CSP記述からの非同期回路設計

首都大学東京
理工学研究科　数理情報科学専攻
計算システム研究室
修士2年　大橋　常毅

目次

- 研究背景
- 非同期回路
- 分散ワーカモデル
- マンデルブロ集合計算
- まとめ

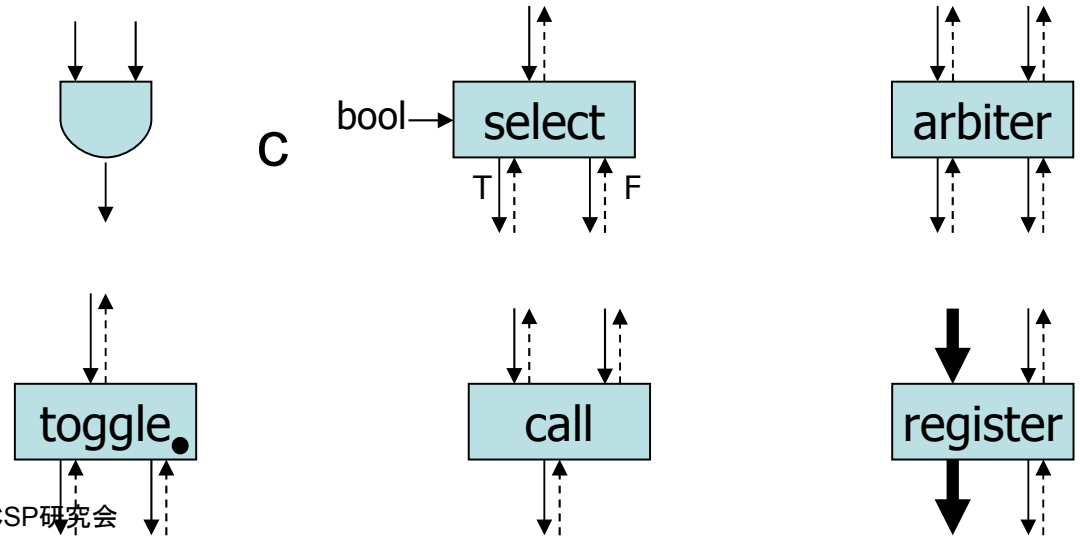
研究背景

- FPGAやHDLによりハード･ソフトの境界が曖昧に

  →両者を区別することなく設計できたら面白い
- CSP理論そのままな回路様式がある(Micropipeline)
- クロックの問題を解決するものとして非同期回路が注目されている

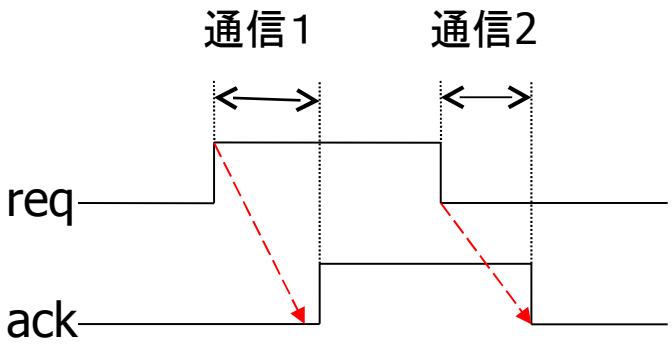
非同期回路

- Ivan Edward SutherlandのMicropipelineをFPGA上に実現
- 組合せ回路、非同期レジスタ、イベント制御素子で構成される
- ハンドシェークプロトコルで同期を取る



