ディジタル回路設計の基礎

京都大学
情報学研究科
小林和淑
kobayasi@i.kyoto-u.ac.jp

内容

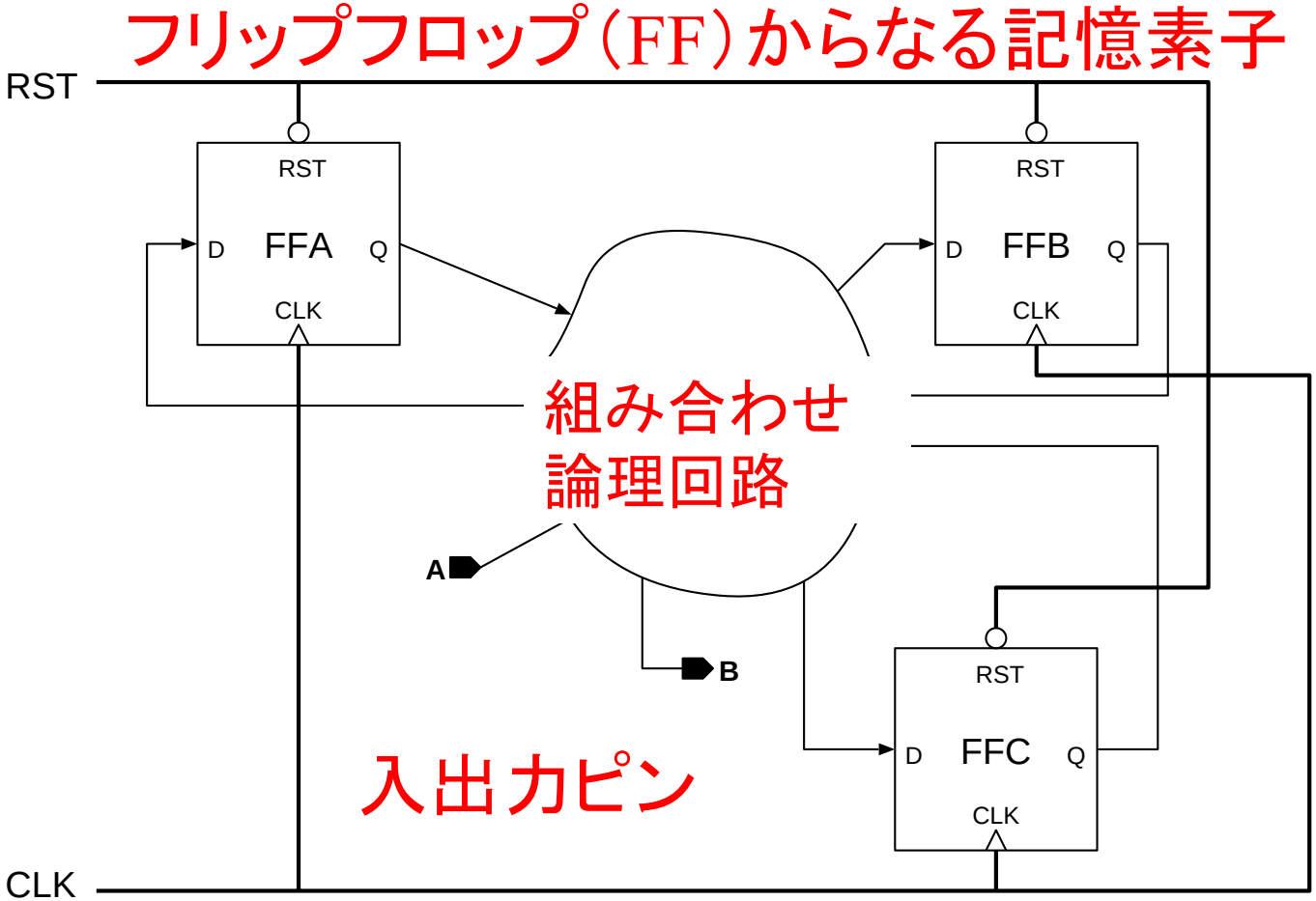
- 単相クロック完全同期回路
  - 構成要素
  - Dフリップフロップ
  - 同期回路の性能
- ハードウエア設計手法
  - 論理設計手法の歴史
  - ハードウエア記述言語
  - RTL設計
- LSIの設計フロー
  - セルベース設計とゲートアレイ
- PLDとFPGA

単相クロック完全同期回路

同期回路とは？

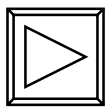
- 時間方向を同期パルス（クロック）により、量子化（ディジタル化）した回路
- クロックにより、クロックとの間のタイミングを考慮するだけでよくなる。
  - クロックがないと、すべての信号の時間関係を考慮して設計を行わなければならない。
- クロックは通常記憶素子（フリップフロップ）に入力される。
  - クロックが入力される毎に、FFの値が変わる

同期回路の構成要素

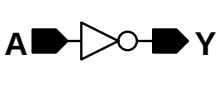
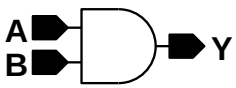
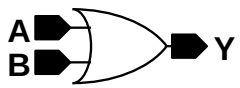
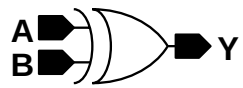

フリップフロップ（FF）からなる記憶素子
RST
RST
D
FFA
Q
CLK
RST
D
FFB
Q
CLK
組み合わせ
論理回路
A
B
RST
D
FFC
Q
CLK
入出力ピン
CLK

単相クロック完全同期回路

- 記憶素子はフリップフロップ(FF)のみである。
- 外部から単一のクロックが与えられる。
- このクロックの立ち上がりもしくは立ち下がりエッジのどちらか一方にすべてのFFが同期して動作する。
- 通常の同期設計では、立ち上がりエッジに同期するFFと立ち下がりエッジに同期するFFが混在してはならない。
- 非同期リセットは通常、電源投入時のみ使用する。



基本論理ゲート

名前		NOT		AND				OR				XOR			
MIL 記号															
入力	A	0	1	0	0	1	1	0	0	1	1	0	0	1	1
	B	-	-	0	1	0	1	0	1	0	1	0	1	0	1
出力	Y	1	0	0	0	0	1	0	1	1	1	0	1	1	0
Verilog-HDL		Y=~A		Y=A&B				Y=A | B				Y=A^B			

