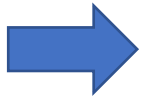
論理回路と TFHE
(L-III 第3回ミーティング)

2021/08/03

L-III 講師 松本直樹

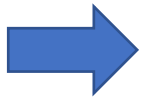
BFV,BGV vs TFHE (第2回MTGより)

- 計算内容を工夫したり暗号学的なトリックで様々な計算を実現してきた
  - if (s == 0) then
    x * 2 + y
    else if (s == 1) then
    x * 4



(x * 2 + y)*s + (x * 4)*(s-1)

- ReLU 関数など実現できそうにない計算も存在
  - if (x > 0) then
    x
    else
    0



?

BFV,BGV vs TFHE (第2回MTGより)

- TFHE は論理ゲートと等価な処理をFHE上で実現できる
- 整数 → 複数個の bit の集まり
  - 正負のある整数(符号付き整数) は2の補数で表現可能
  - 8bit 符号付き整数(int8_t)とか
    - 00000101 → 5
    - 11111001 → -7
- ReLU 関数も容易に表現可能
  - 符号bit を見る
    - 0 ならそのまま出力
    - 1 なら全部 0
      → 符号bit を反転して AND を取ればOK

Logical circuit over TFHE(第2回MTGより)

- Chisel ならReLU関数も簡単に記述可能!
- **なんなら暗号を知らなくても実装できる**

```
class ReLU8bit extends Module {
    val io = IO(new Bundle{
        val in = Input(UInt(8.W))
        val out = Output(UInt(8.W))
    })

    when(!io.in(7)){
        io.out := io.in
    }.otherwise{
        io.out := 0.U(8.W)
    }
}
```

入力(整数8bit)

出力(整数8bit)

符号ビットが0(=正の数)なら
そのまま出力

符号ビットが1(=負の数)なら0を出力

論理回路の基本

- 論理回路とは？
  →ディジタルな電子回路による、**論理演算**や記憶を行う回路
  (by Wikipedia)
- 論理演算とは？
  A, B をバイナリ(0か1)として
  - NOT A (否定)
  - A AND B (論理積)
  - A OR B　(論理和)
  - A XOR B (排他的論理和)
  - A NOR B (否定論理和)
  - などなど

論理回路の基本

- 入力と出力の対応関係(**真理値表**)
- A AND B

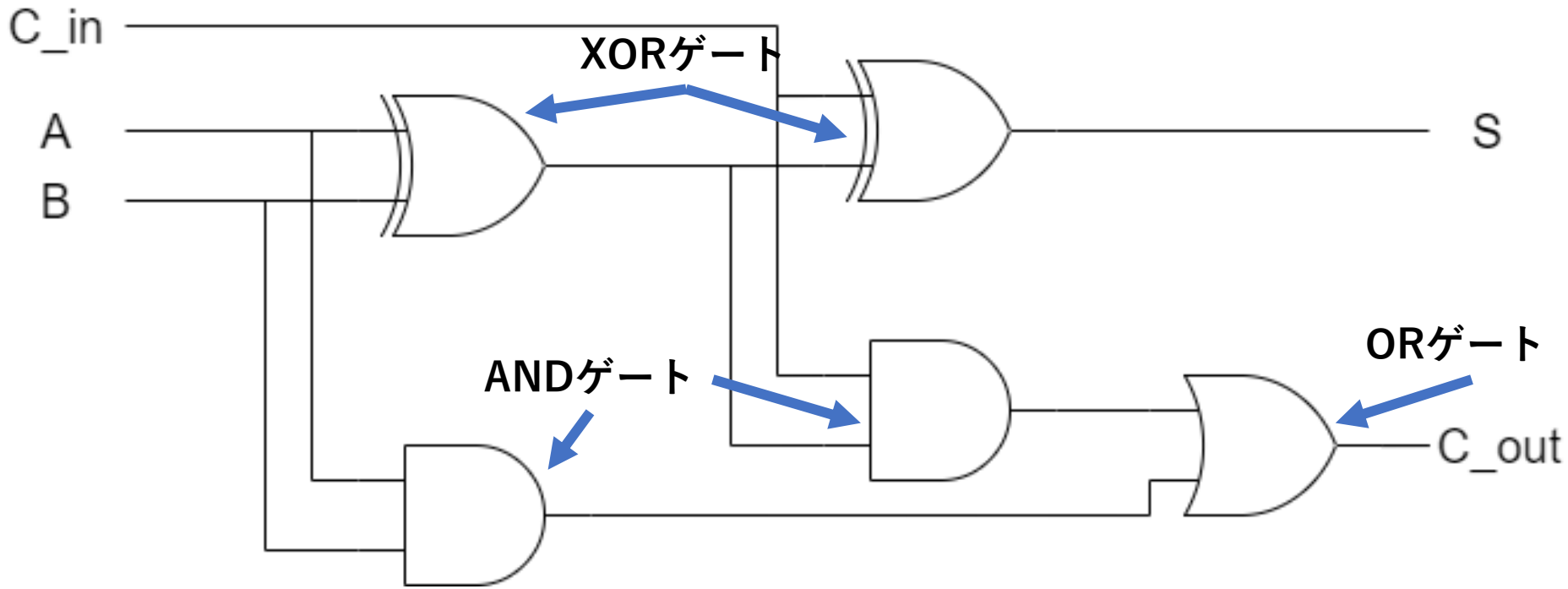
入力A	入力B	出力
0	0	0
0	1	0
1	0	0
1	1	1

- A OR B

入力A	入力B	出力
0	0	0
0	1	1
1	0	1
1	1	1

論理回路の基本

- 現代の計算機のほとんどが論理ゲート同士を接続して任意の演算を実現(論理ゲートがCPUを構成する最小要素)
- Ex) 全加算器(加減算演算回路の基本要素)