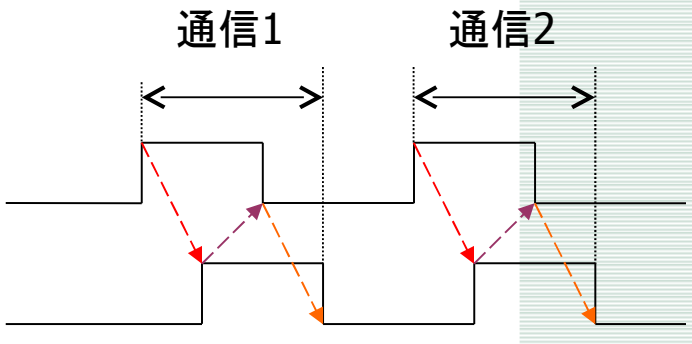
ハンドシェークプロトコル

- リクエスト・アクノレジ信号でタイミングを取る
- 2相式と4相式が一般的
- FPGAに実装するにあたり4相式を採用



2相式

信号の変化をイベントと取る



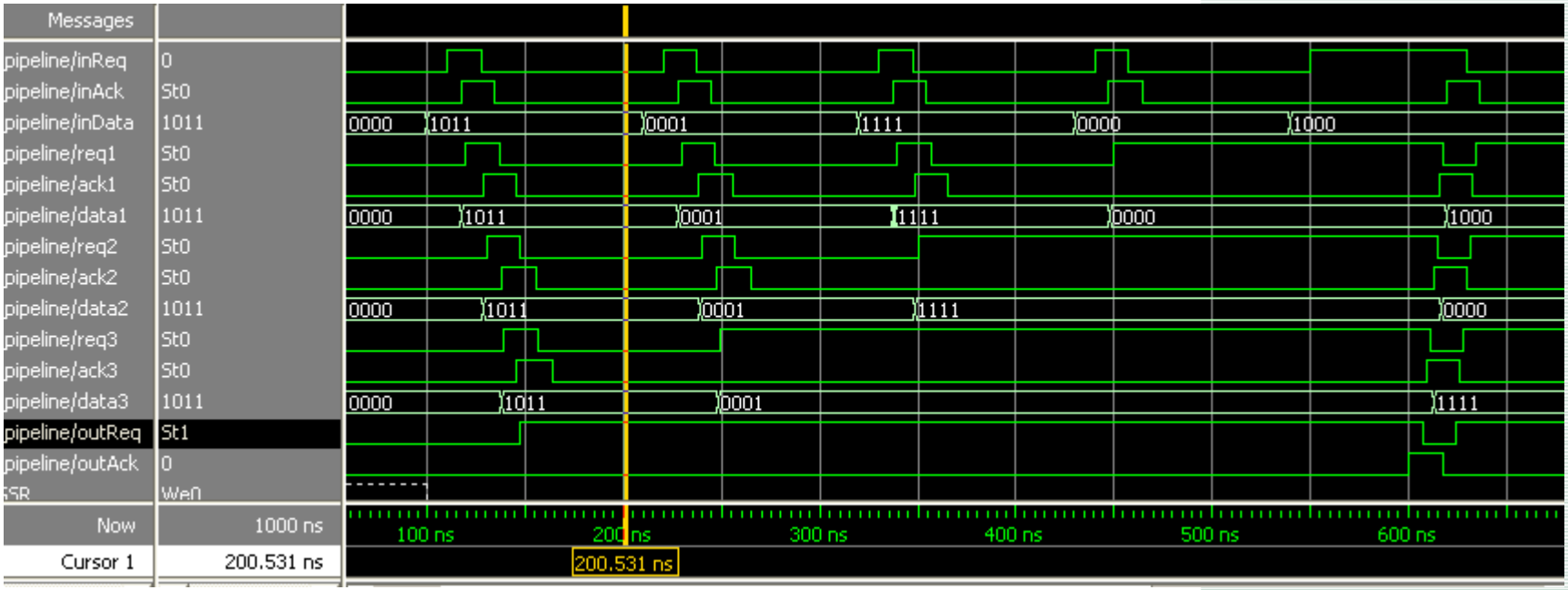
4相式

0→1の変化のみ

例　非同期FIFO

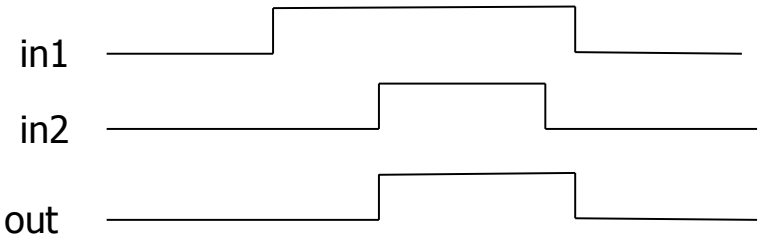
最終段がackを返さず、データが詰まっていく様子

同期式と異なり後段が遅れても終わるまで待つ



C素子

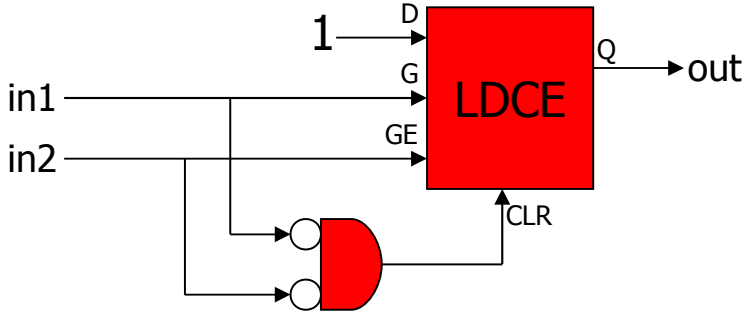
- 複数のイベントを同期させるときに利用
- ブールガードとして使えるがハザードに注意