- 2進数の各種演算を実行するための基本演算
- 複合演算を実現する「複合ゲート」
  - ANDNOR等

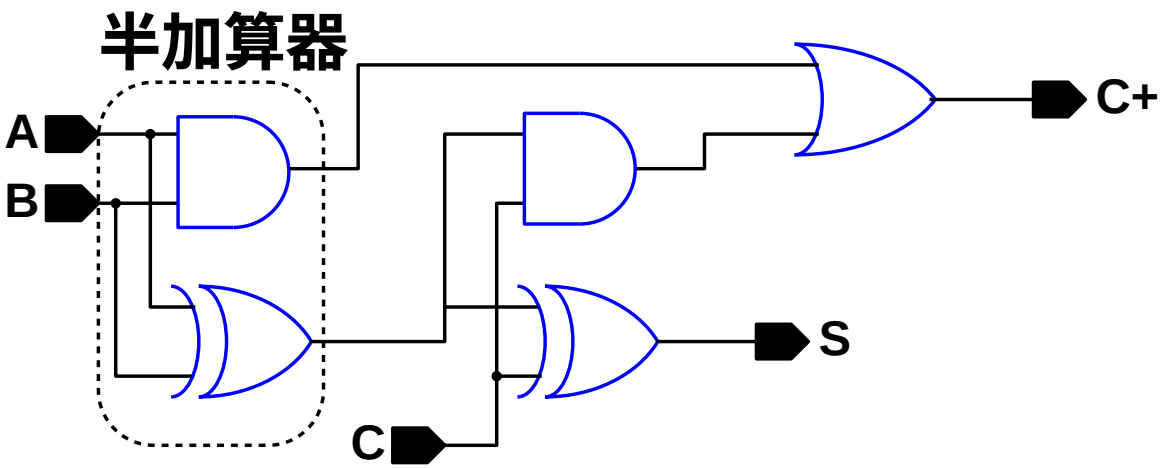
組み合わせ論理回路の例　全加算器

A	B	C	S	C+
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

$$S = A \oplus B \oplus C$$

$$C+ = A\&B|B\&C|C\&A$$

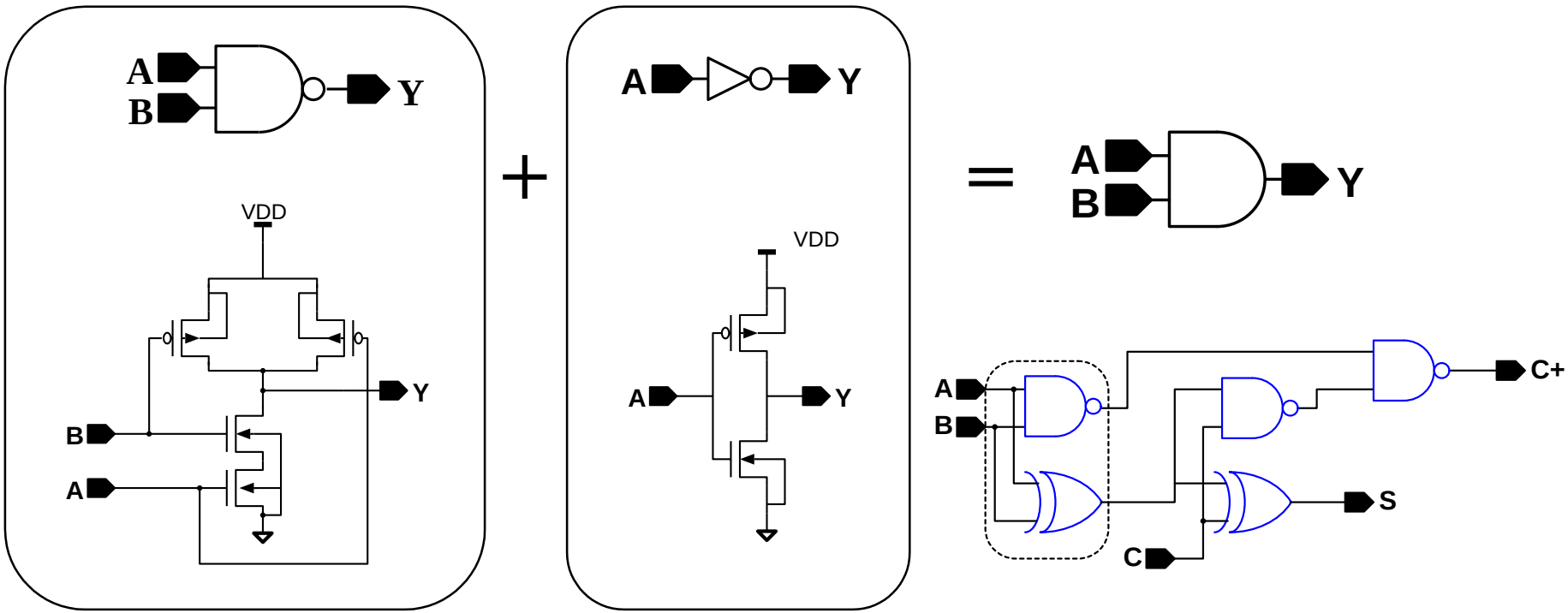
真理値表と論理式



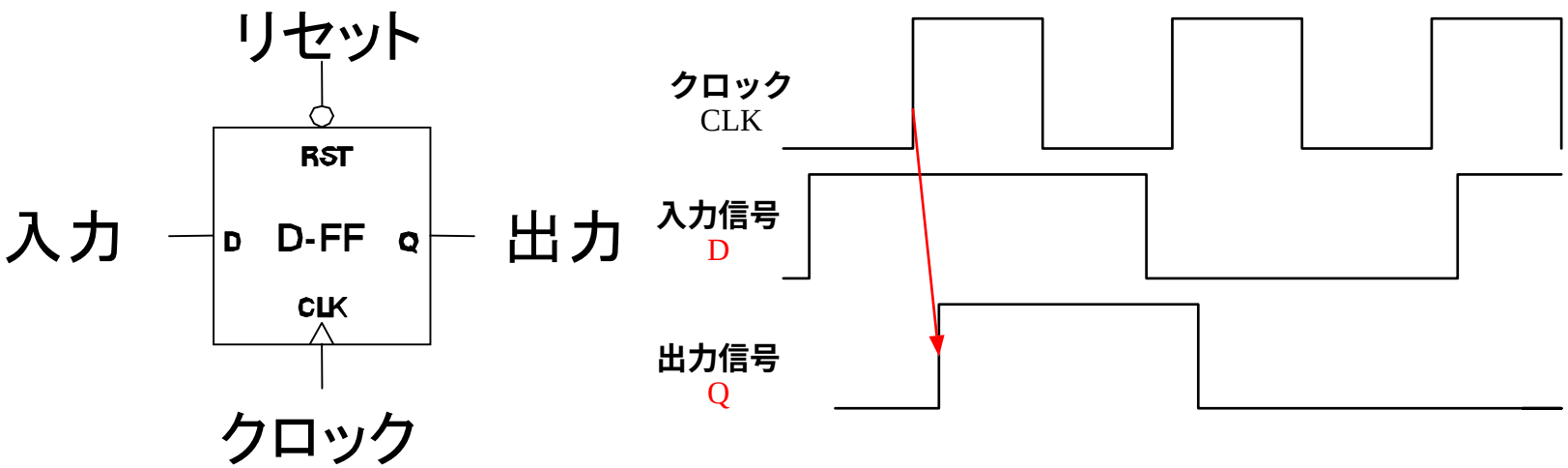
回路図

LSI中のCMOS基本論理ゲート

- CMOS論理ゲートは負論理(NOT, NAND, NOR)が基本



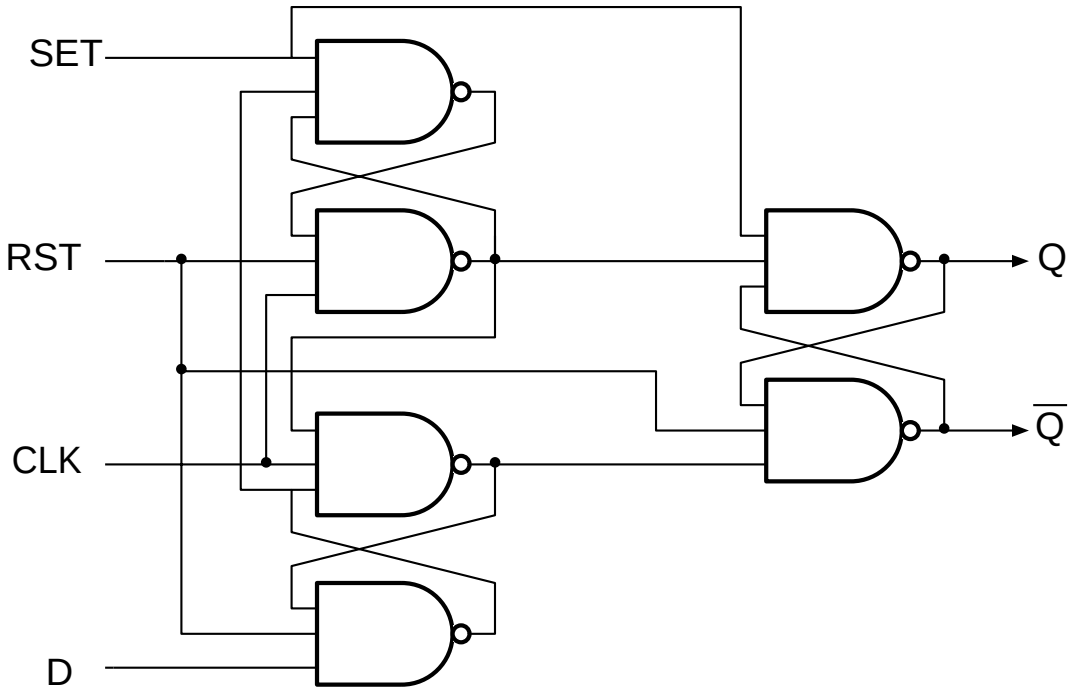
Dフリップフロップ



- 入力の変化がクロックの立下り、または立ち上がりにより出力に伝わる。
- 通常、リセットはクロックとは非同期

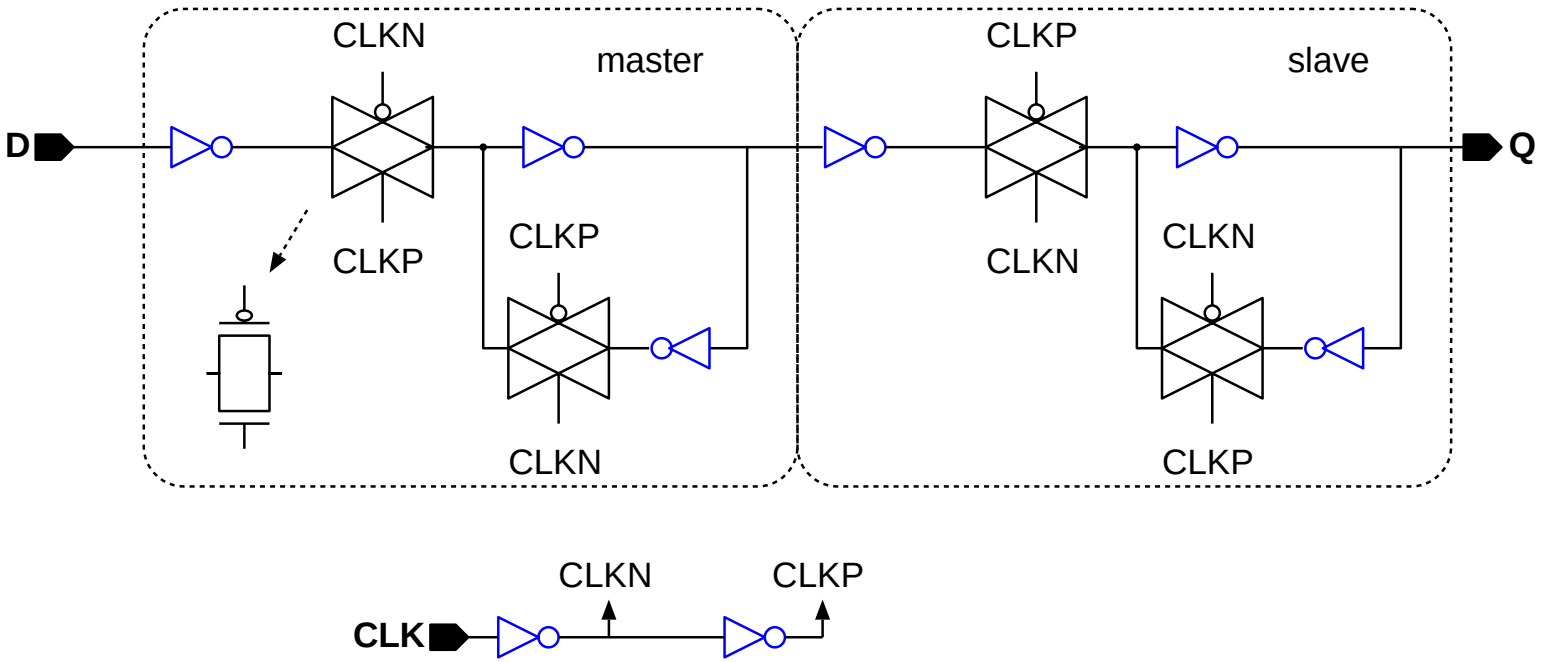
CMOS論理ゲートを用いたD-FF

- 面積が大きくなるので、LSI中では使用されない



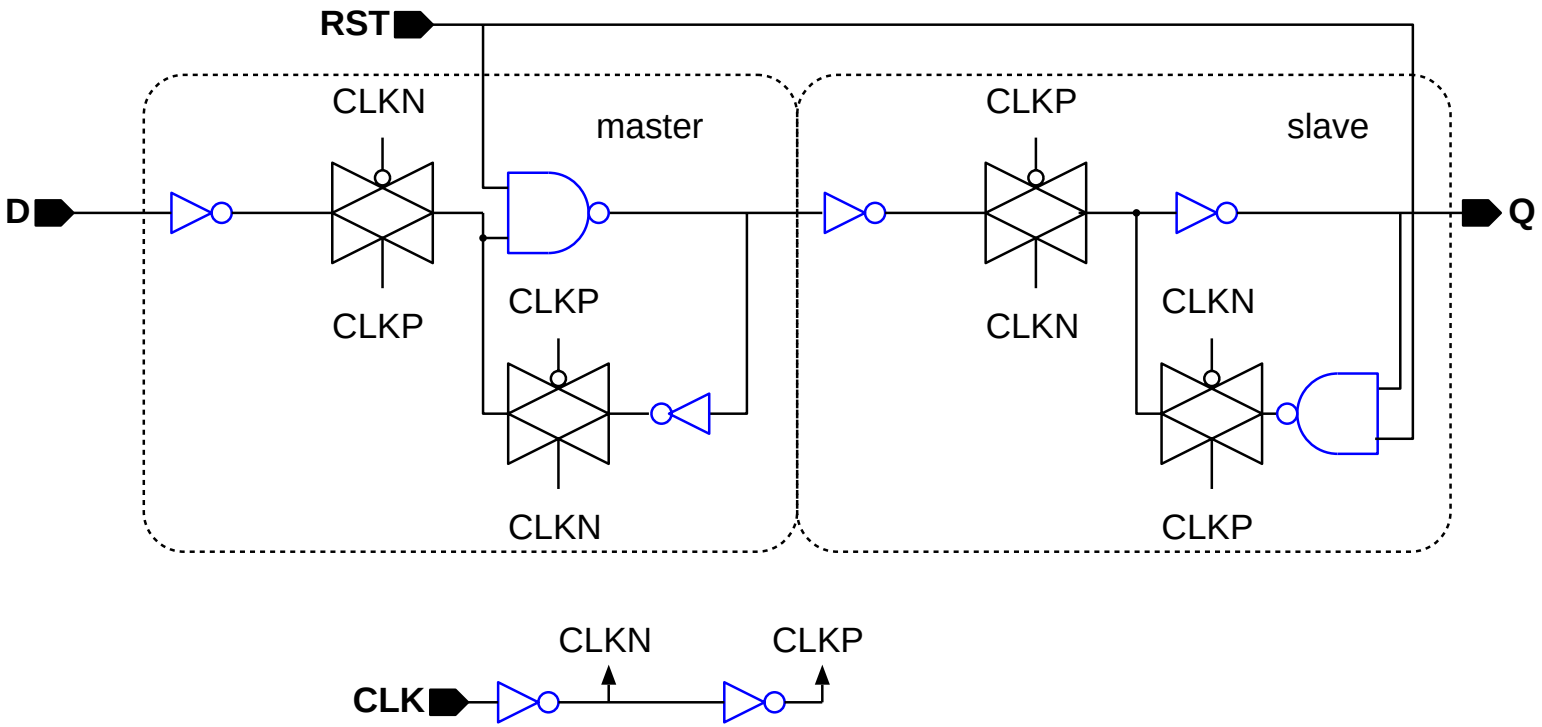
Dフリップフロップの回路構造

- 集積回路中のD-FFは, マスターとスレーブの2個のD-Latchを接続した構造をしている。



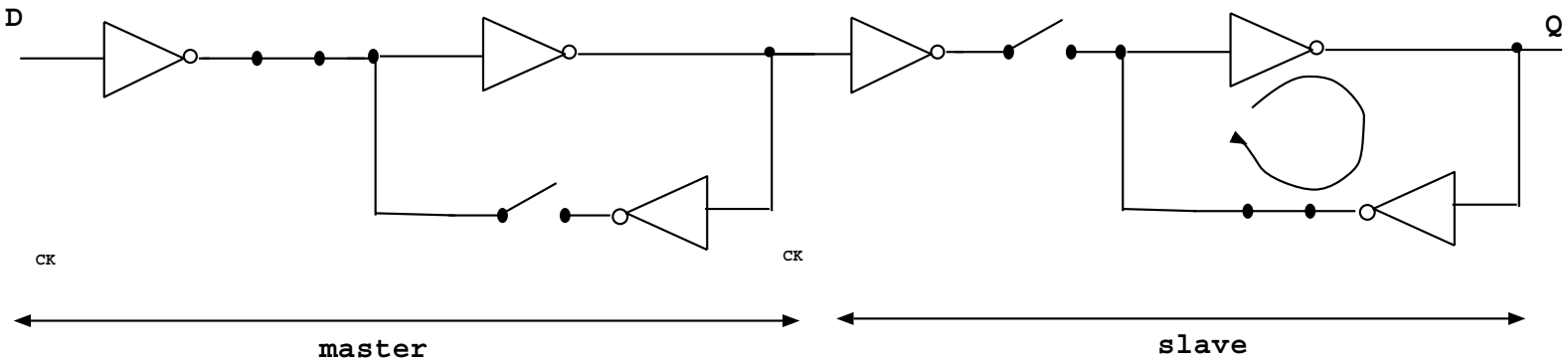
ポジティブエッジトリガ型Dフリップフロップの一例(リセットなし)