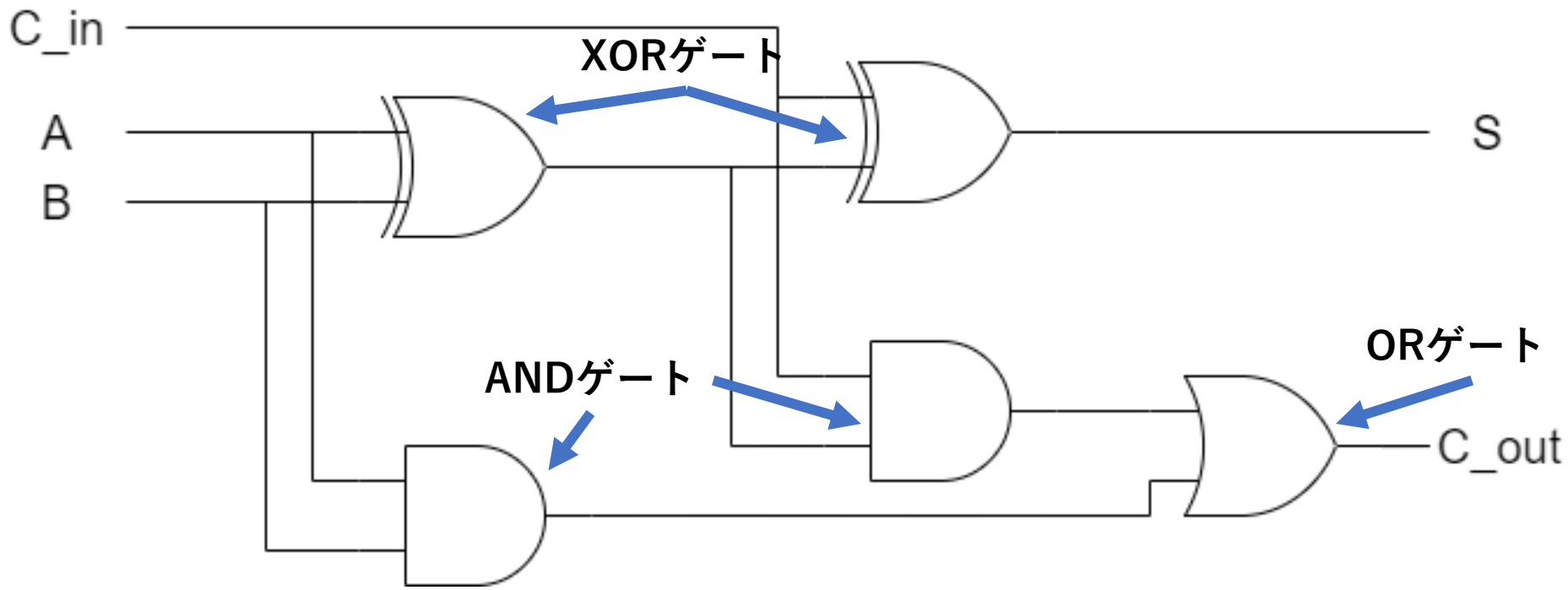


論理回路の基本

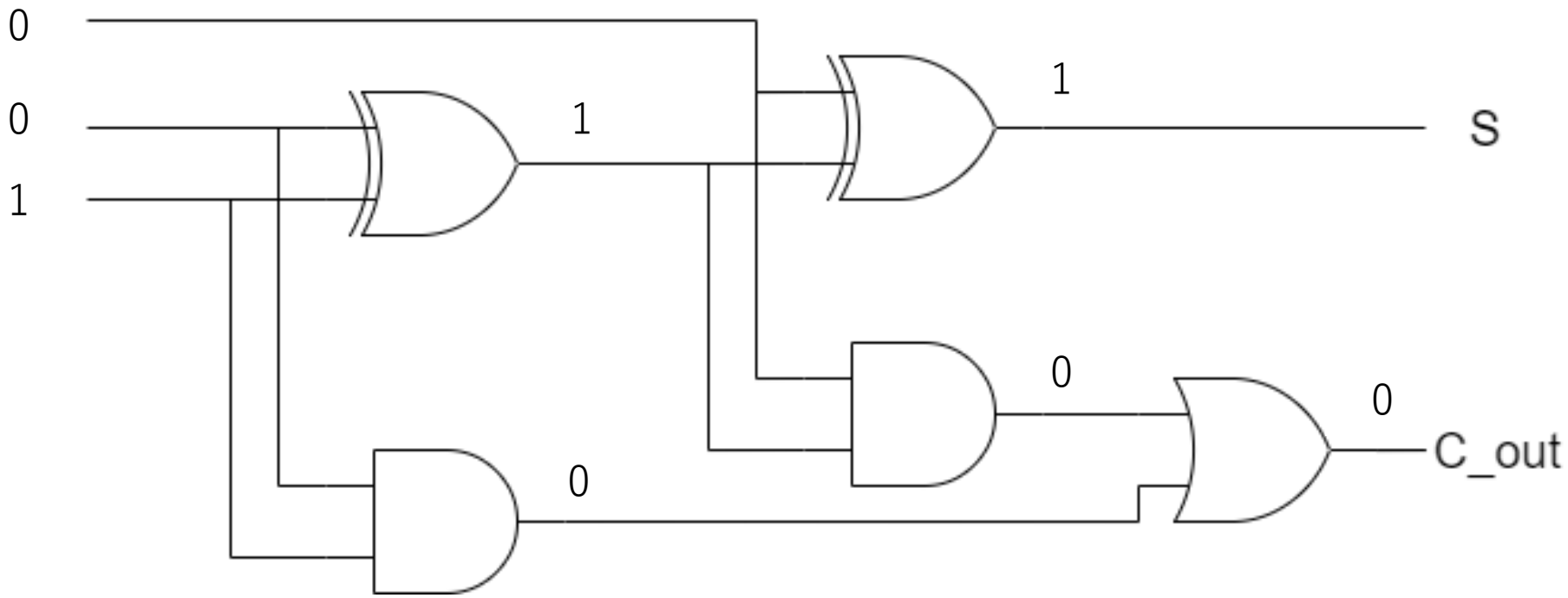
- **組み合わせ回路** と **順序回路**
- 組み合わせ回路
  → 論理ゲートから構成される回路
  　入力が決まると出力が決まる
- 順序回路
  → 組み合わせ回路と**同期記憶回路**から構成される回路
  　クロックによって回路の動作が同期される