in1	in2	out
0	0	0
0	1	-
1	0	-
1	1	1

C素子

- FPGAでの実装

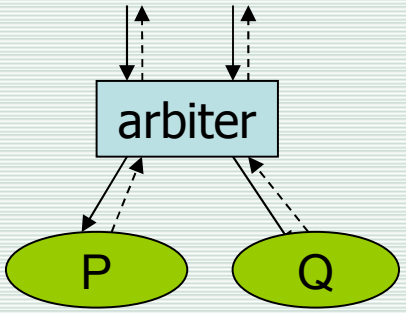


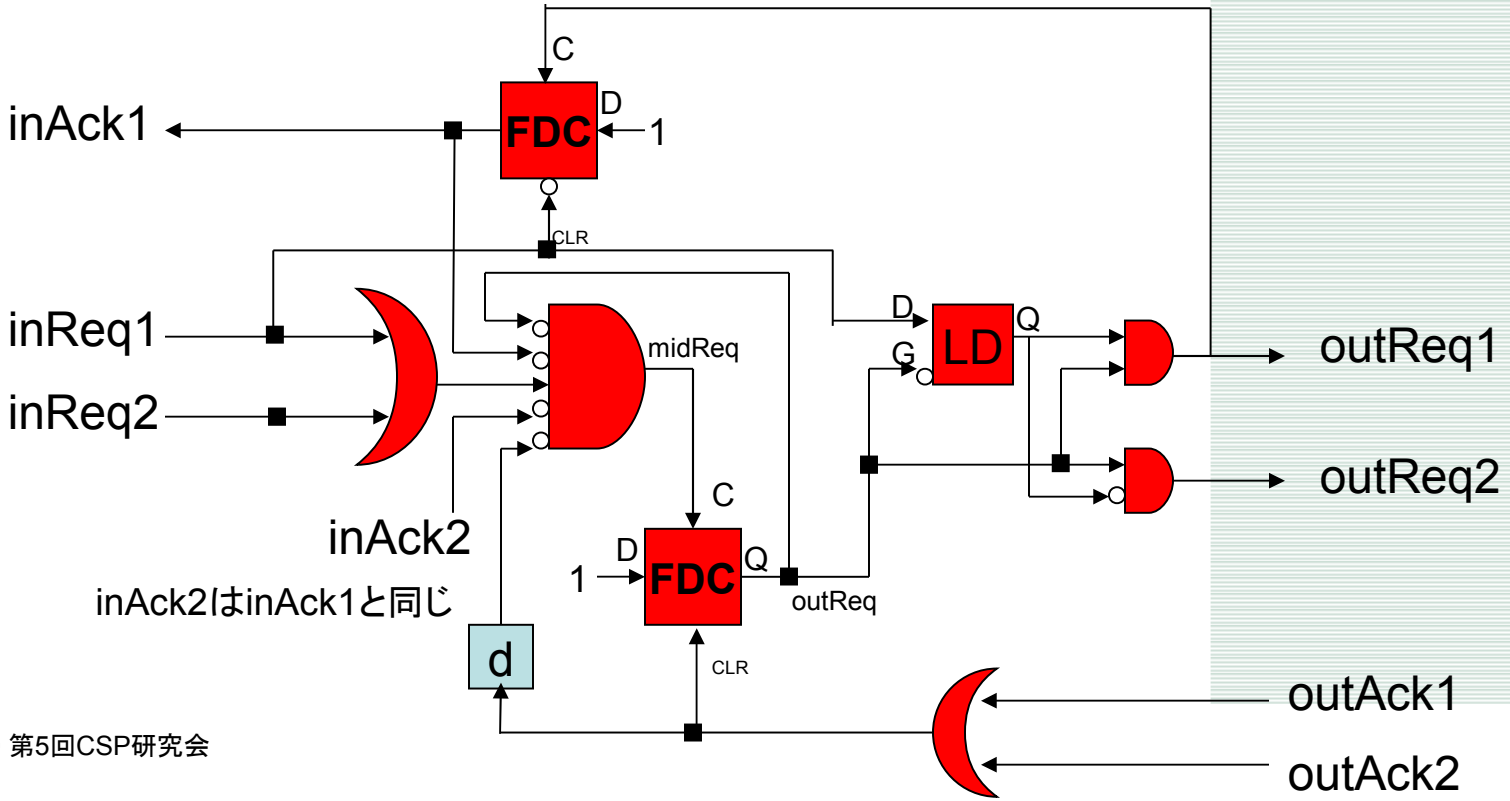
CLR	G	GE	D	Q
1	x	x	x	0
0	0	x	x	-
0	1	1	D	D
0	1	0	x	-
0	1	↓	D	D