Dフリップフロップの回路構造(続き)

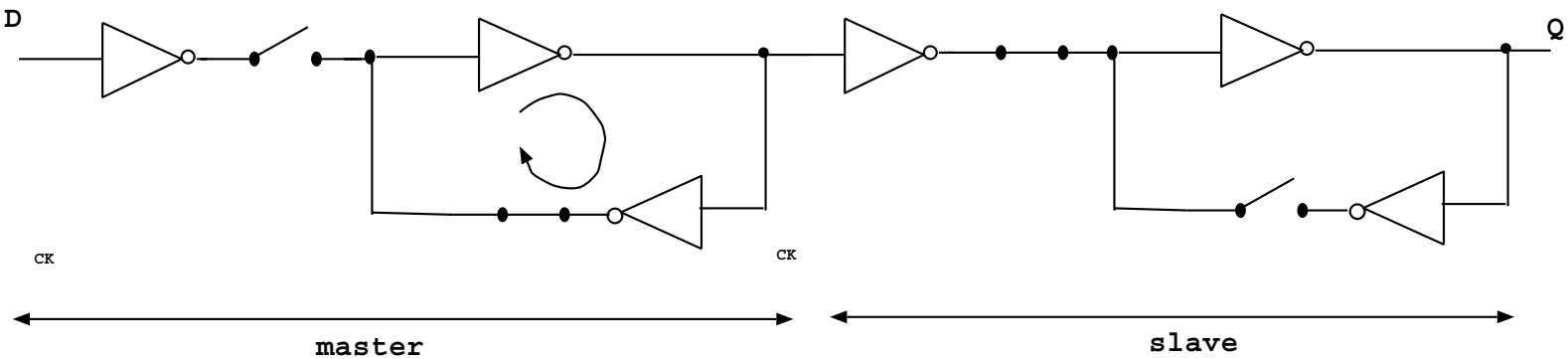


D-FF 非同期リセットつき

Dフリップフロップの動作

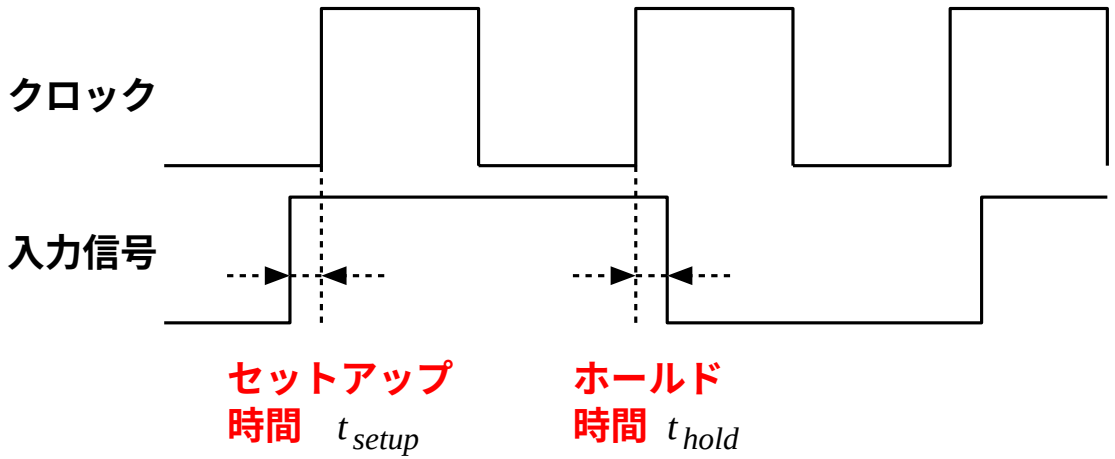


マスター動作時（CLK=0)



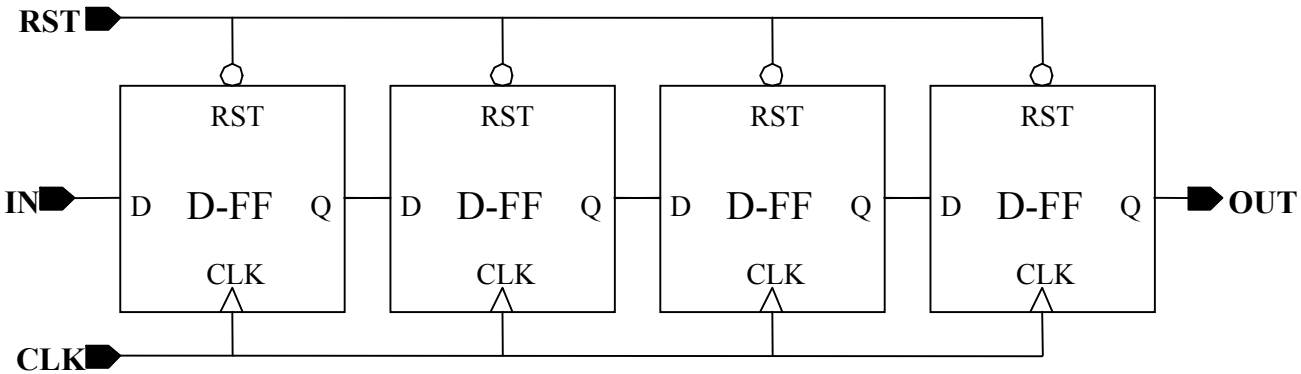
スレーブ動作時（CLK=1)

セットアップとホールド時間

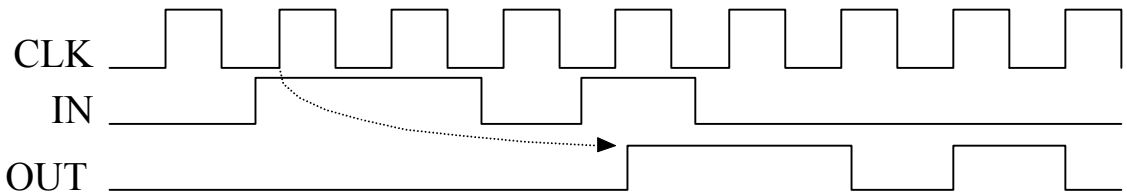


- セットアップ、ホールド時間を守らないと、誤動作（ハザード）を起こす場合がある
- 先ほどの回路構造からセットアップ時間、ホールド時間がどの部分の遅延に依存するか考えてみよう。

DFFによる遅延素子

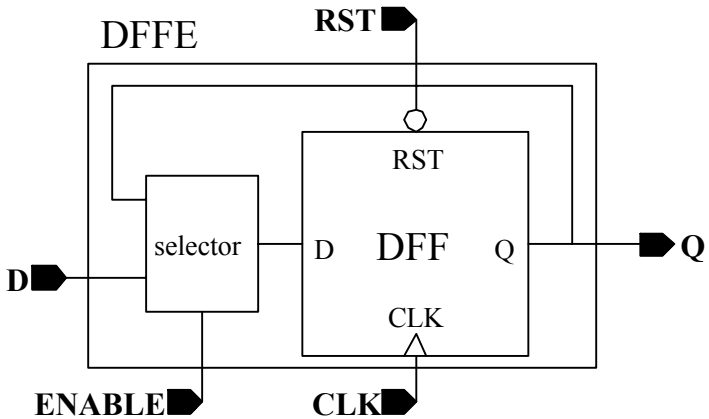


D-FFを4個つなげば、4クロック遅れる遅延素子に



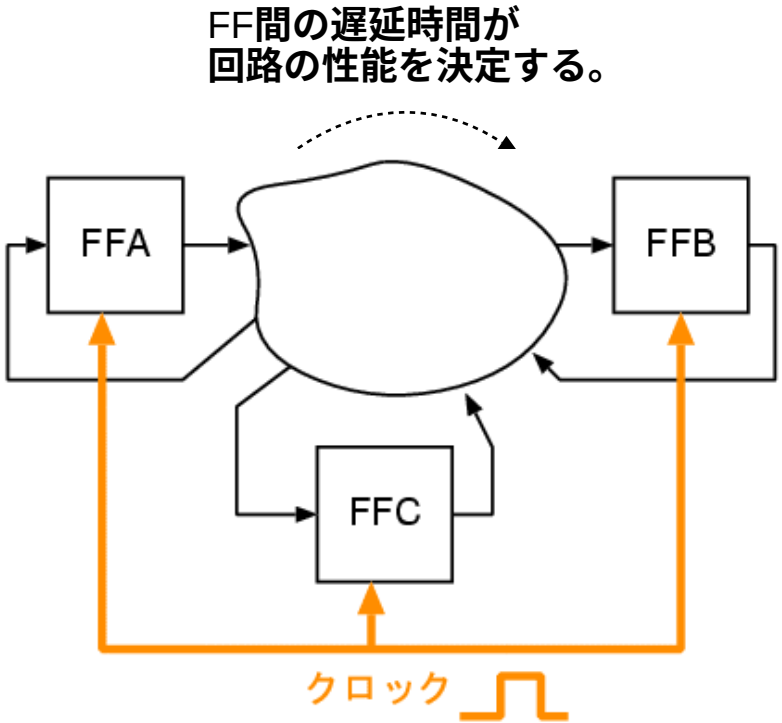
- 遅延させるだけでは、何もできない。
- 記憶を行わなければならない。

イネーブルつきDFF （DFFE)



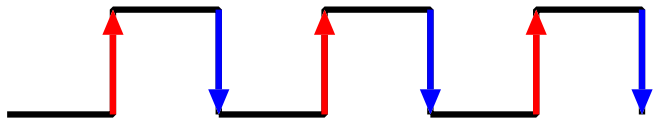
- ENABLE=1のとき、Dの入力を取り込む。
- ENABLE=0のときは、Qを保存
- 値の記憶に使う

同期回路の性能



- FF間の遅延を削減すれば、性能（クロック周波数）があげられる。
  - クリティカルパス： 最大遅延パス
- FFへのクロック供給をしっかり行い、クロックのずれ（スキュー）をなくすことが大事
  - クロックツリー等

クロックの両エッジ

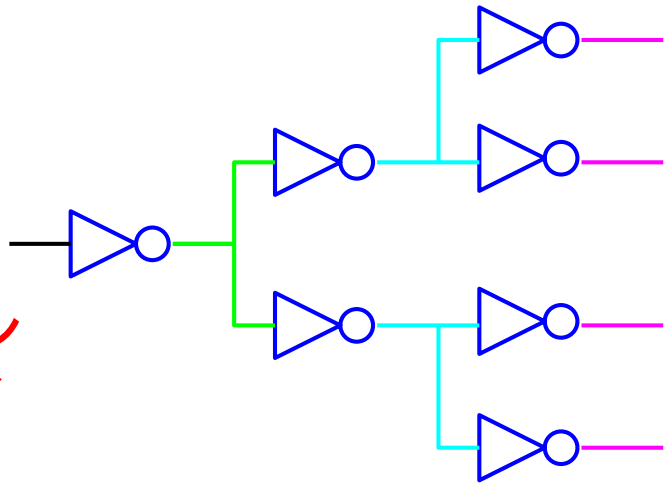


- 設計者の陥る誘惑
  - クロックの両エッジを使うと、1クロックで倍の仕事ができる
- 両エッジを使うのは、誤動作の温床
  - 周波数は変わらないが、デューティー比は簡単に変わる
  - 両エッジを使うなら、クロック周波数を倍にせよ
  - デューティー比をコントロールできれば、使用することも可