論理回路の基本

- 組み合わせ回路
- 簡単に言えば、論理ゲートだけの回路

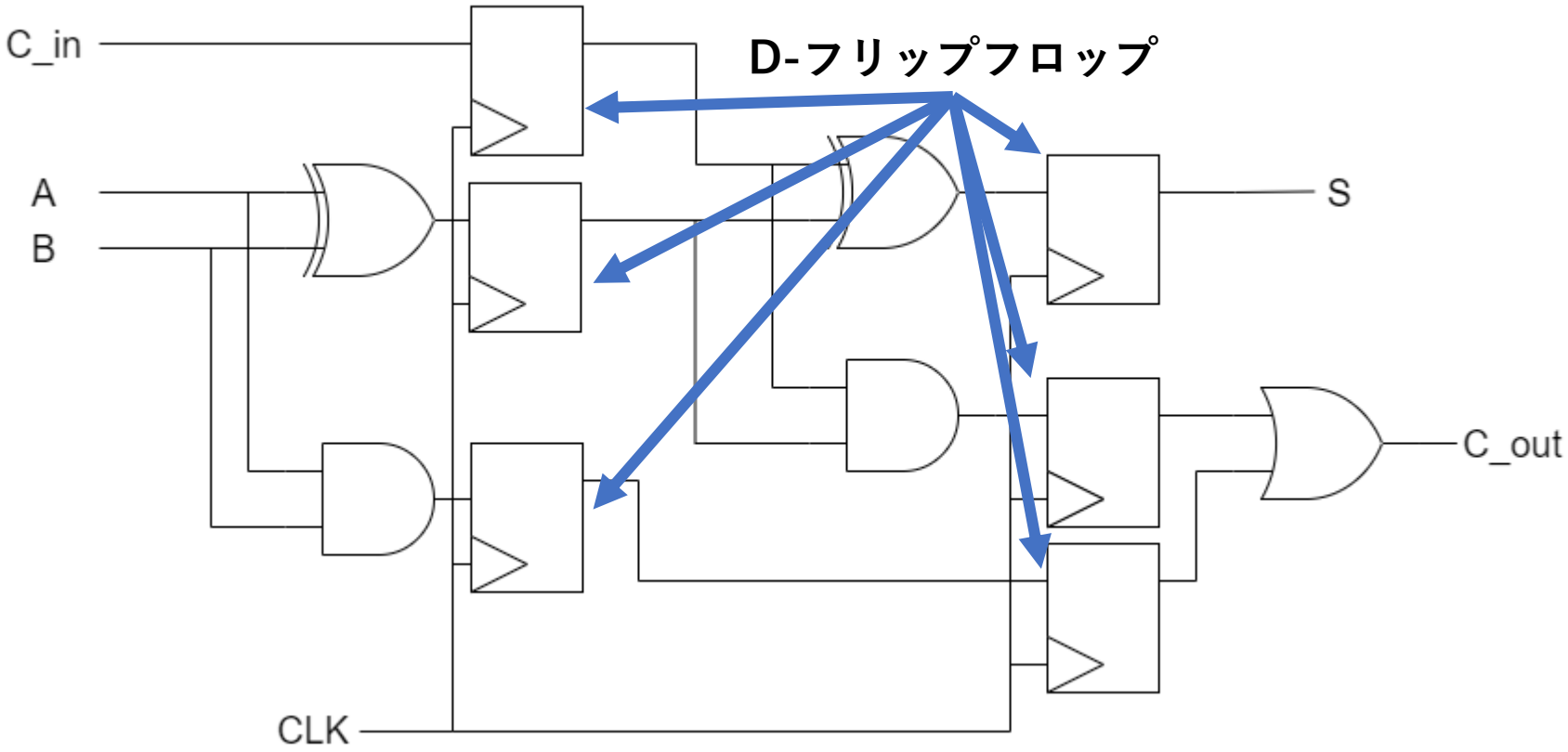


論理回路の基本



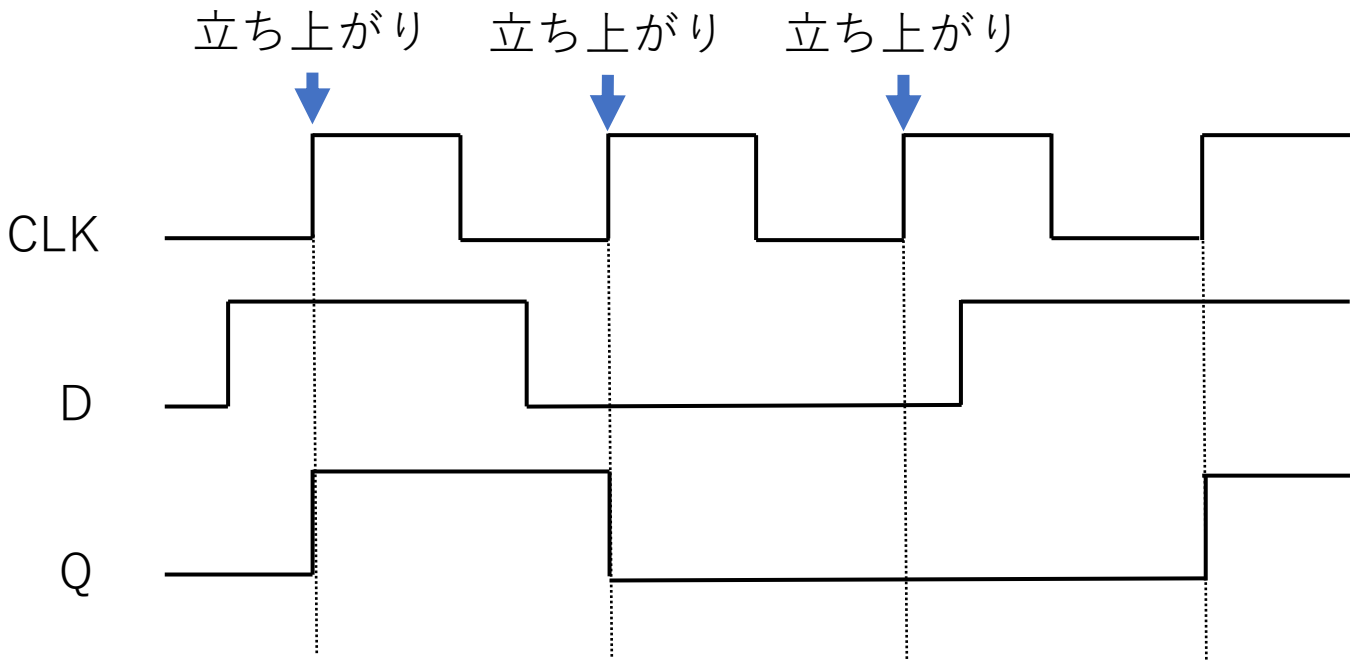
論理回路の基本

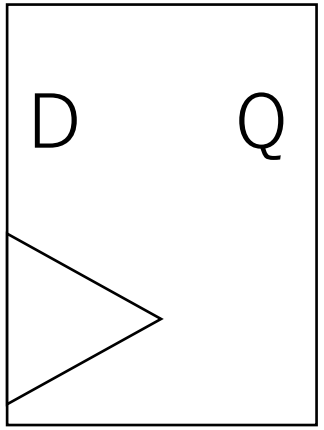
- 順序回路
- **フリップフロップ(D-FF 等)**によってクロックに同期