LDCE

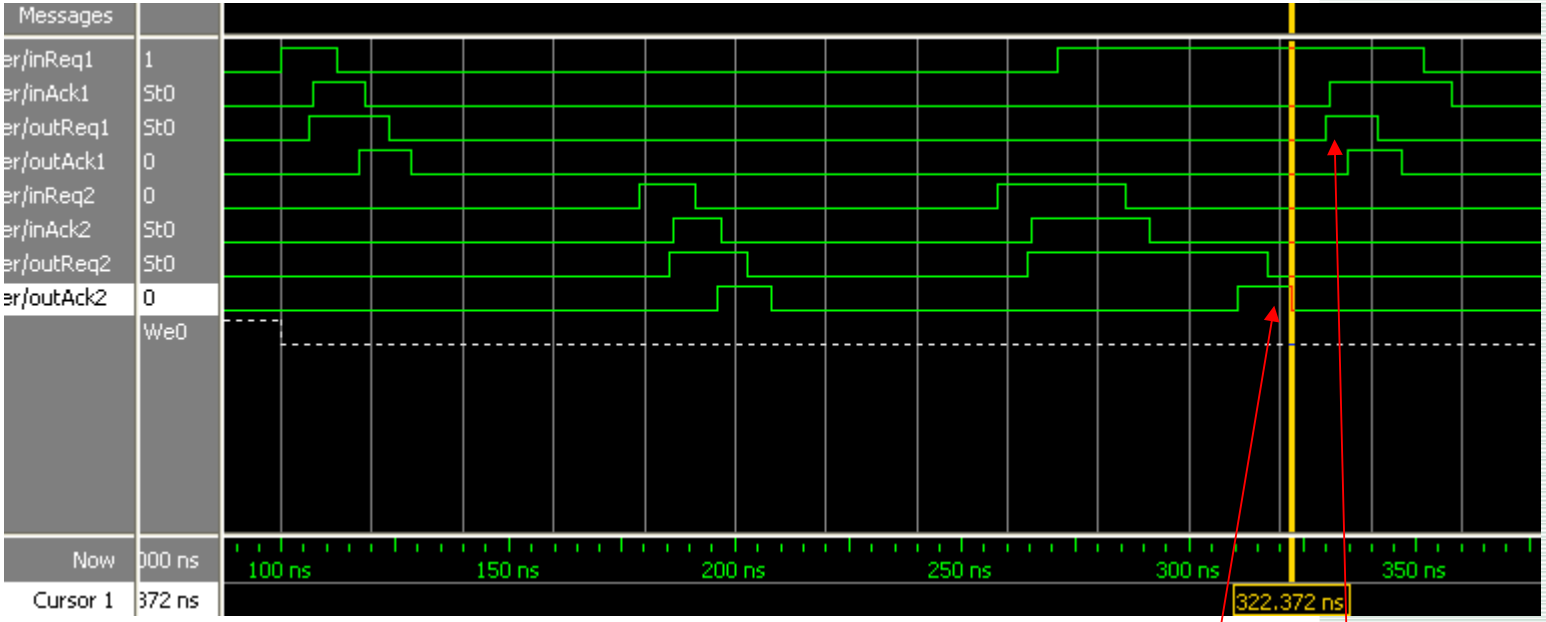
arbiter

- CSPの選択に相当
- 一方のハンドシェークが終わるまで他方をブロック





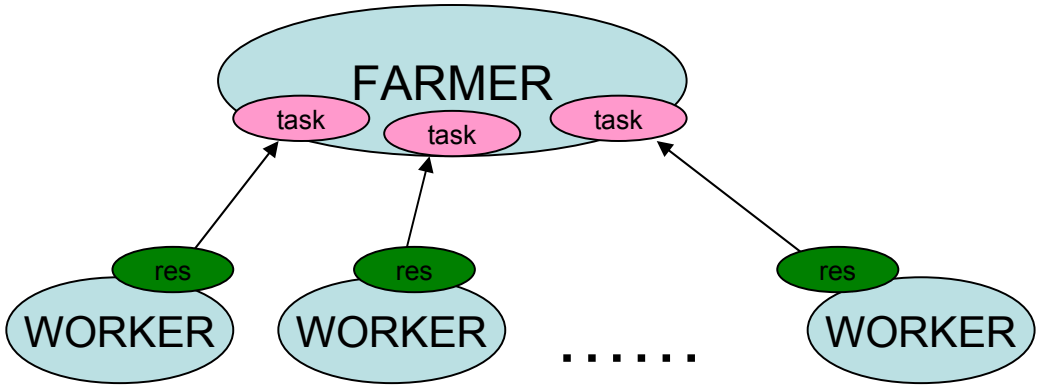
arbiter



req2に対するackが返るまでreq1をブロックしている

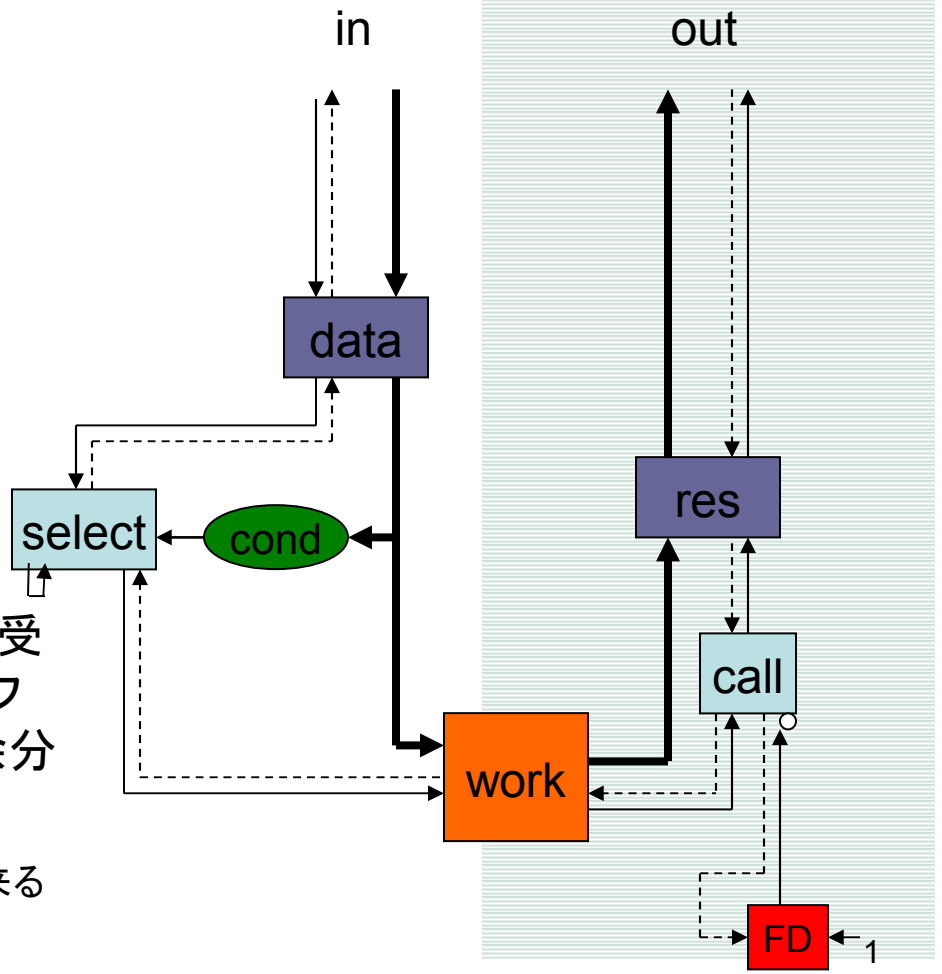
分散ワーカモデルの回路化

- 第1回研究会で山川氏が発表した分散ワーカモデルを回路にしてみる
- TPCOREの制限から来るMIDプロセスは省略



WORKER

```
WORKER(in,out) =
  let
    WORK =
      in?data ->
        if      data==task then
          out!work(task) -> WORK
        else
            SKIP
    work(task) = res
  within out!res -> WORK
```