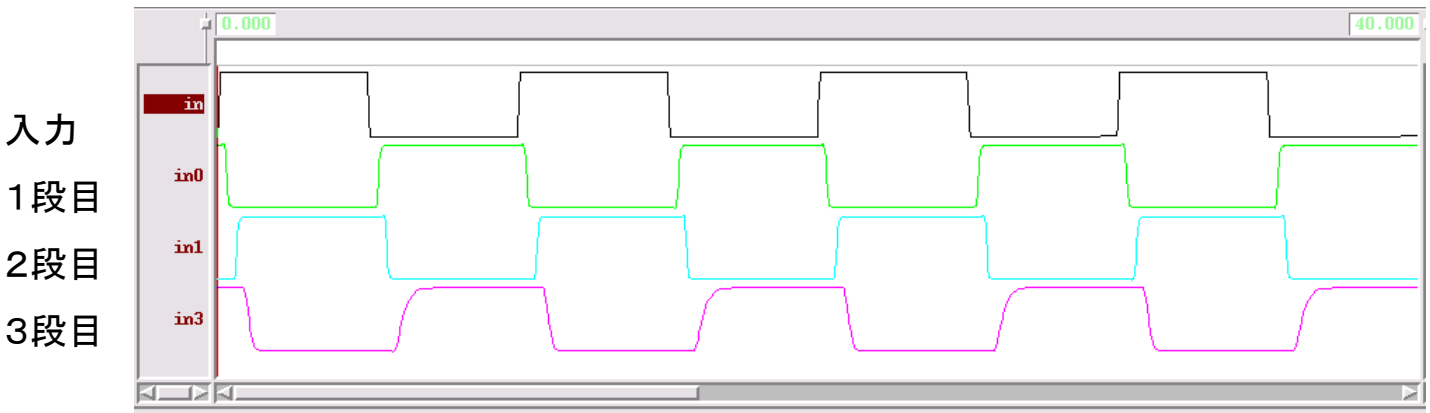
クロックのデューティー比

入力デューティー比　　50:50

3段目デューティー比　48.8:51.2



両エッジを使うと、さらに余分なインバータが入り、クロックの位相がずれる。



ハードウエア設計手法

ハードウエア設計手法の歴史

- 50代： 頭で考えて、紙に書き、TTLで実装
- 40代： 回路図をCADで描いて、LSIで実装
- 30代： HDLより論理合成 ← ここを説明
- 20代： さらに高級な言語（C, C++等）より、直接回路を合成（システムレベル記述言語）

回路規模(ゲート数)

ゲート数に比例して回路設計の抽象度がどんどんあがっている

回路設計のつぼ

- 回路設計技術の向上が目覚しいが、動作から自動的に最適な回路を作る技術はまだまだ
- 人間による最適な回路構造の設計が重要
- 3つの力
  - 動作から、回路をおこす想像力
  - よりコンパクトで高速な回路を考える技術力
  - さまざまなツールを使いこなす応用力
    - » CADツール間のフィルタ記述(perl, awk, sed等）
    - » WindowのボタンをクリックしているだけではよいLSIはできない

HDLの目的

- 目的とする回路の機能を人間のわかりやすいようにテキストで記述して，その記述から自動的に回路記述を生成する。

HDL記述→回路	論理合成
高級言語→回路	高位合成

動作記述とRTL記述

- 動作記述
  - 回路の動作を記述する
  - クロックがない
  - ソフトウエアとほぼ同じ
- RTL（Register Transfer Level)記述
  - レジスタ（記憶素子、FF）間の接続関係を表現したもの
  - そのまま論理合成に使える事が多い

HDLの種類
Verilog-HDLとVHDL

- Verilog-HDL: Cadence社の論理シミュレータ用言語から派生
- VHDL: 回路仕様を書くことを目的に、標準化

どちらも、論理合成を目的として開発した言語ではない。

- 記述できるが、合成できない。
- シミュレーションできるが合成できない、合成しても正しく動作しない。

Verilog-HDLとVHDL（Cont.）

- Verilog-HDL:
  - 抽象度が低い
  - 回路的
  - 電気系向き
- VHDL:
  - 抽象度が高い
  - プログラム的
  - 情報系向き

S0, S1, S2の状態へのマッピング

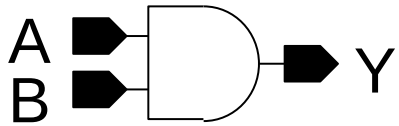
```
`define S0 2'b00;
`define S1 2'b01;
`define S2 2'b10;
```

ビット数と割り当てを明示

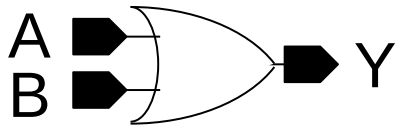
```
type state is (S0, S1, S2);
```

割り当ては合成任せ

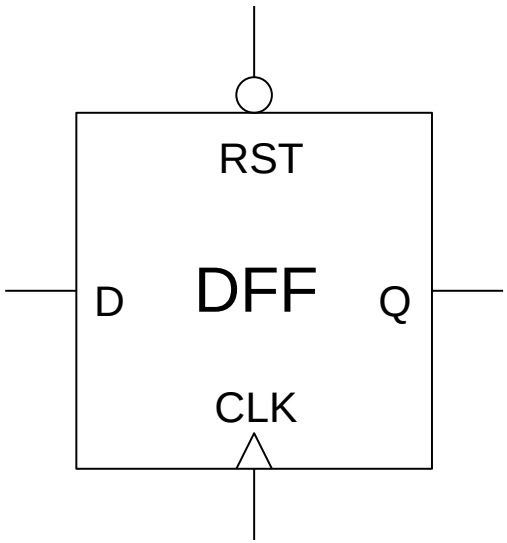
Verilog-HDLによる論理素子



assign Y=A&B;



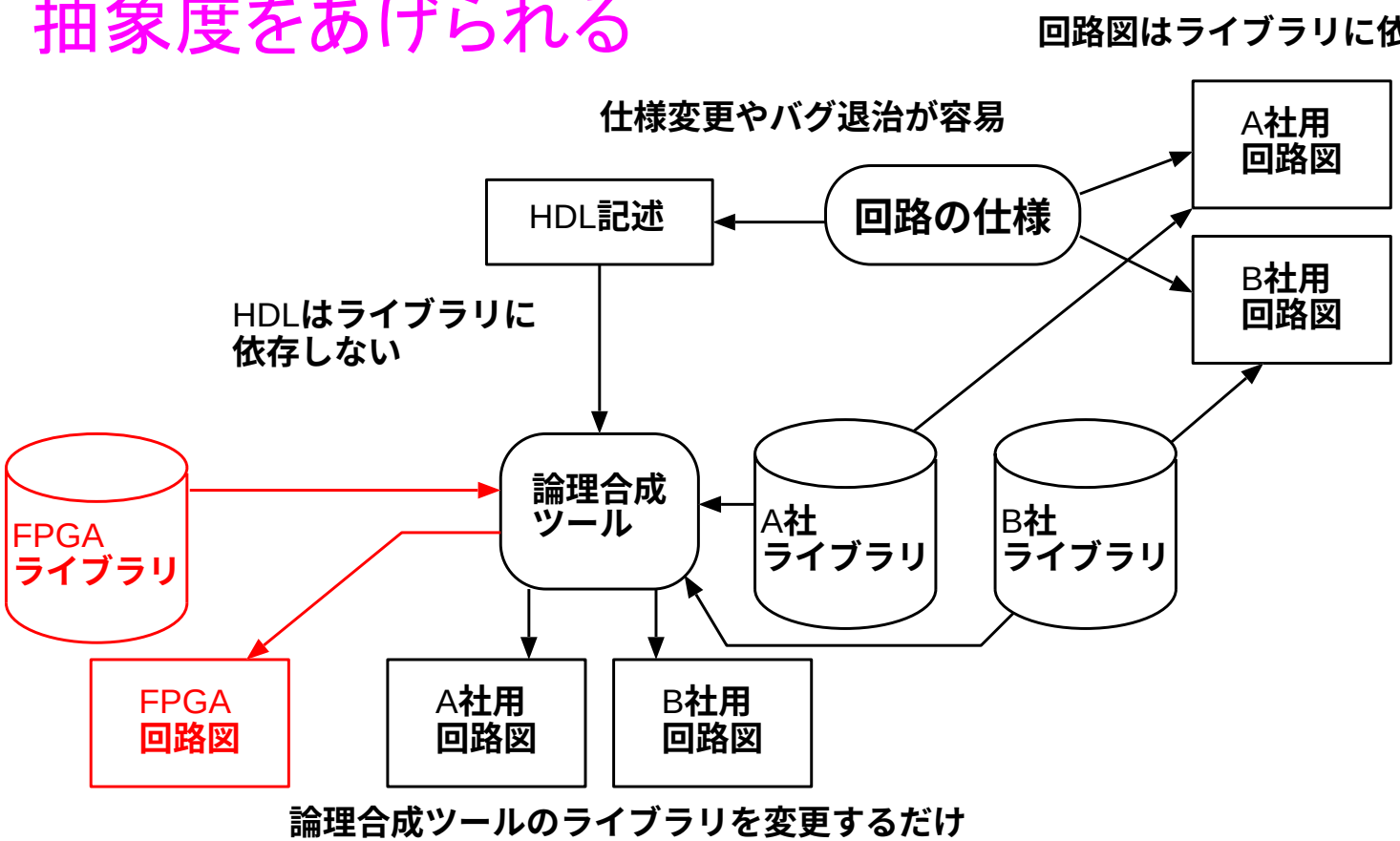
assign Y=A|B;




```
always @(posedge CLK or
                        negedge RST)
    if(!RST) Q<=0;
    else     Q<=D;
```

HDLを使うメリット

抽象度をあげられる



HDLを書くための準備

- 回路中のフリップフロップ，レジスタ等の記憶素子の構成を決める。
- それらフリップフロップ，レジスタをどのように接続するかを考え，ブロック図を書く。
  - レジスタ間に存在する組み合わせ回路の詳細は考える必要はない。
  - レジスタの動作を記述する。(RTL設計）
- どのような回路を意図するか？が重要
  - ブロック図を書いて意図した以外のFF，レジスタが入るのを防ぐ。

HDL設計の例（カウンター）

記述A

```
module counter(out,CLK);
  input CLK;
  output [3:0] out;
  DFF DFF0(q0,d0,CLK);   DFF
  DFF DFF1(q1,d1,CLK);
  ....
  assign d0=q0&~q1|....;組み合わせ
  assign d1=....;              論理回路
endmodule
```

記述B

```
module counter(out,CLK);
  input [3:0] in;
  input CLK;
  output [3:0] out;
  reg [3:0] out;
  always @(posedge CLK)
    out<=out+1;
endmodule
```