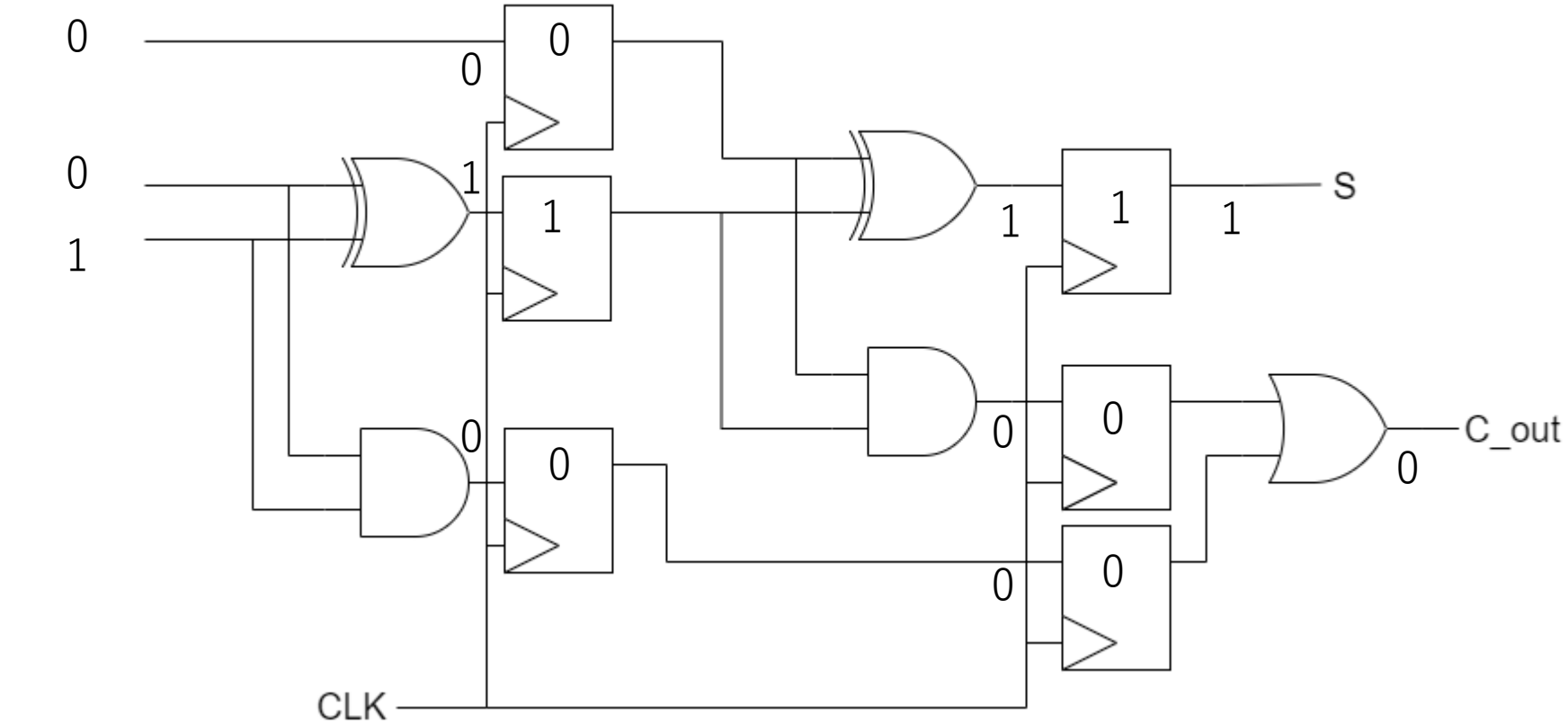
論理回路の基本

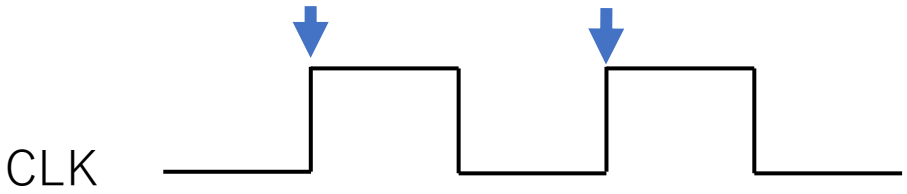
- D-フリップフロップ(D-FF)について
- クロックの立ち上がり(立ち下がり)時にDの値を保存しQに出力する