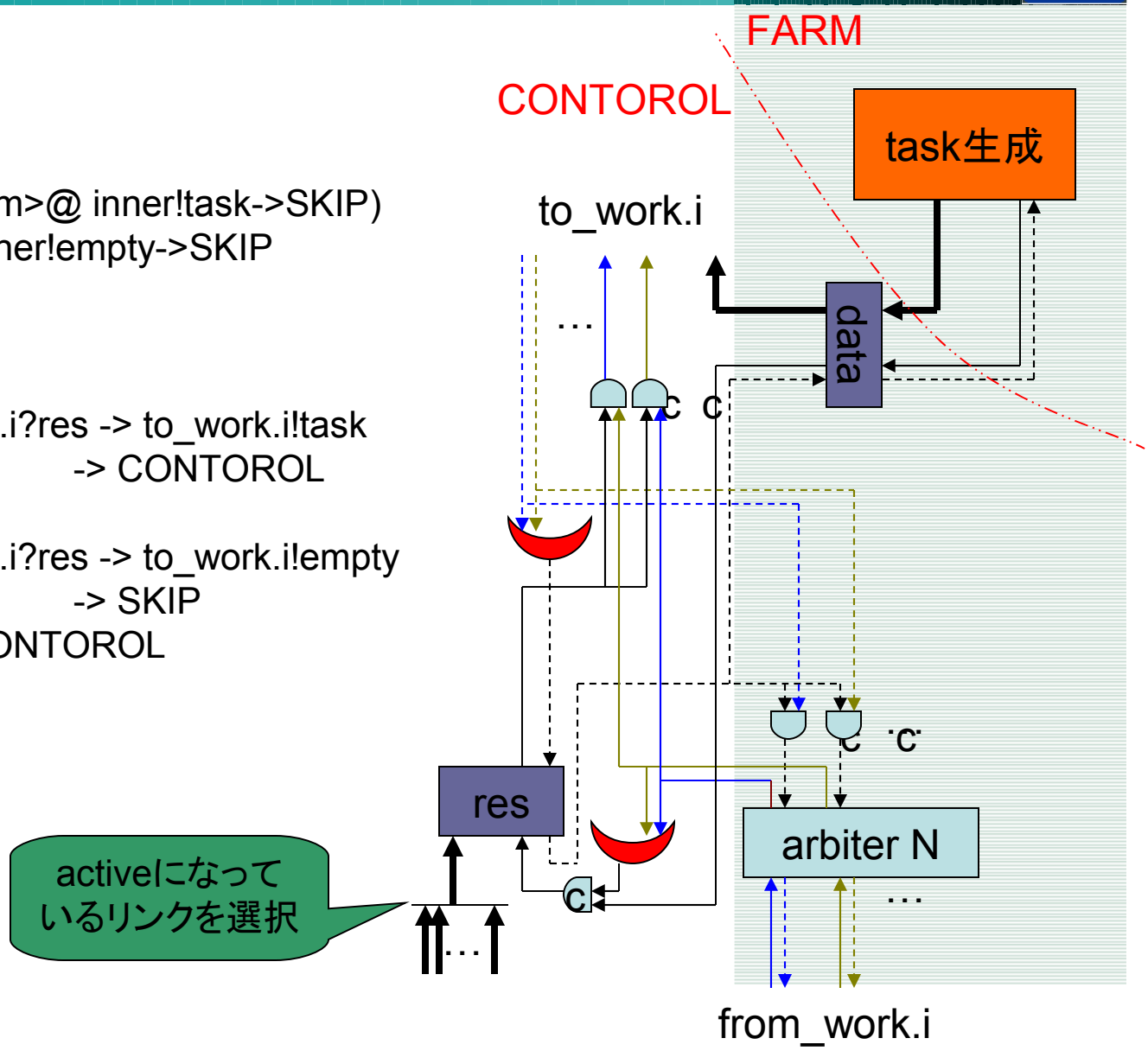


右の回路ではSKIP後もdataへの入力が受付可能だが、通常はdeadlock , livelockフリーであることを確認して実装するので余分な入力は起きないと考えてよい。