ただ単に、回路図をテキストで書いただけ

レジスタ（フリップフロップ）の動作を記述する

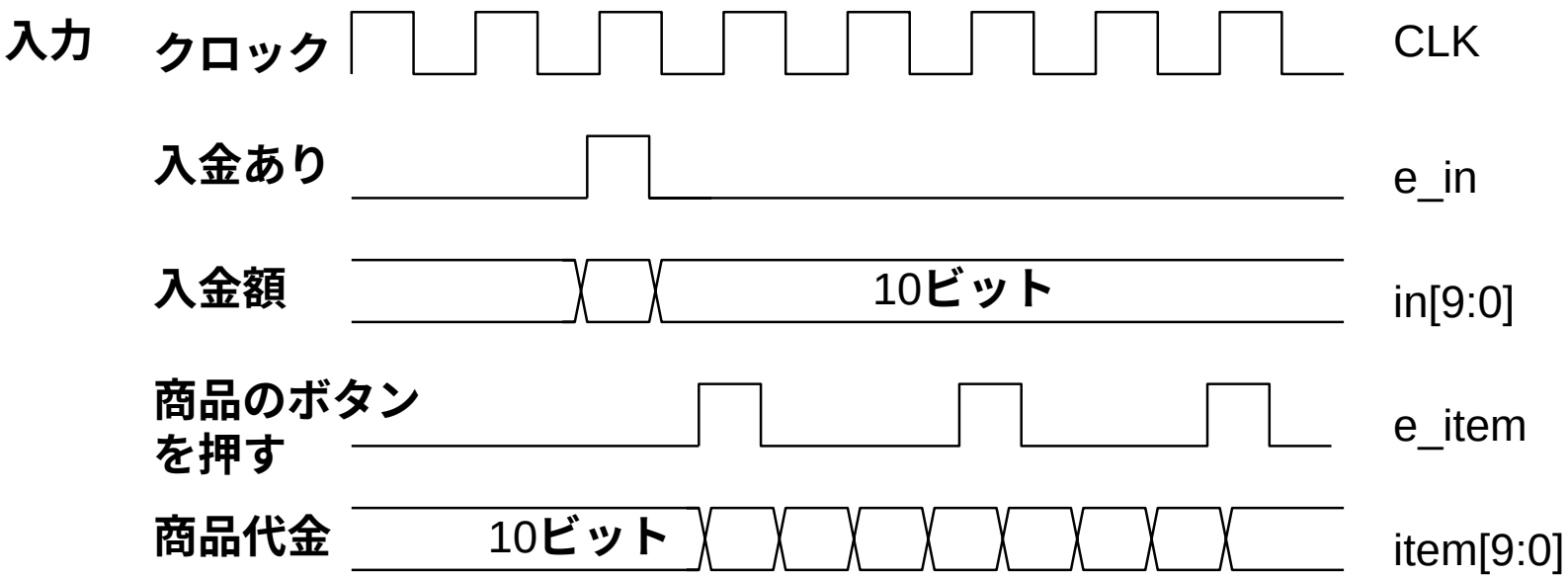
RTL設計の例

例題：　自動販売機の入金額と商品購入額よりお釣りを計算する。

動作

入力：　入金額（1回のみ）、購入商品代金（複数）
出力：　お釣り

自動販売機お釣り計算



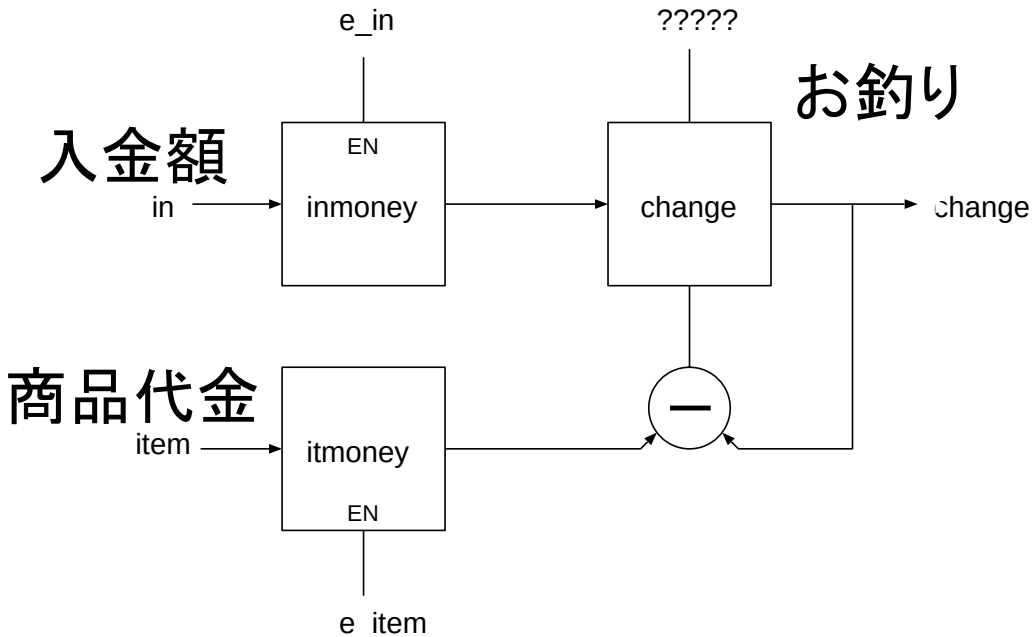
◆入金後商品のボタンを押せば、その金額が送られてくる。

◆入金があったときと、商品のボタンを押したときには、それに同期したパルスが入力される。

RTL設計

- 使用するレジスタを決める
- レジスタ間の接続を決める。
- レジスタに書き込む条件を決める。

使用するレジスタ
inmoney[9:0]　入金額
itmoney[9:0]　商品代金
change[9:0] お釣り



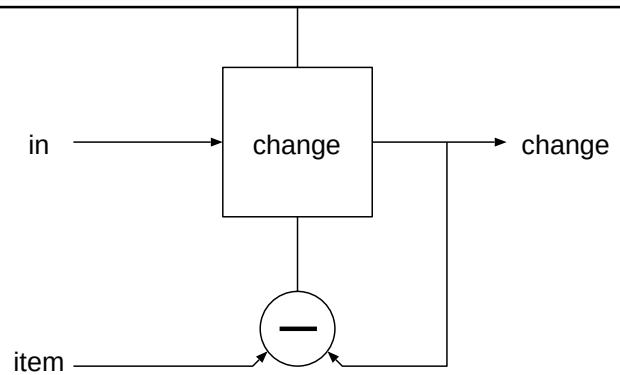
Changeに書き込む条件？

RTL設計の最適化

- 入金額、商品代金は覚えておかなくてよい。
- 制御が簡単なように、RTLを変更する。

使用するレジスタ
change[9:0] お釣り

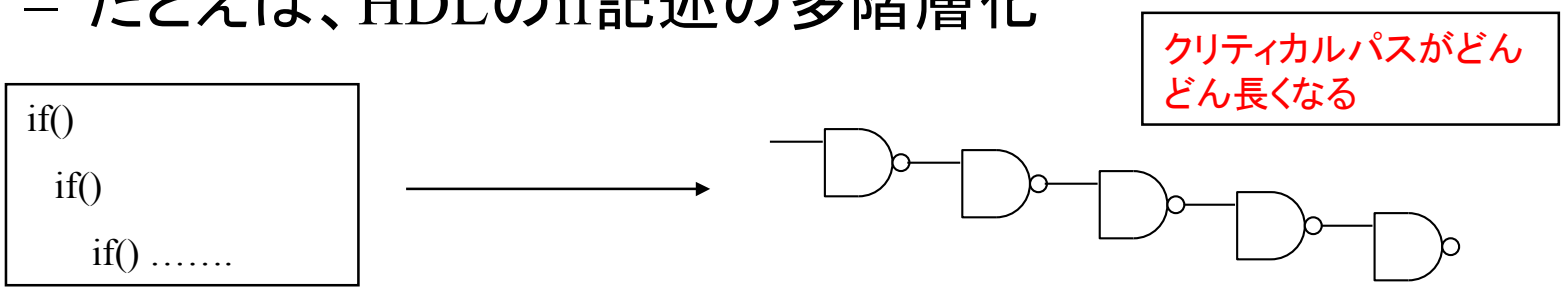
e_inが1ならin, e_itemが1なら減算結果を書き込む




```
Verilog-HDL記述
module vend(change,e_in,e_it,in,item,CLK);
  output [9:0] change;
  input  e_in,e_it,CLK;
  input [9:0] in,item;
  reg [9:0] change;
  always @(posedge CLK)
    if(e_in)
      change<=in;
    else if(e_it)
      change<=change-item;
endmodule''
```

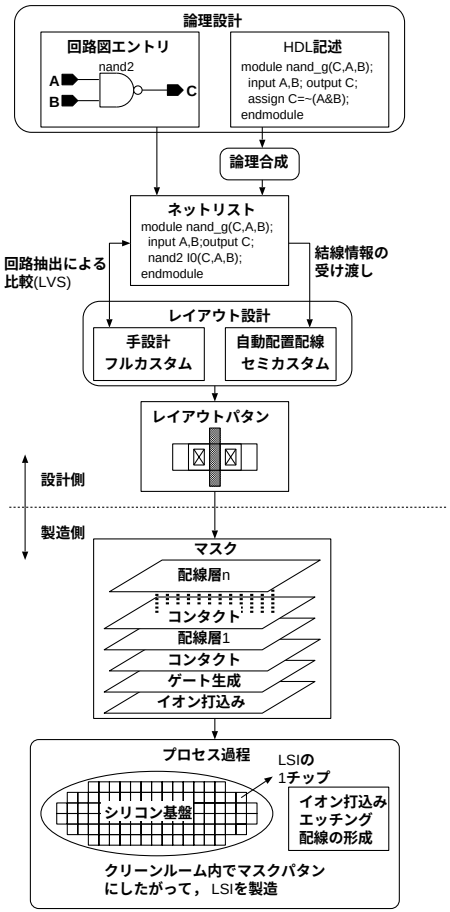
最適なRTLを設計するには

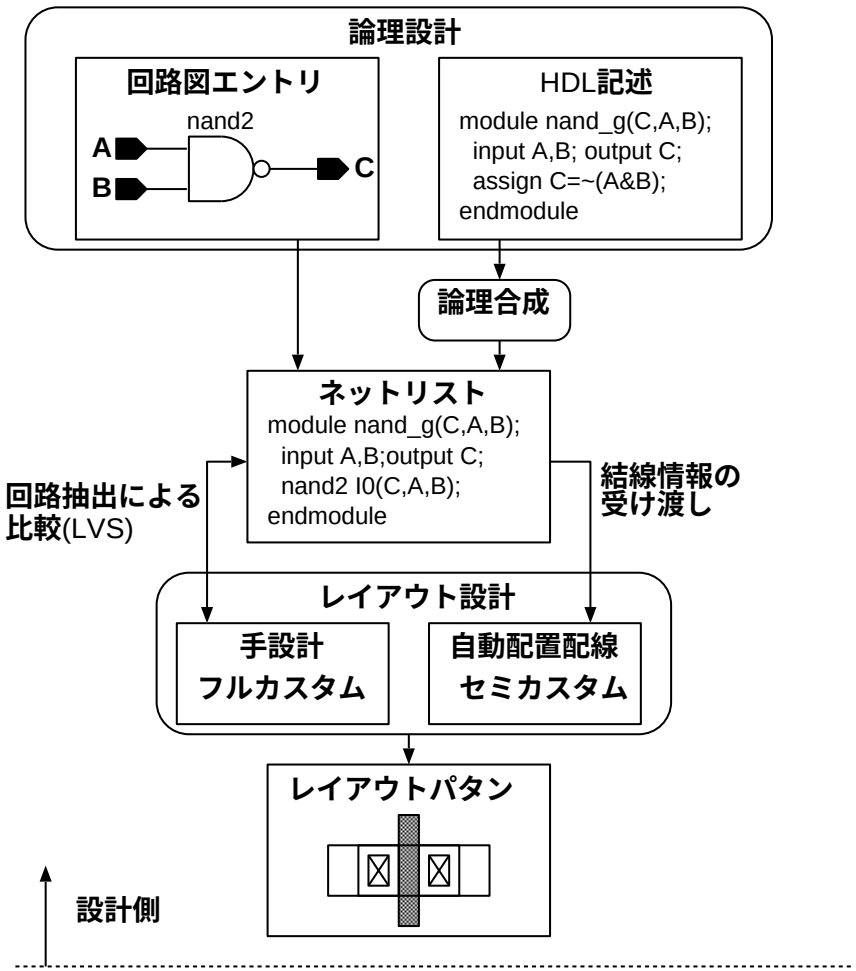
- 経験と勘が必要
  - 豊富な設計経験
- 共有できるものは共有する（リソースシェアリング）
  - ただし共有するとかえって悪くなる場合もある。
- どのような回路が合成される（た）かを考える
  - RTL記述がどのような回路になるか？
  - たとえば、HDLのif記述の多階層化



