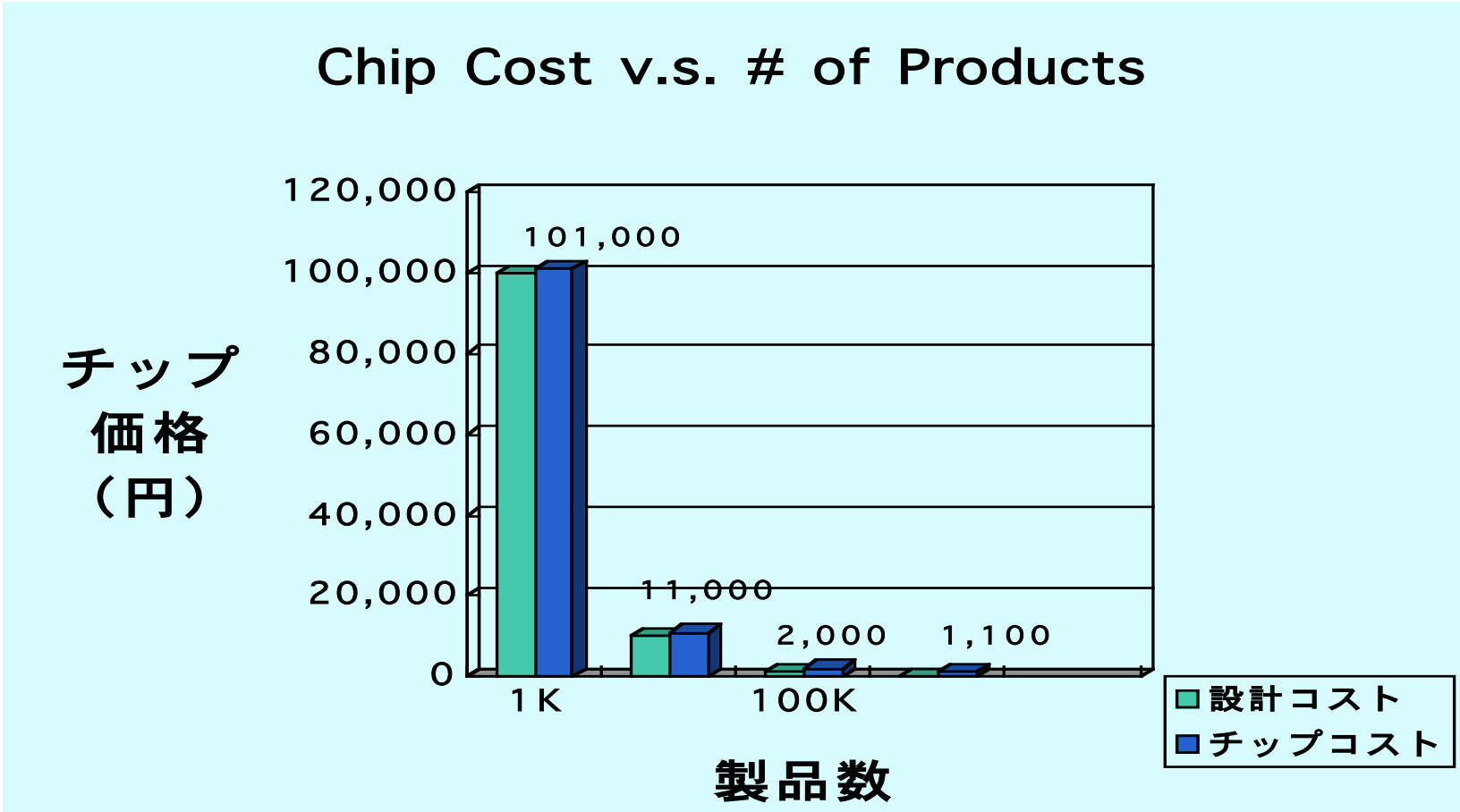
COSTfab = 200,000 円　　Y = 0.8

Nwefer = 500個　　TESTchip = 80円

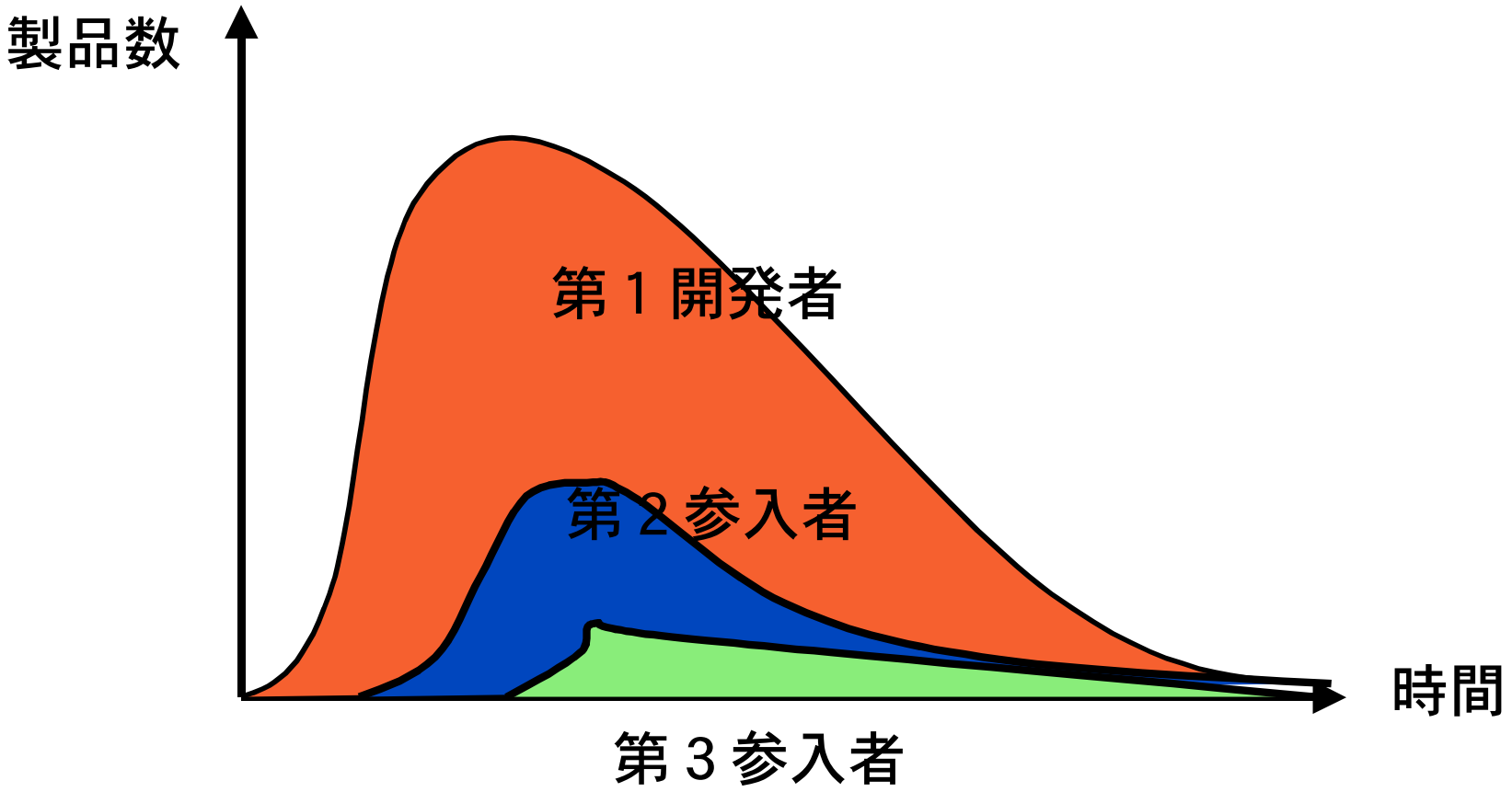
Pack = 300円　　SW = 100円

COSTchip = 100,000,000/ Ngood + 500 + 100
+ 300 + 100
= 100,000,000/ Ngood ＋ 1000（円）