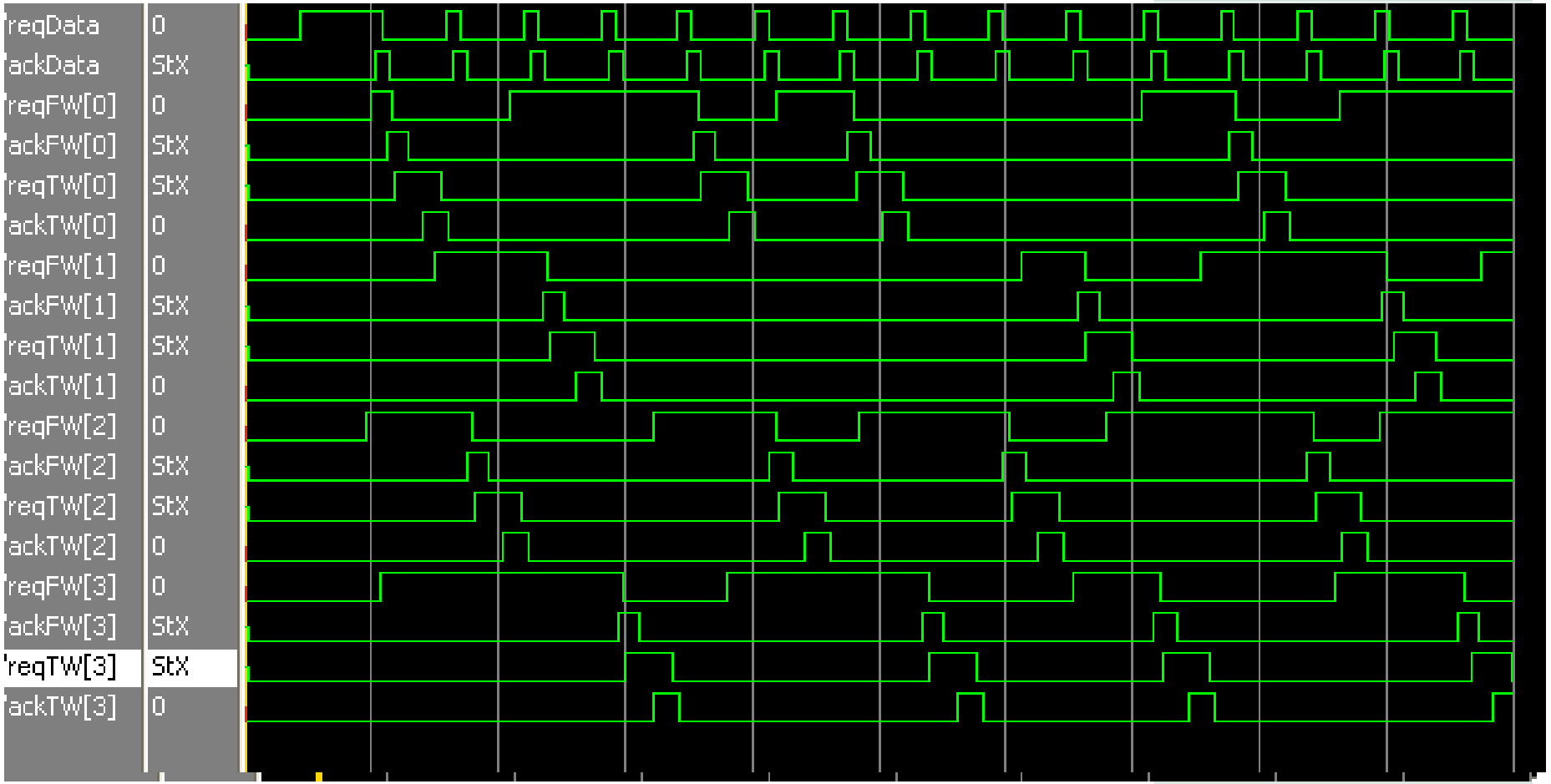
もちろん入力を受けつないようにロックすることも出来る

FARMER

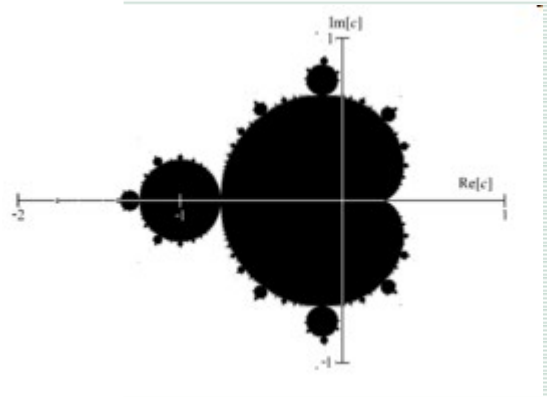
```
FARMER =
  let
    FARM = (;i:<1..TaskNum>@ inner!task->SKIP)
                           ; inner!empty->SKIP
    CONTOROL =
      inner?data ->
        if      data==task then
          []i:Tag@ from_work.i?res -> to_work.i!task
                                   -> CONTOROL
        else
          []i:Tag@ from_work.i?res -> to_work.i!empty
                                   -> SKIP
within FARM [|{|inner|}|] CONTOROL
```



FARMER動作


reqData 0
ackData StX
reqFW[0] 0
ackFW[0] StX
reqTW[0] StX
ackTW[0] 0
reqFW[1] 0
ackFW[1] StX
reqTW[1] StX
ackTW[1] 0
reqFW[2] 0
ackFW[2] StX
reqTW[2] StX
ackTW[2] 0
reqFW[3] 0
ackFW[3] StX
reqTW[3] StX
ackTW[3] 0