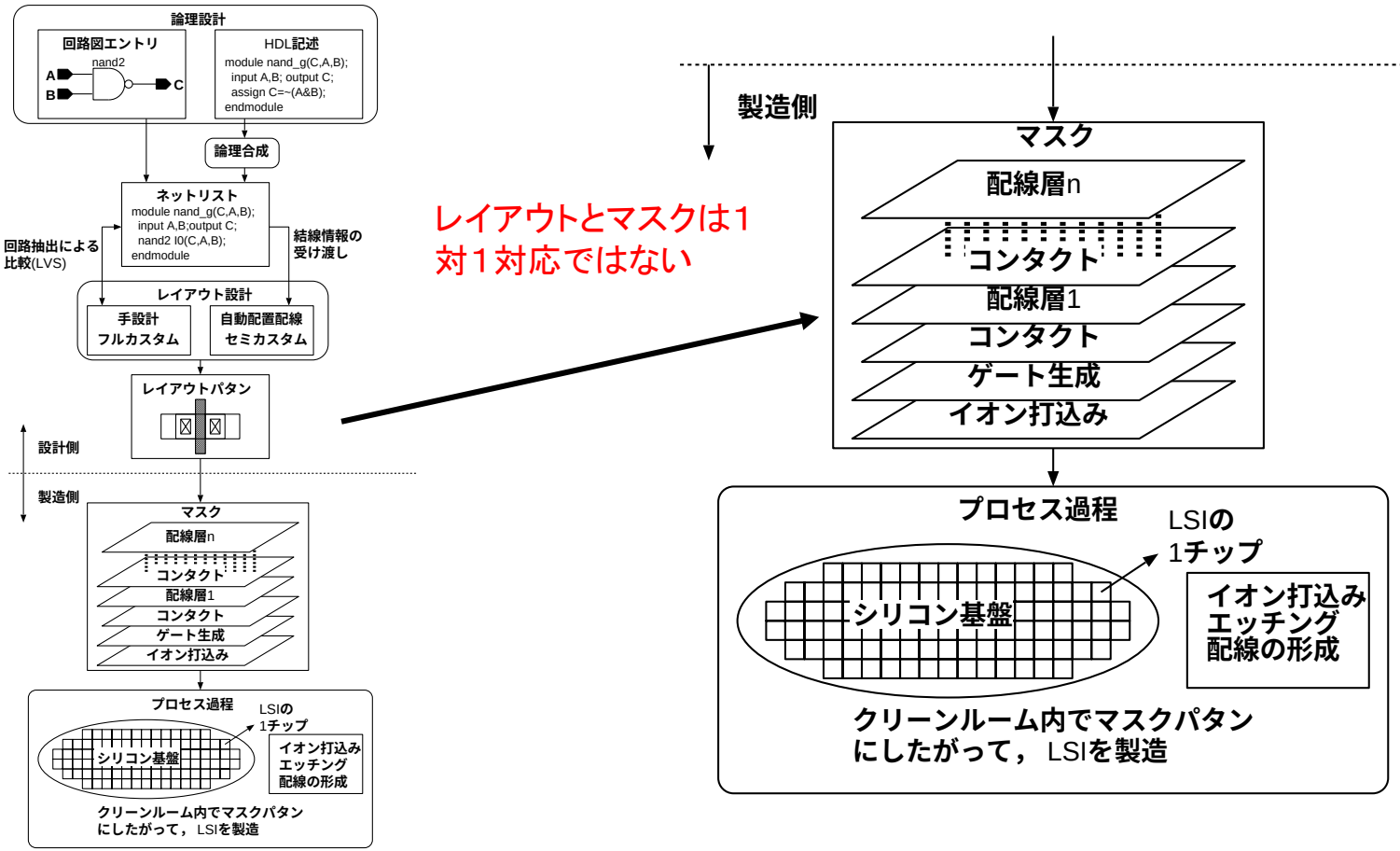
LSIの設計フロー

LSIの設計フロー(設計側)





LSIの設計フロー(製造側)



LSIの設計フロー(Cont.)

- 論理設計: ネットリスト
  - 回路図エントリ
  - HDL記述からの論理合成
- レイアウト設計: レイアウトパタン
  - フルカスタム: 全部手で書く
  - セミカスタム: ある程度自動化
- マスク設計: マスクパタン
  - レイアウトパタンからマスクを作成
- LSIの製造
  - クリーンルームにて
  - 原理は印刷と同じ

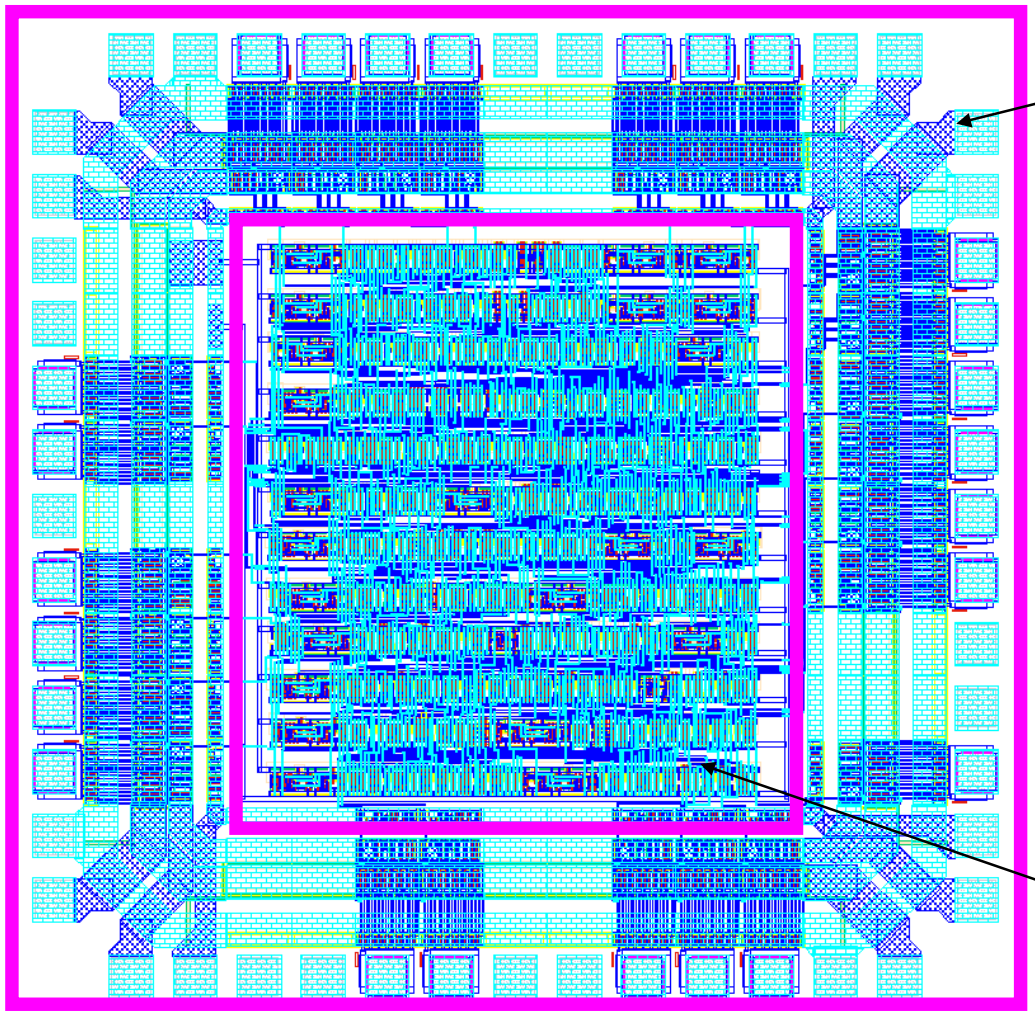
セルベース設計とゲートアレイ（GA）

フルカスタム（レイアウトを手で描く）では、時間と費用がかかりすぎる！！

ただし、性能は圧倒的によくなる

- セルベース設計（スタンダードセル）
  - あらかじめ、基本論理ゲートのパタンを用意し、それを規則的に並べる。
- ゲートアレイ（GA）
  - 基本論理ゲートの下地を作っておき、配線のみ変更
  - 設計期間、製造期間の短縮
  - マスク代が安くなる。