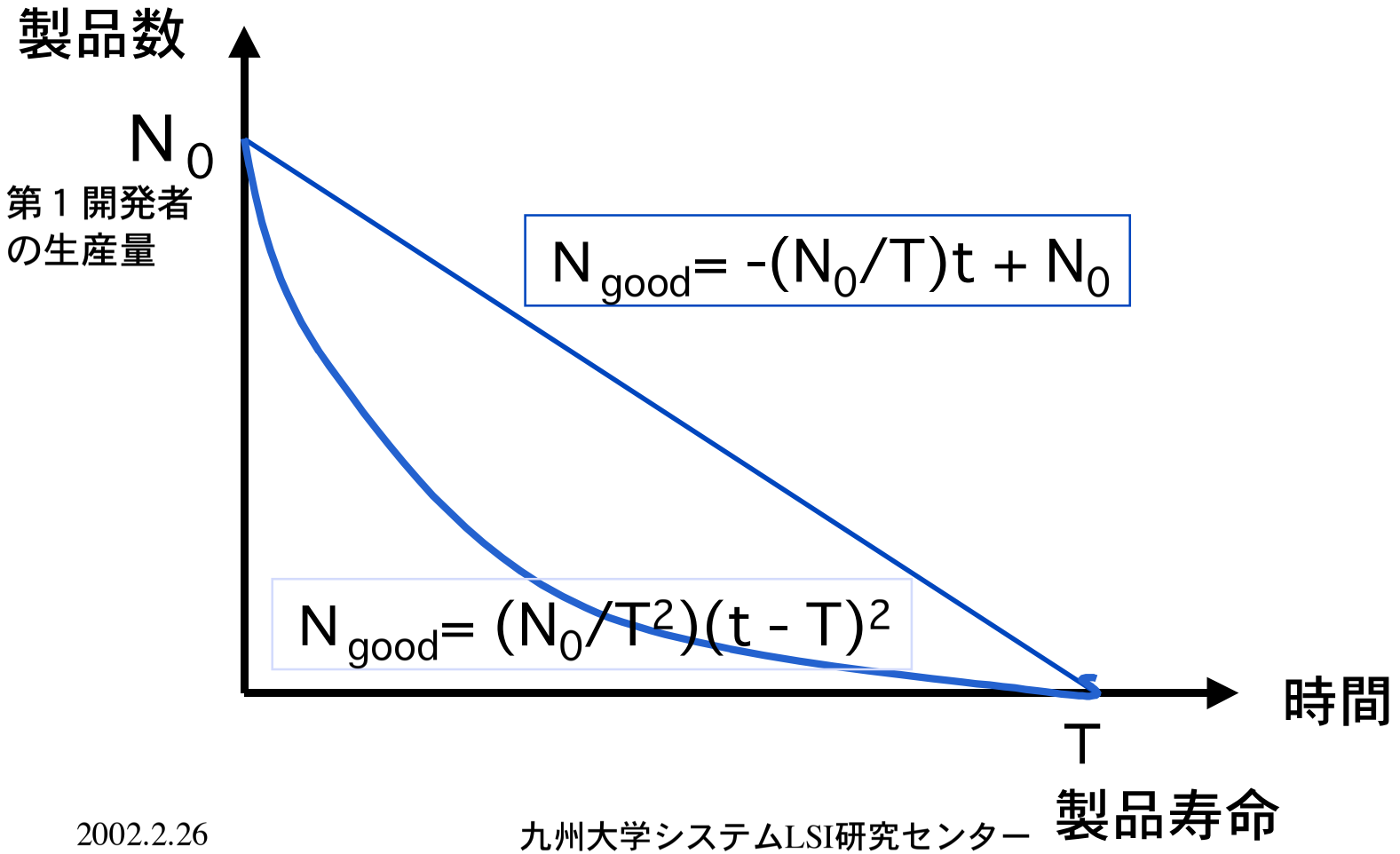
市場規模とチップコスト



開発期間の影響



近似モデル

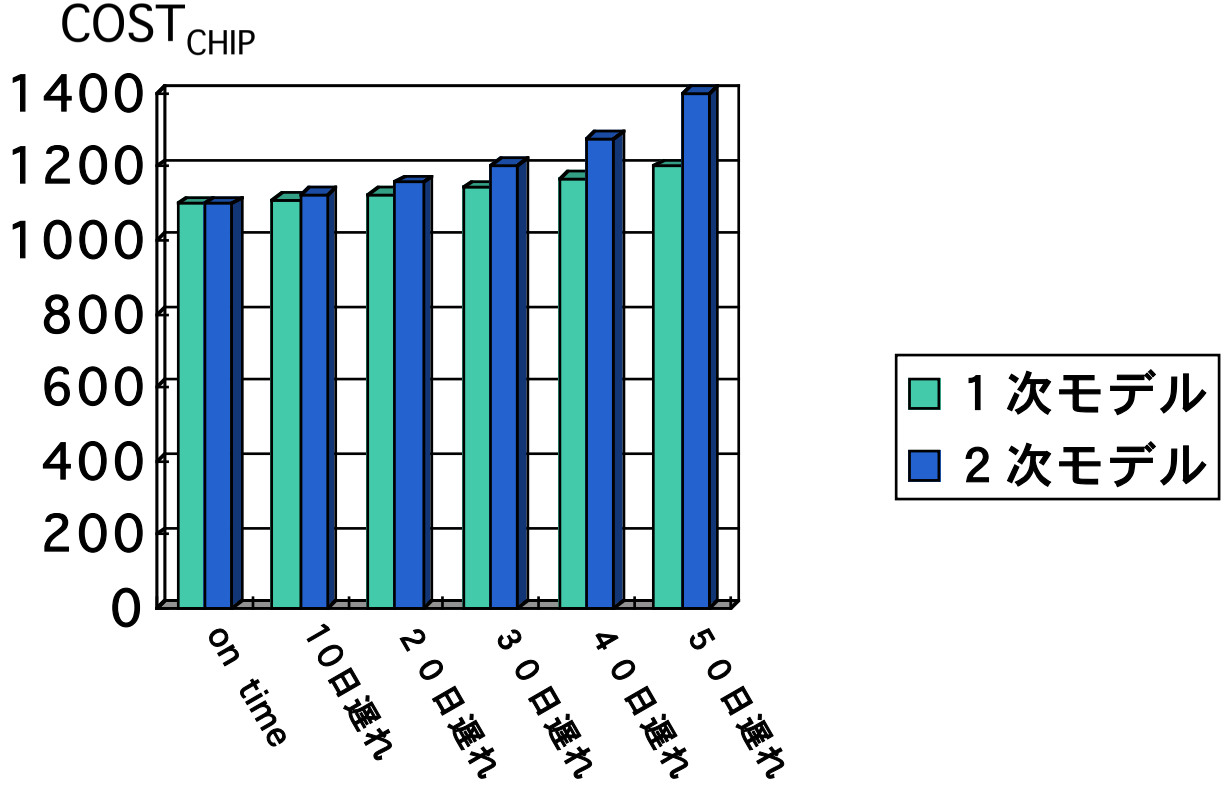


開発期間とチップコスト

COSTchip = 100,000,000/ Ngood ＋ 1000（円）

T = 100日

N_0 = 1M



開発期間と利益

販売価格を１３００円と仮定する.

利益 = Ngood （1300 ー　COSTchip)（円）