論理回路の基本

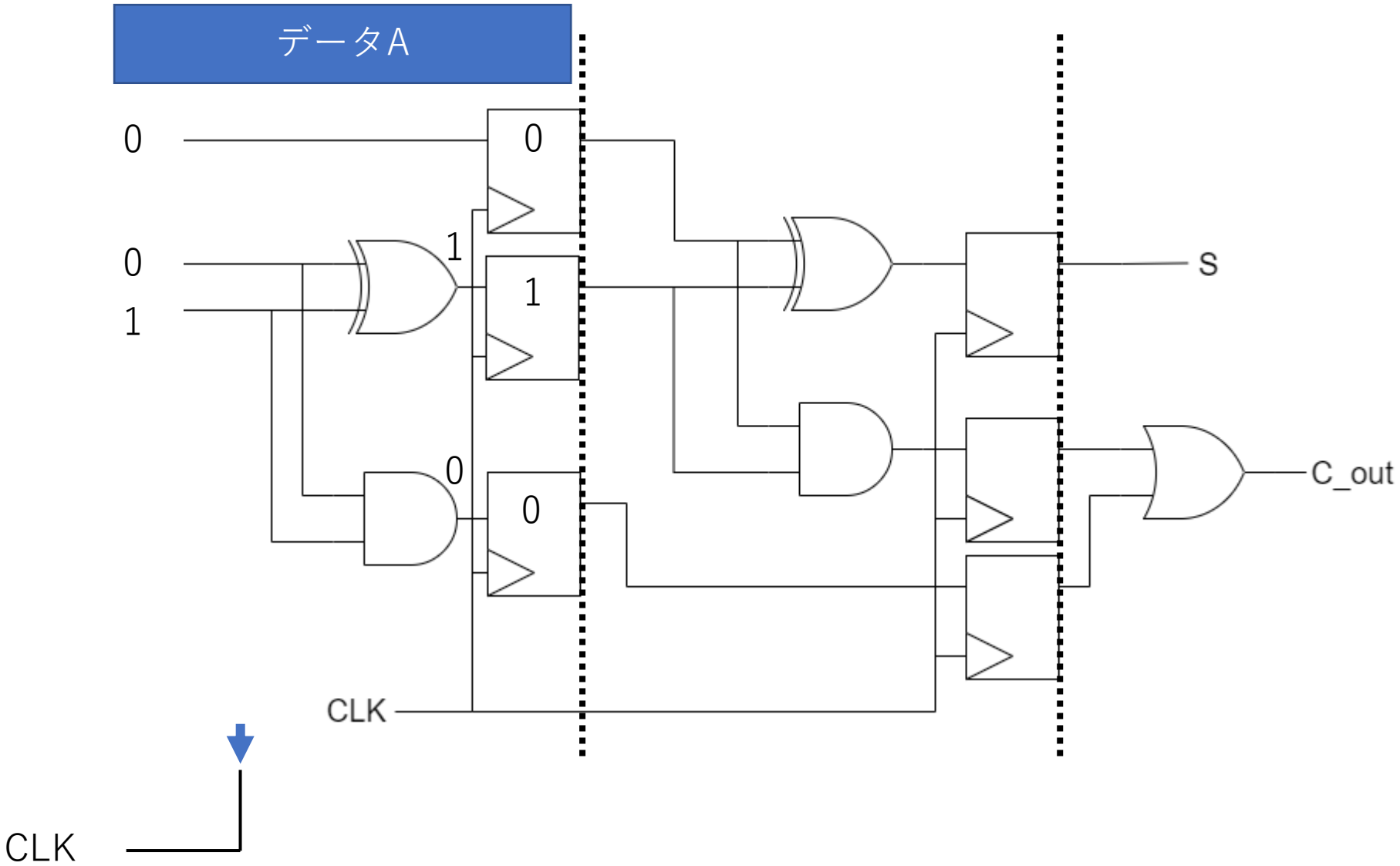




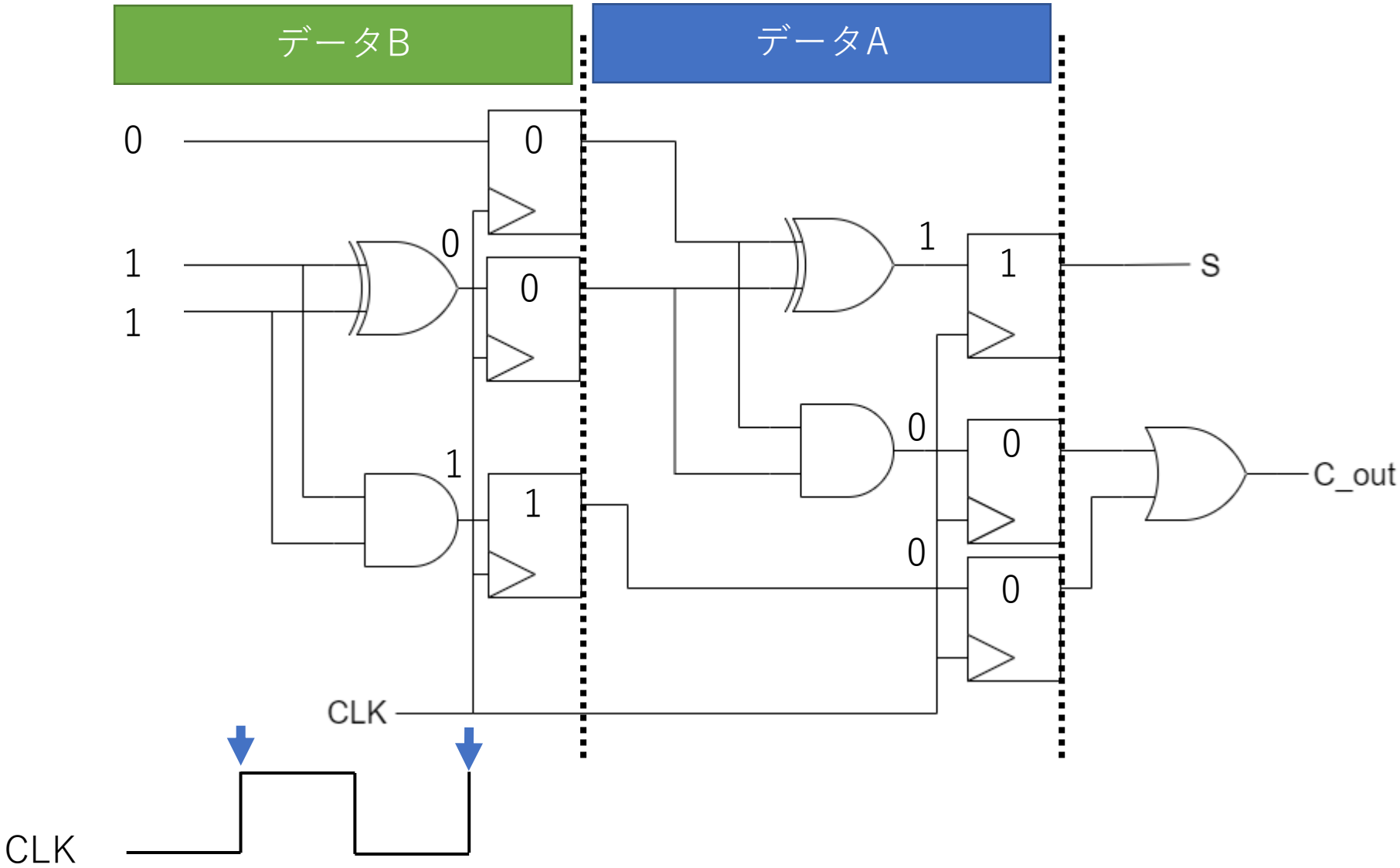
論理回路の基本

- パイプライン化
  → 組み合わせ回路を D-FF で分割し回路遅延を減らす
  　データをクロック毎に入力することでスループットが向上
- 図の全加算器は
  - 組み合わせ回路の 3 分の 1 の遅延
  - 平均1データ/ 1 サイクルで処理