スタンダードセルレイアウト例
チップ全体のレイアウト



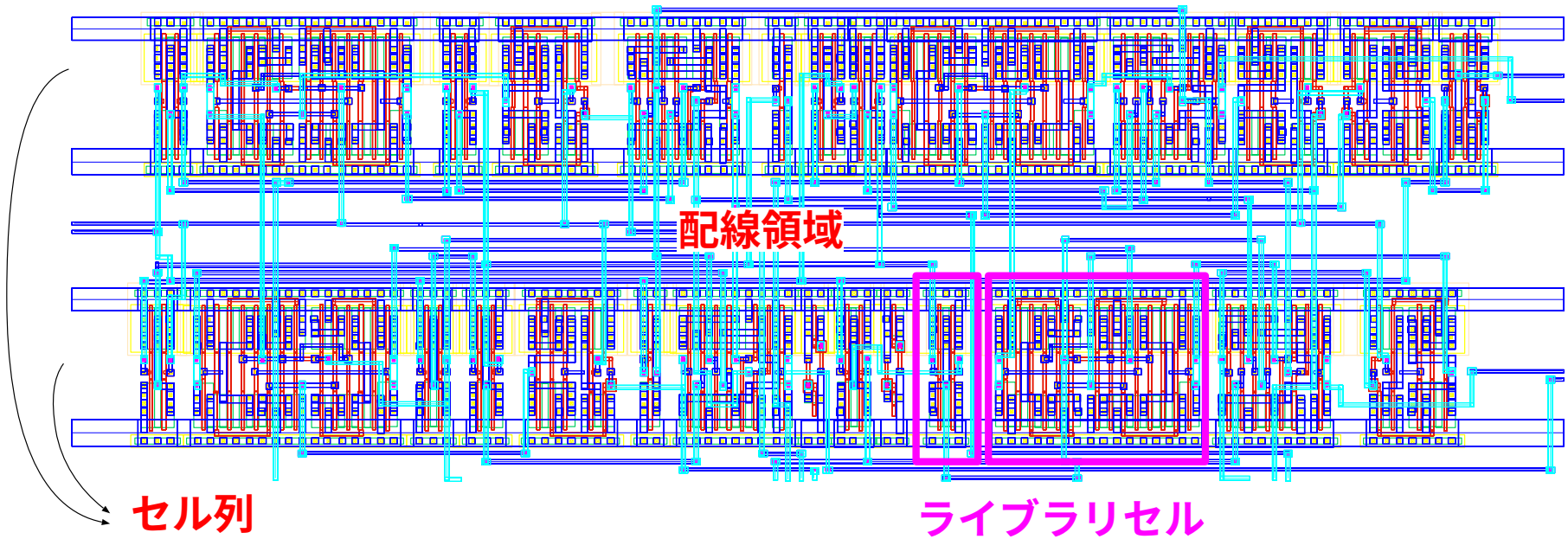
PAD領域

VDEC
CMOS1.2um

2.3mm角

Core領域

スタンダードセルレイアウト例
レイアウトの一部



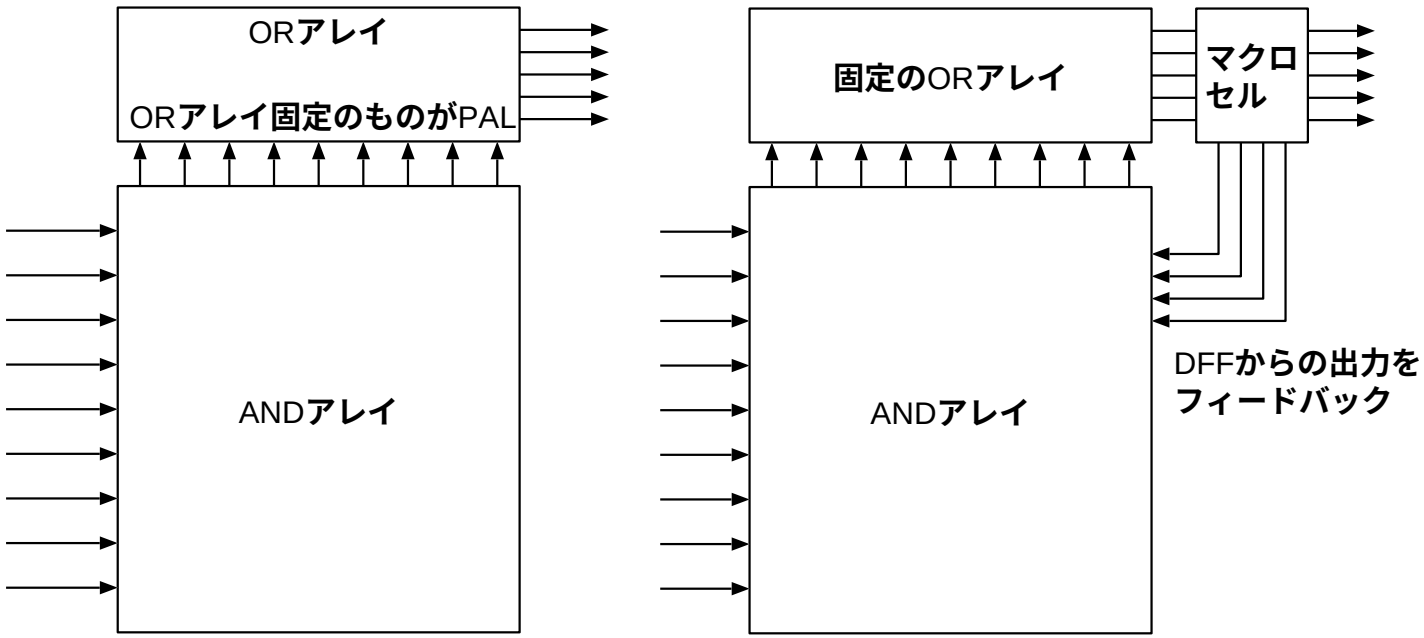
- 論理ゲート、FF等のライブラリセルをアレイ上に並べてから、配線を行う

PLDとFPGA

プログラマブルロジックデバイス（PLD）

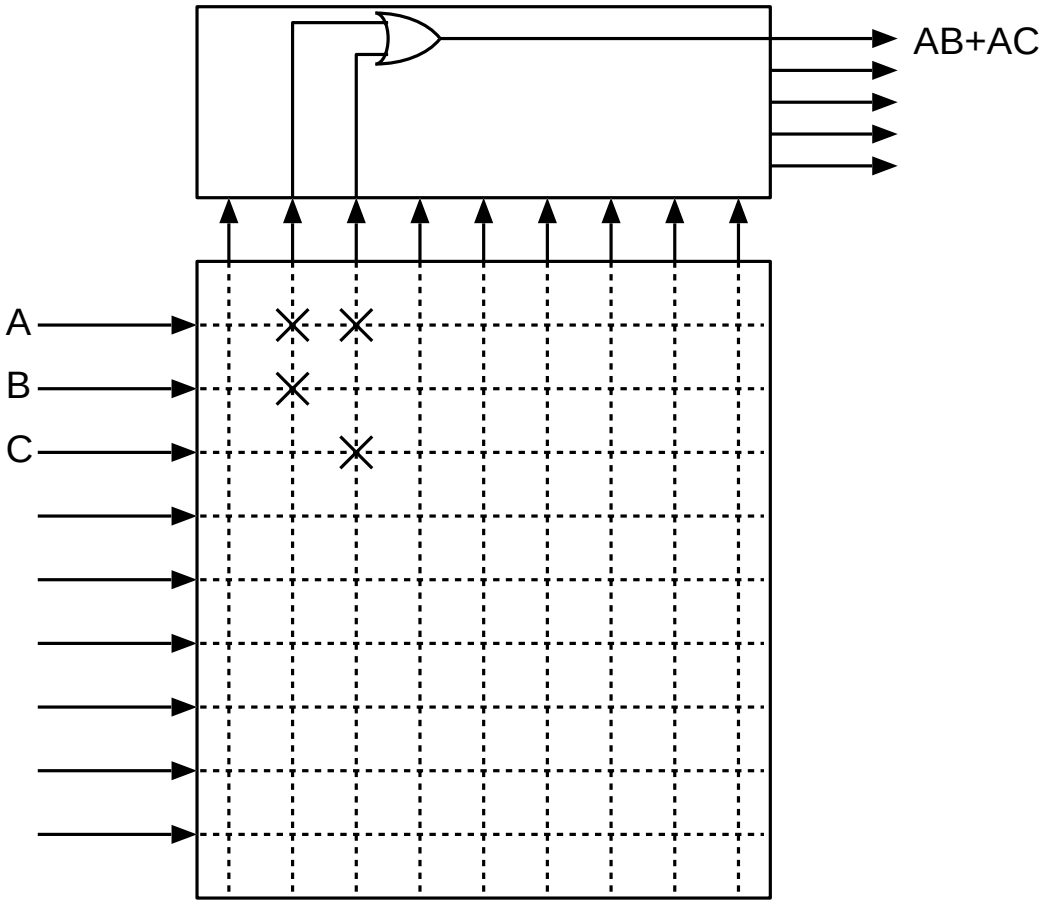
- 設計者が自由にその機能を変更できるLSIの総称（広義）
  - MPD:　マスクプログラマブル→製造時に変更
  - FPD:　フィールドプログラマブル→その場で変更
- 小規模PLDの種類
  - PLA:　Programmable Logic Array
    - » AND-OR アレイ
  - PAL:　Programmable Array Logic
    - » ORアレイが固定
    - » 派生品として、GAL,　PLD（狭義）

PLDの構造



PLA, PAL　　GAL, PLD

PALのプログラム例



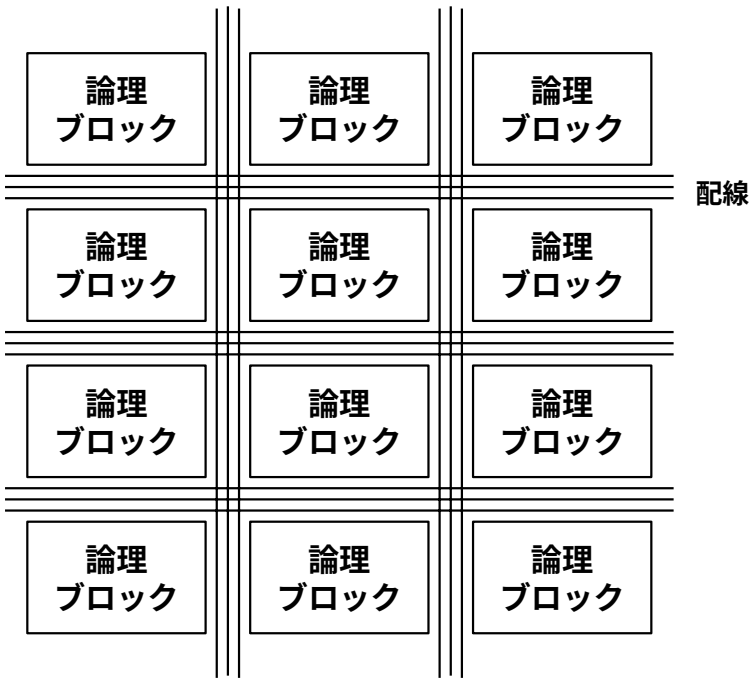
FPGA (Field Programmable Gate Array)

- フィールドプログラマブルな大規模集積回路
  - ようは大規模なPLD
- 論理ゲートとフリップフロップをアレイ上に敷き詰めて、その間の結線を自由に変更
  - ただし論理ゲートそのものが内蔵されているとは限らない
- コンフィグレーションデータを書き込むことにより機能が変化する

MPGA: Mask Programmable Gate Array
一般的にGA

FPGAの構造

- 組み替え可能な論理ブロック
- 論理ブロック間を接続する組み替え可能な配線

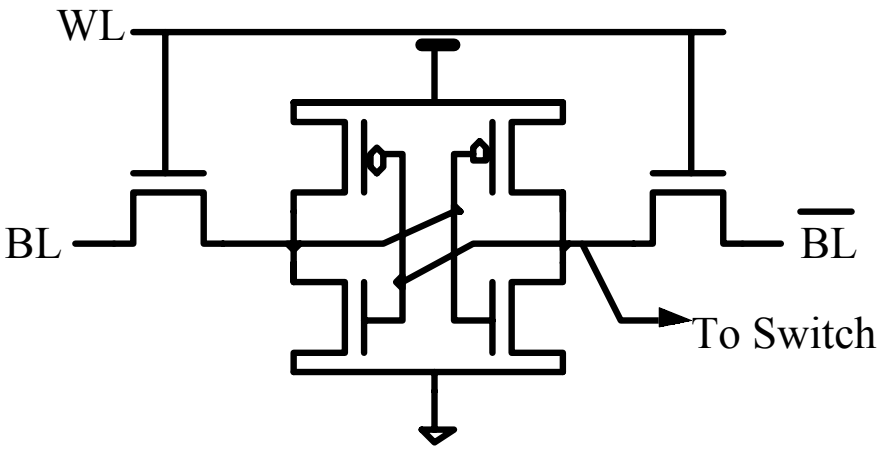


FPGAのプログラム記憶方式

FPGAの現在の構成(コンフィグレーション)を覚えておく方法

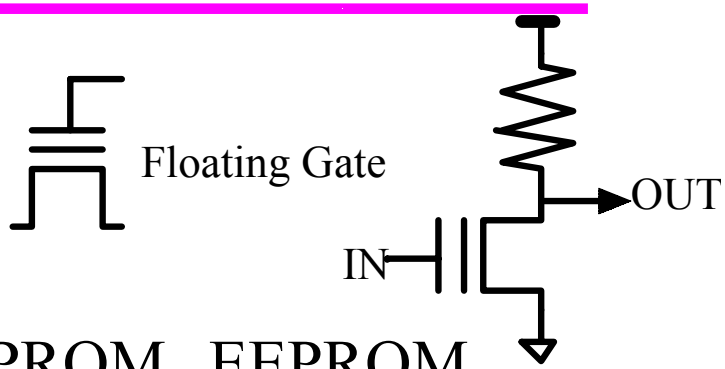
- SRAM等の揮発性メモリに書き込む.
  - もっともポピュラー
  - 特別なプロセスを必要としない
- EPROM, EEPROM等の不揮発性メモリに書き込む.
- 電圧をかけて, アンチヒューズを短絡させる.

プログラム方式



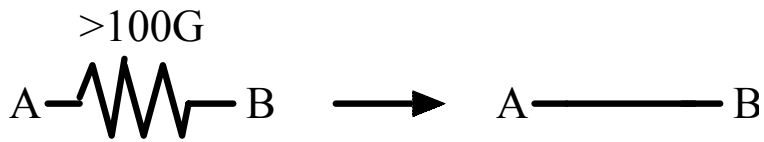
SRAM

- ロジックと同じプロセスで製造できる。
- 冗長度が大きい。



EPROM, EEPROM

- 特殊なプロセスを要求
- 冗長度は小さい。



Anti-fuse

Apply 16V between A and B

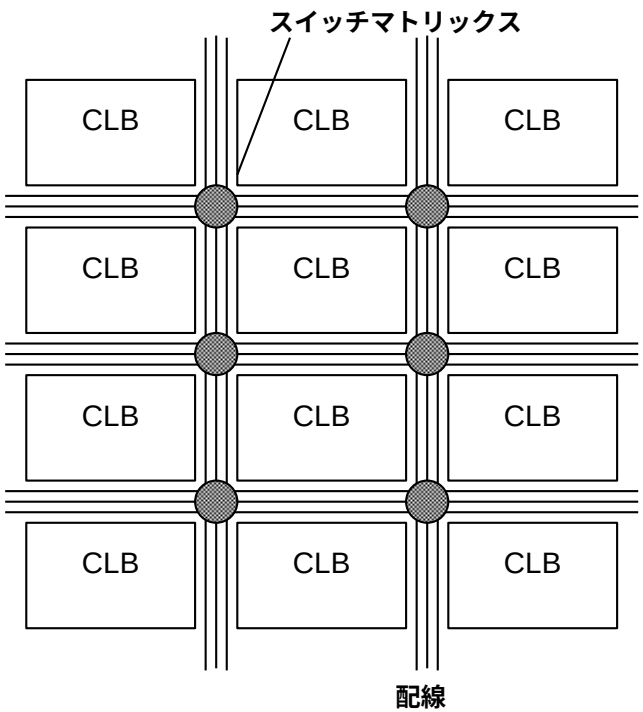
- 小さくて高速
- 書き込みは一度だけ

FPGAの特性分類

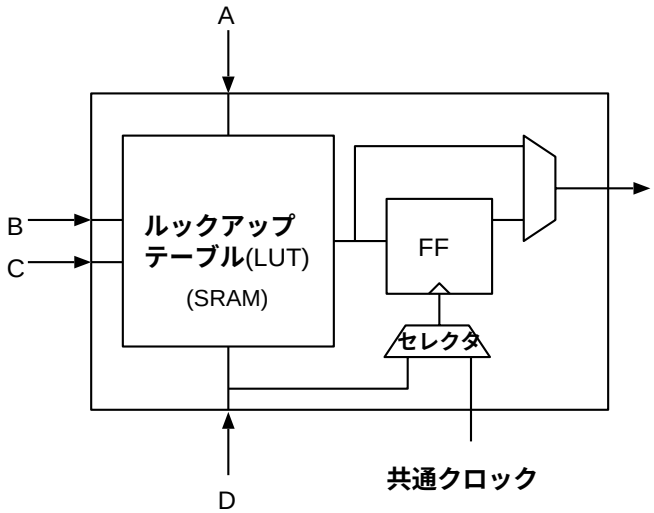
プログラム方式	再書込	不揮発性	動作速度	冗長度
SRAM	○	×	遅い	大
EPROM	△	○	中	中
EEPROM	○	○	中	中
アンチヒューズ	×	○	速い	小