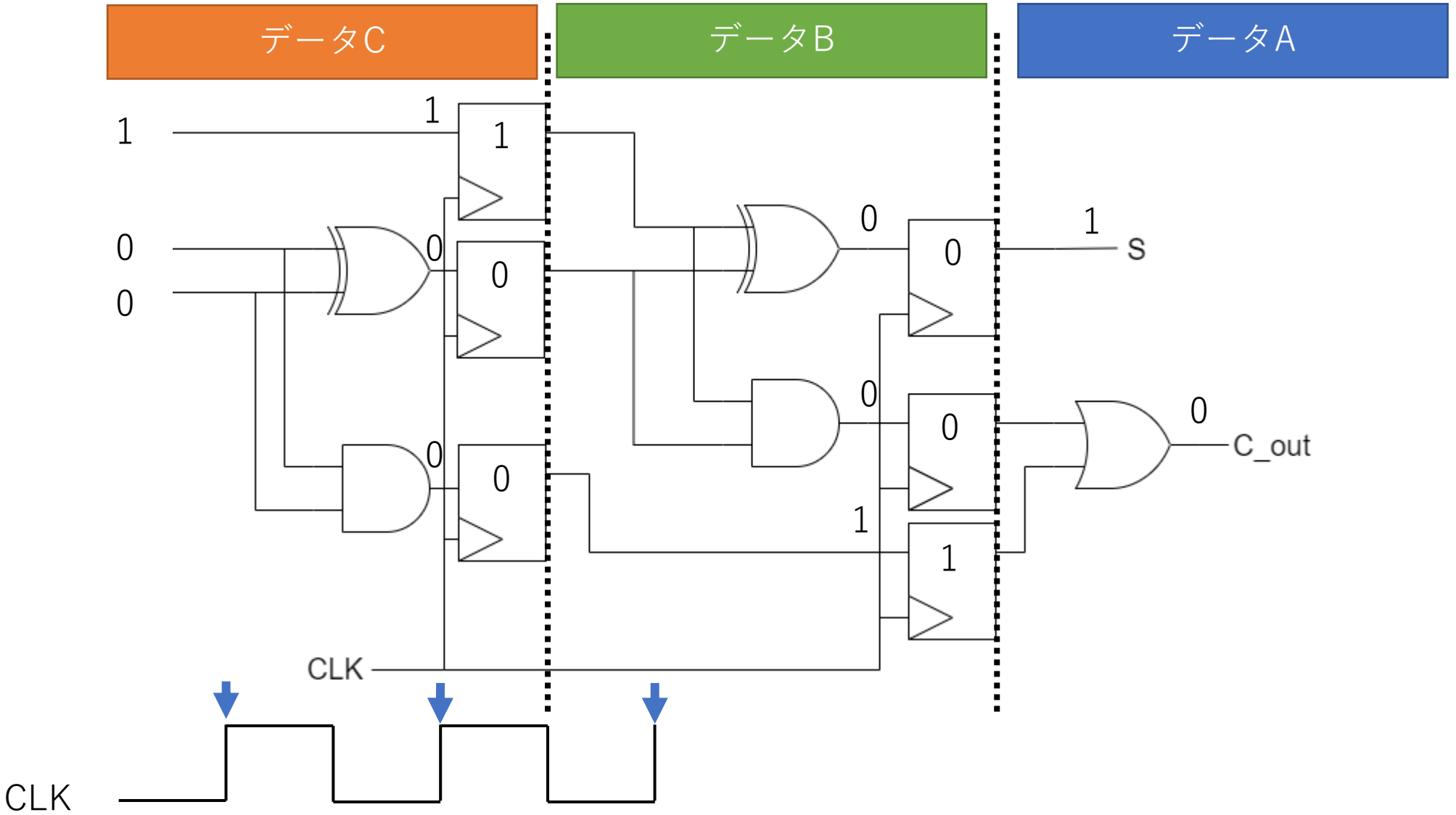
論理回路の基本



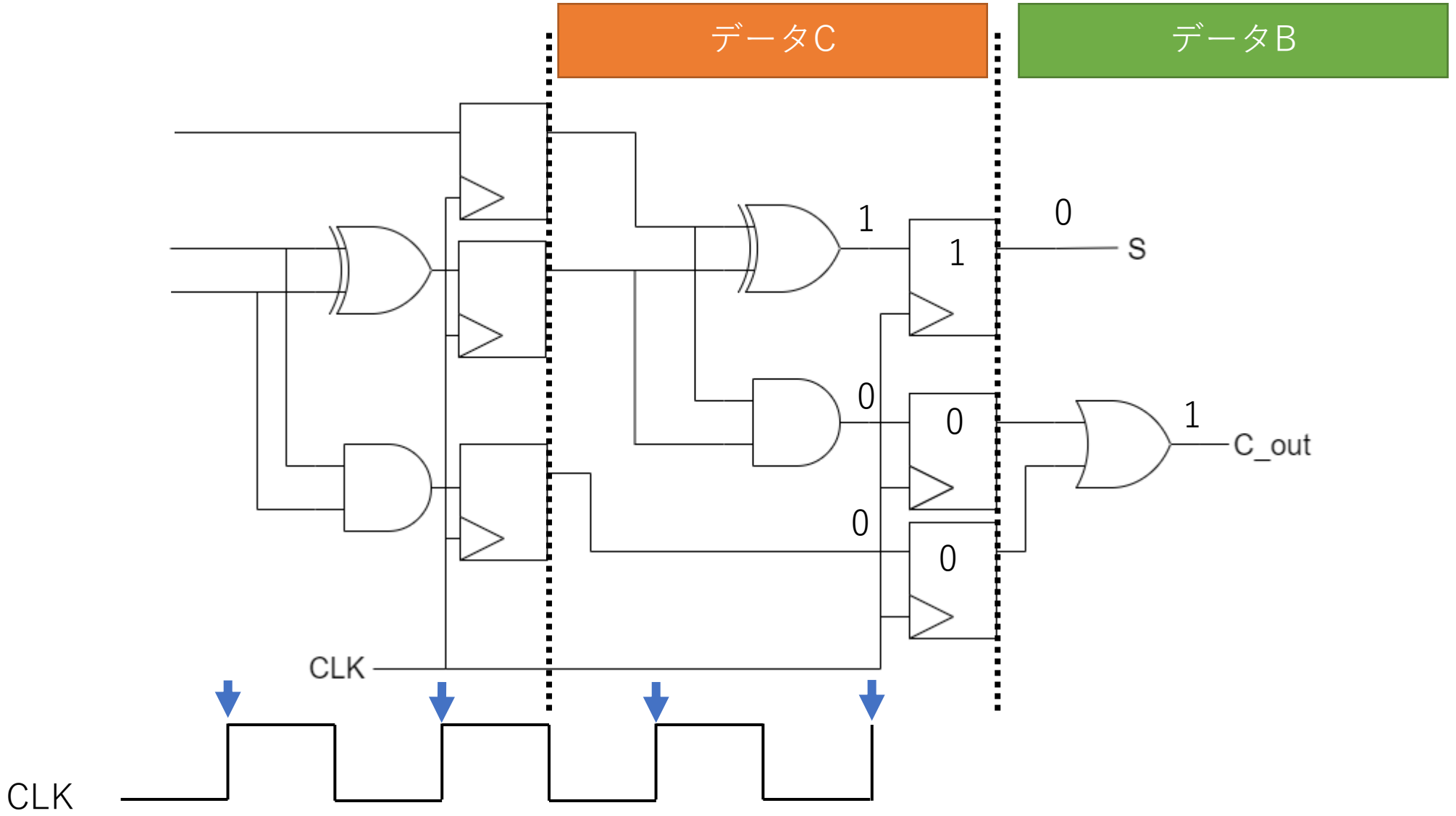
論理回路の基本



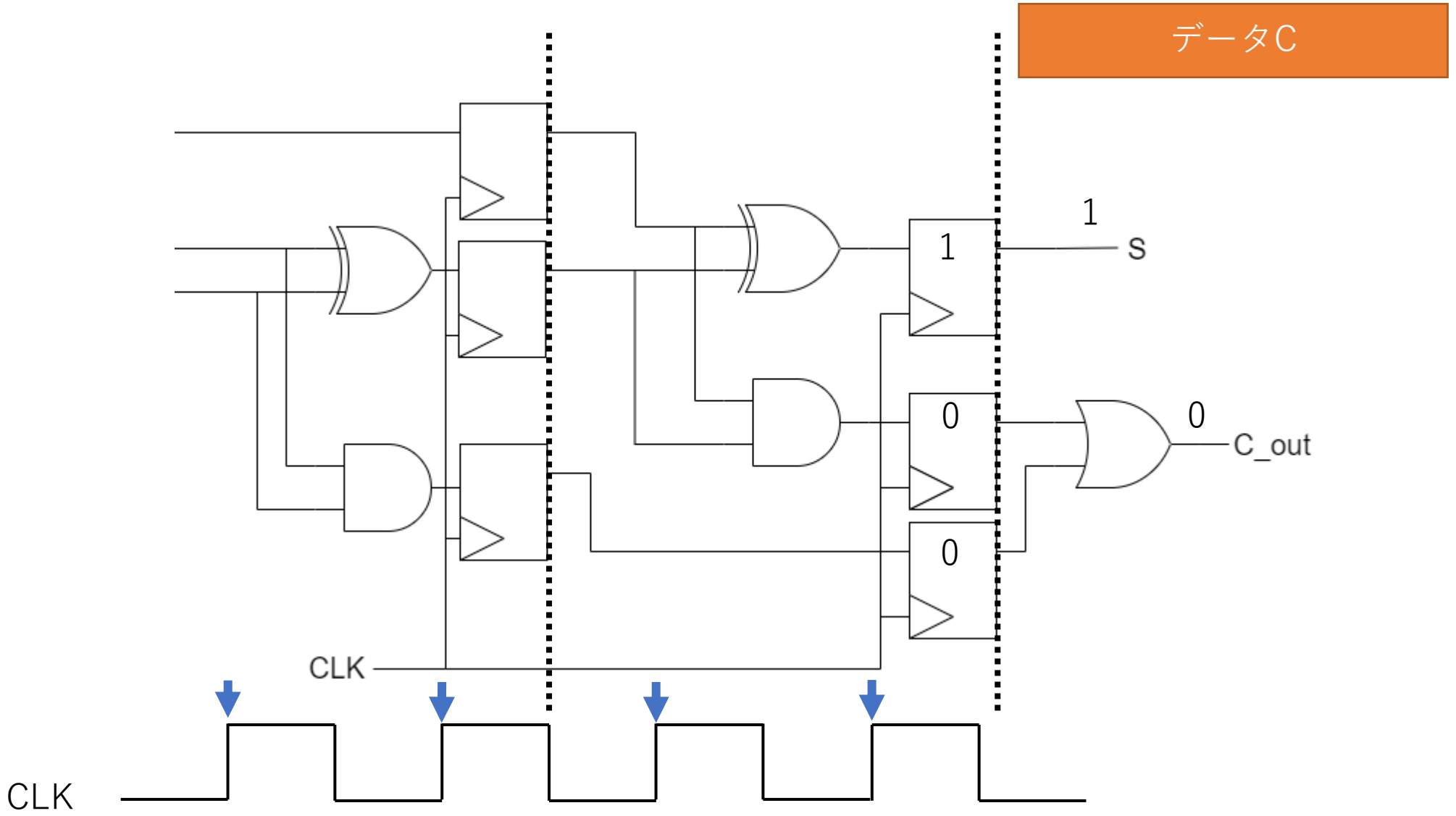
論理回路の基本



論理回路の基本

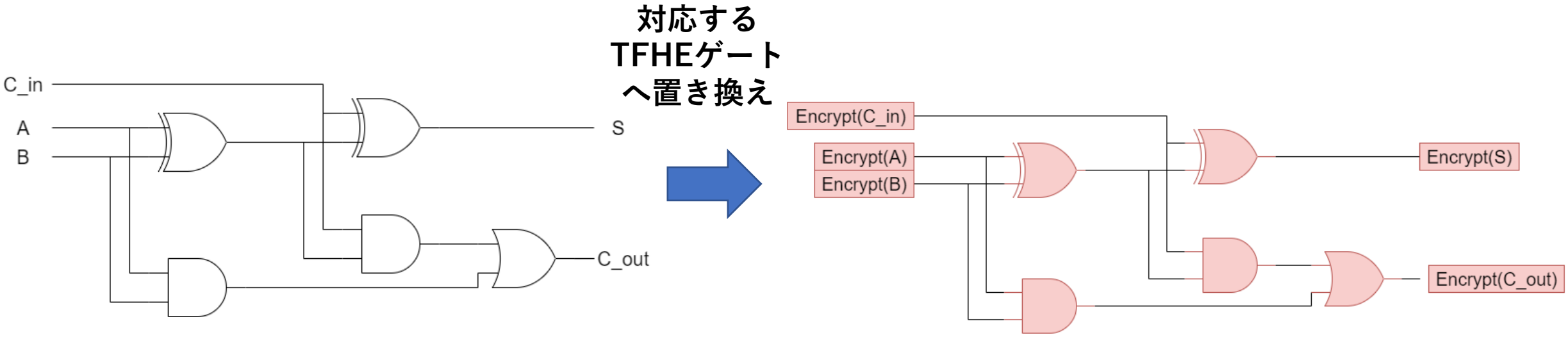


論理回路の基本



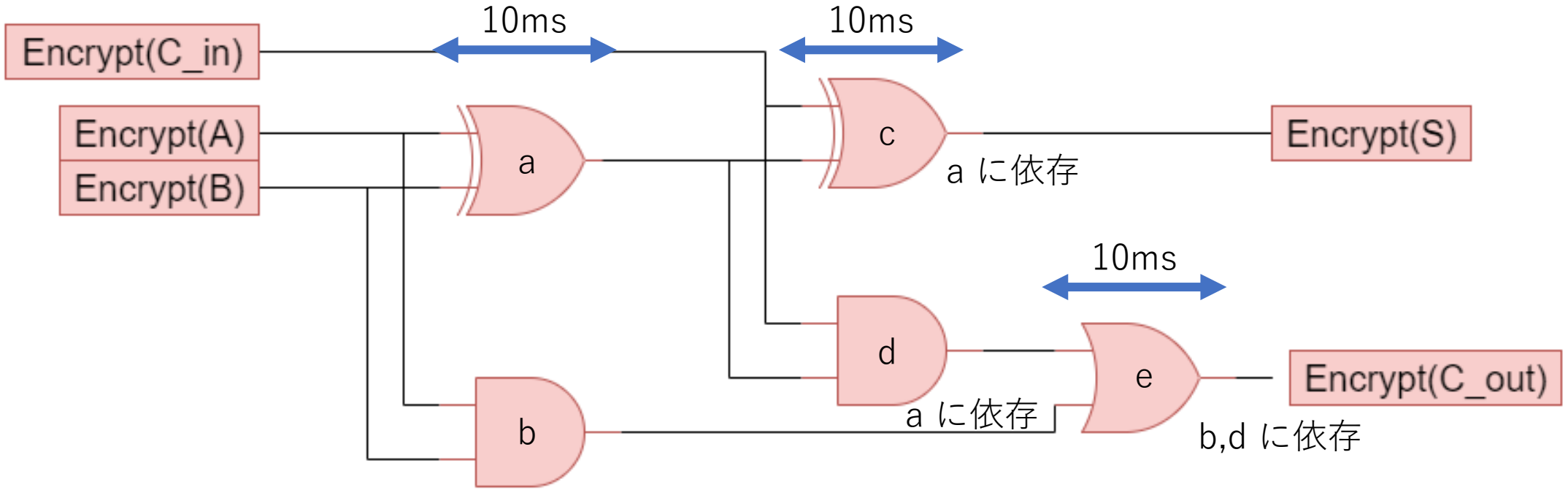
論理回路とTFHE

- 論理ゲートをTFHEゲートに置き換えることで完全準同型暗号による処理に対応できる→**非常に容易**



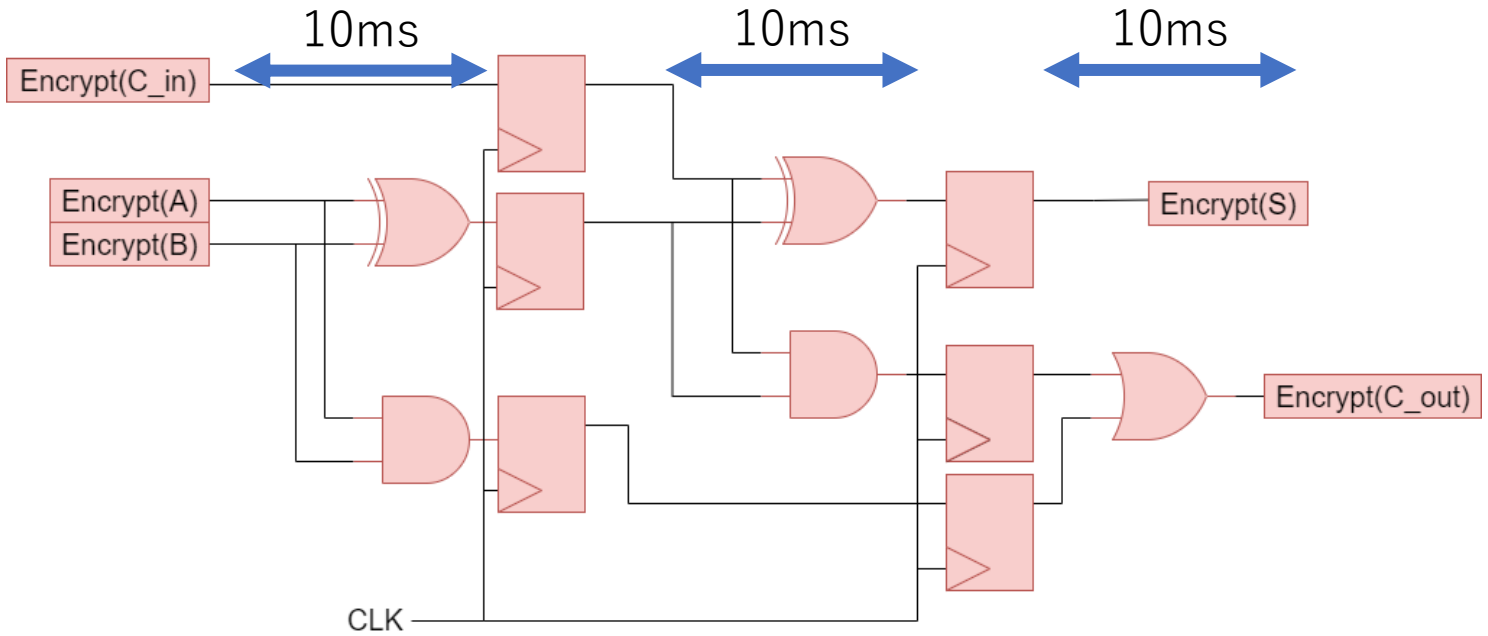
論理回路とTFHE

- TFHEゲートひとつあたりの処理時間は10ms前後
  → 組み合わせ回路が深くなると処理時間が線形に増加
  上流ゲートの結果に依存するため並列実行も不可能
- 下図は処理に最小でも30msかかる



論理回路とTFHE

- TFHE における D-FF は単なる変数
- クロックの立ち上がり → 変数への結果の保存を一斉に行う
- **パイプライン化**により、並列実行を効率的に行える
  → 最小処理時間は10ms x 3 = 30ms(次々にデータを流せば平均10ms)



論理回路とTFHE

- 回路の依存関係を解決して並列に実行する必要がある
  → lyokan がその役割を担う
- 回路をDAG(Directed Acyclic Graph)で表現し依存関係を解決

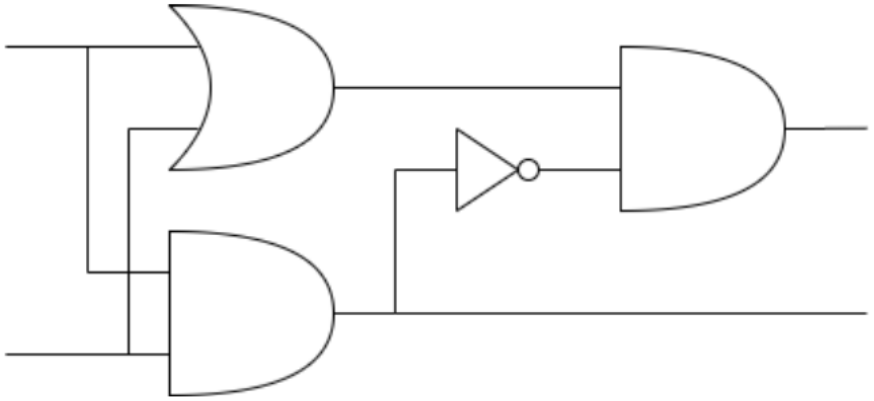


図 18: ネットリスト形式の回路

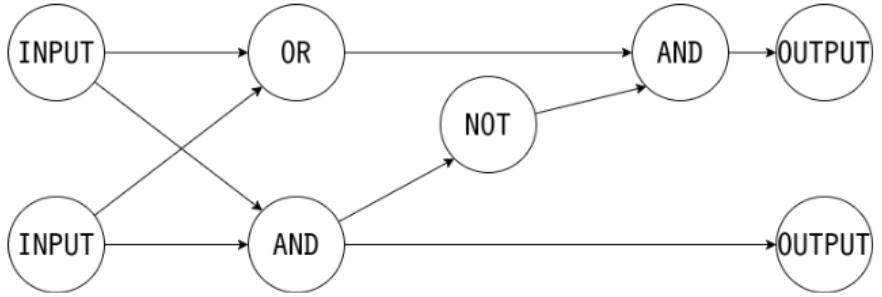


図 19: DAG に変換された回路

https://github.com/virtualsecureplatform/MitouDocument/blob/master/report.pdf より

論理回路とTFHE

- アプリケーション(回路)の開発手順

1. 回路の記述(Chisel)
2. 回路の試験
3. 回路の合成(yosys で全自動)
4. Iyokan を利用して回路を実行

- 回路を記述すれば基本的に任意の演算が可能