書き込み回数に制限のあるものが多い

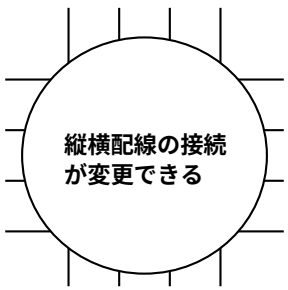
XILINX XCシリーズの構造



CLB: Configurable Logic Block



CLBの構造



スイッチマトリックス

LUT(Look-up Table)

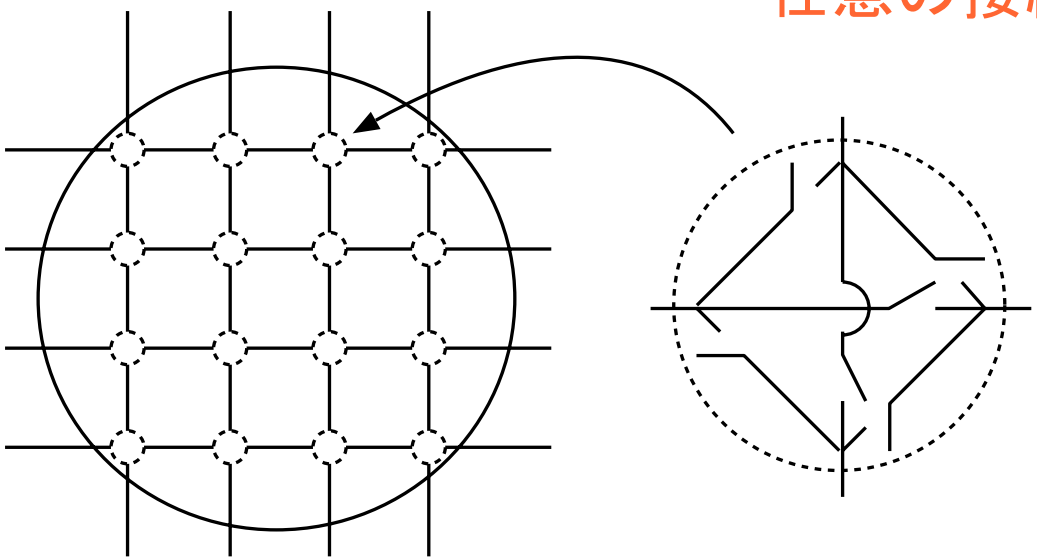
- SRAM型FPGAの可変論理を実現する。
- A, B, C, Dの4ビット入力をワード線とした1ビットのSRAM
- SRAMの中身を書き換えることで任意の論理を実現
- (A|B)&(C|D)に対するLUT→

A	B	C	D	設定値
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

スイッチマトリックス

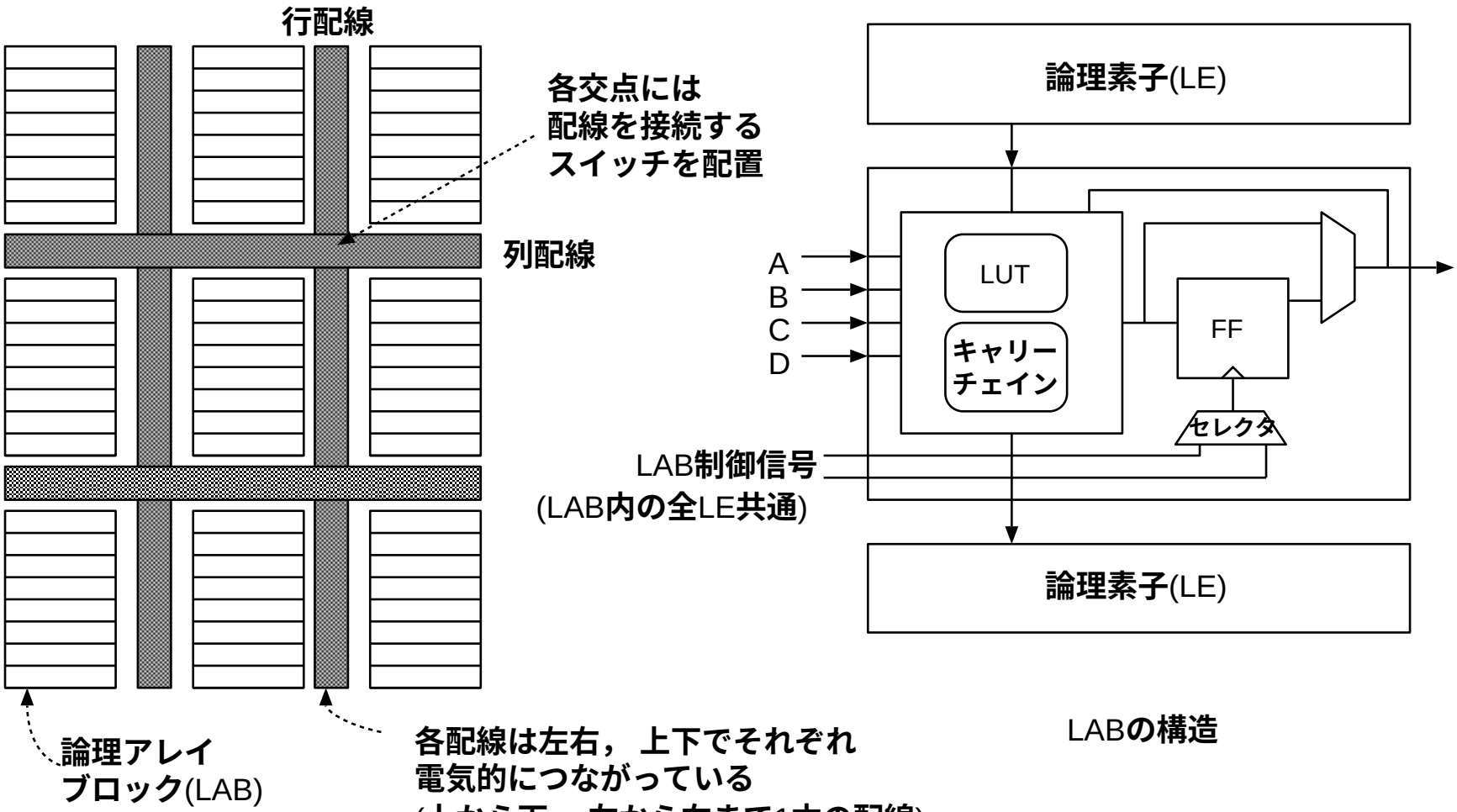
- SRAM型FPGAの可変配線を実現

任意の接続が可能



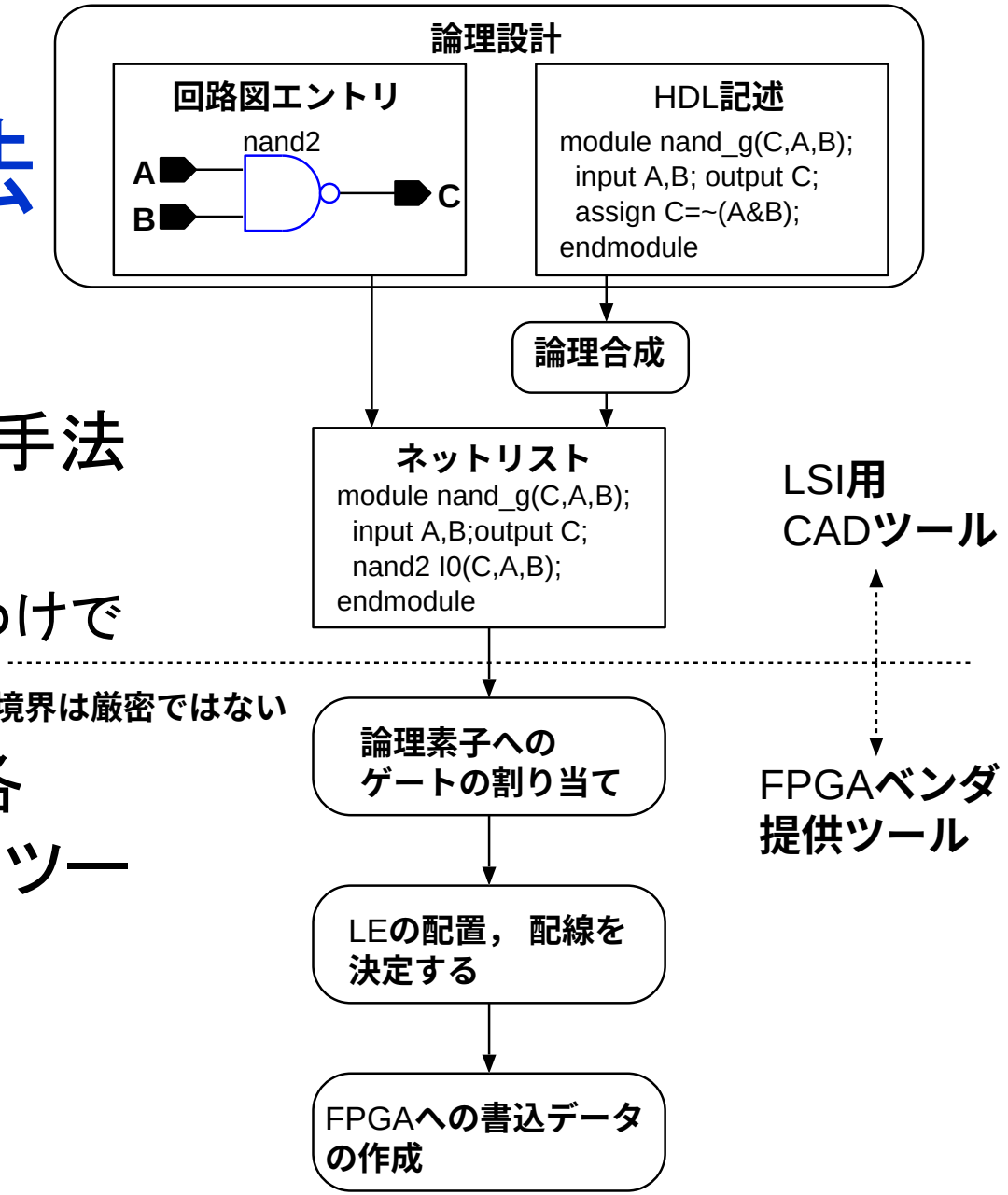
スイッチマトリックス　　各交点の構造

ALTERA FLEXの構造



FPGAの設計法

- 通常のLSIと同じ設計手法を取る。
  - LUTを直接設計するわけではない
- FPGAの配置配線は各FPGAベンダー配布のツールにより行う



CAD、FPGAベンダのツール

ベンダ名	ツール名	用途
CADベンダ		
Synopsys	FPGA Compiler II	論理合成
Exemplar	spectrum	
Synplicity	Synplify	
Synopsys(Viewlogic)	WorkView Office	回路図エントリ
FPGAベンダ		
ALTERA	MAX+plus II，Quartus	回路図エントリ，論理合成，シミュレーション，タイミング解析
XILINX	Alliance等	
Actel	DeskTop	

- FPGA Compiler IIはVDECのライセンスで利用可能
- 各社FPGAの無償ツールあり。大学向けのプログラムもあり
- See http://www.ベンダ名.com/

まとめ

- 記述レベルがどんどん抽象化していっても、RTLはしばらく生き残る。
- 回路の動作から最適なRTLを導き出す能力を養うことが肝要
- 組み合わせ回路の最適化は、計算機に任せればよいが、記述からどのような回路が合成されているかは見ておく

参考資料

- ディジタル集積回路の設計と試作
  - VDEC監修 浅田邦博 編 (培風館)
  - ISBN4-563-03547-5 / 定価 3000円
  - 著者: 越智 裕之(広島市立大学)、池田 誠(東京大学)、小林 和淑(京都大学)
- 本資料
  - http://www-lab13.kuee.kyoto-u.ac.jp/~kobayasi/refresh にてPDFで公開予定
- 琵琶湖WSポスターセッションの投稿を！！
  - 賞金10万円、締切　8月31日