マンデルブロ集合計算例



- マンデルブロ集合とは漸化式

$$z_{n+1} = z_n^2 + c\,, z_0 = 0$$

で定義される複素数列 $\{z_n\}$ が

$n \to \infty$ で無限大に発散しない複素数c全体の集合

- タスクとして計算座標を渡し発散速度を返す

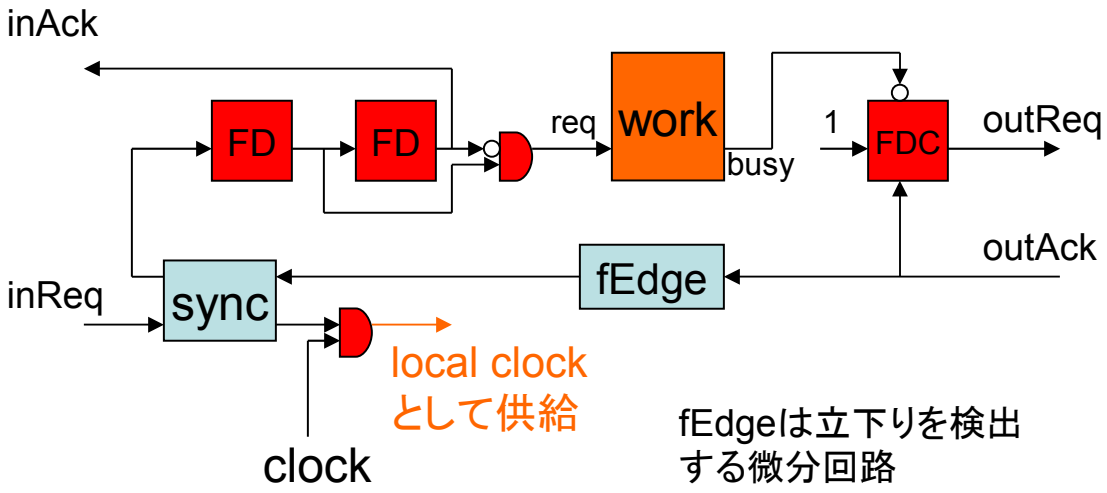
画像引用：wikipedia マンデルブロ集合

WORK

逐次的な処理は同期回路で組むほうが回路として設計しやすいし合成ツールの恩恵を受けやすい。

そこで非同期-同期インターフェースを作り非同期回路の中に同期回路を取り込むことを考える。

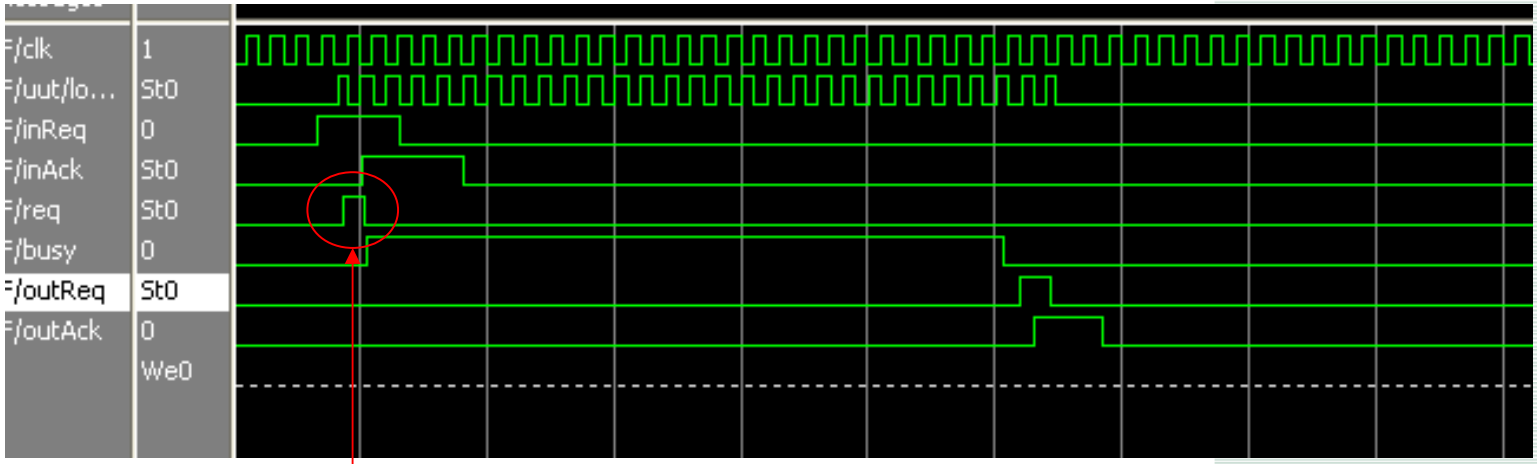
(同期式workは(同期)reqと動作中を示すbusy信号を持つものとする)



計算アルゴリズム

```
int work(float cx,cy){
   float x=0 , y=0 , nextx;
   for(int i=0;i<MAX;i++){
      nextx=x^2-y^2+cx;
      y=2xy+cy;
      x=nextx;
      if(x^2+y^2>4) return i;
   }
   return -1; //収束
}
```

非同期IF



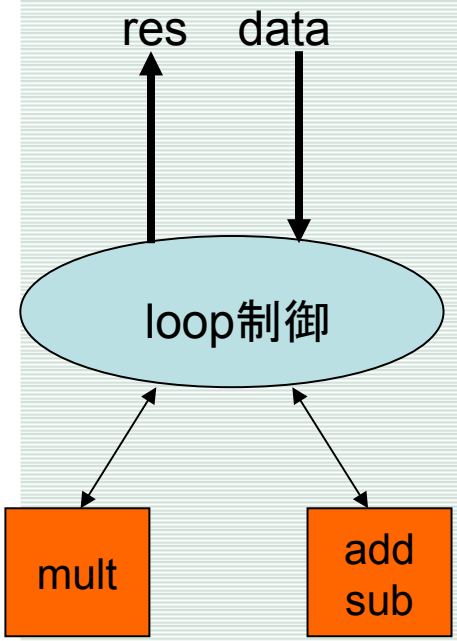
クロック同期リクエスト

内部へはリクエストが来てからクロックが供給され、同期回路へは完全にクロックに同期したリクエストが送られている

WORK

- 非同期インターフェースを用いれば任意の段階まで同期回路を分割することが出来る
- これによりクロックドメインを分割することが容易になり、それぞれのブロックを最大速度で動かせる

(乗算器300MHz, 加減算器150MHzで動かせる)

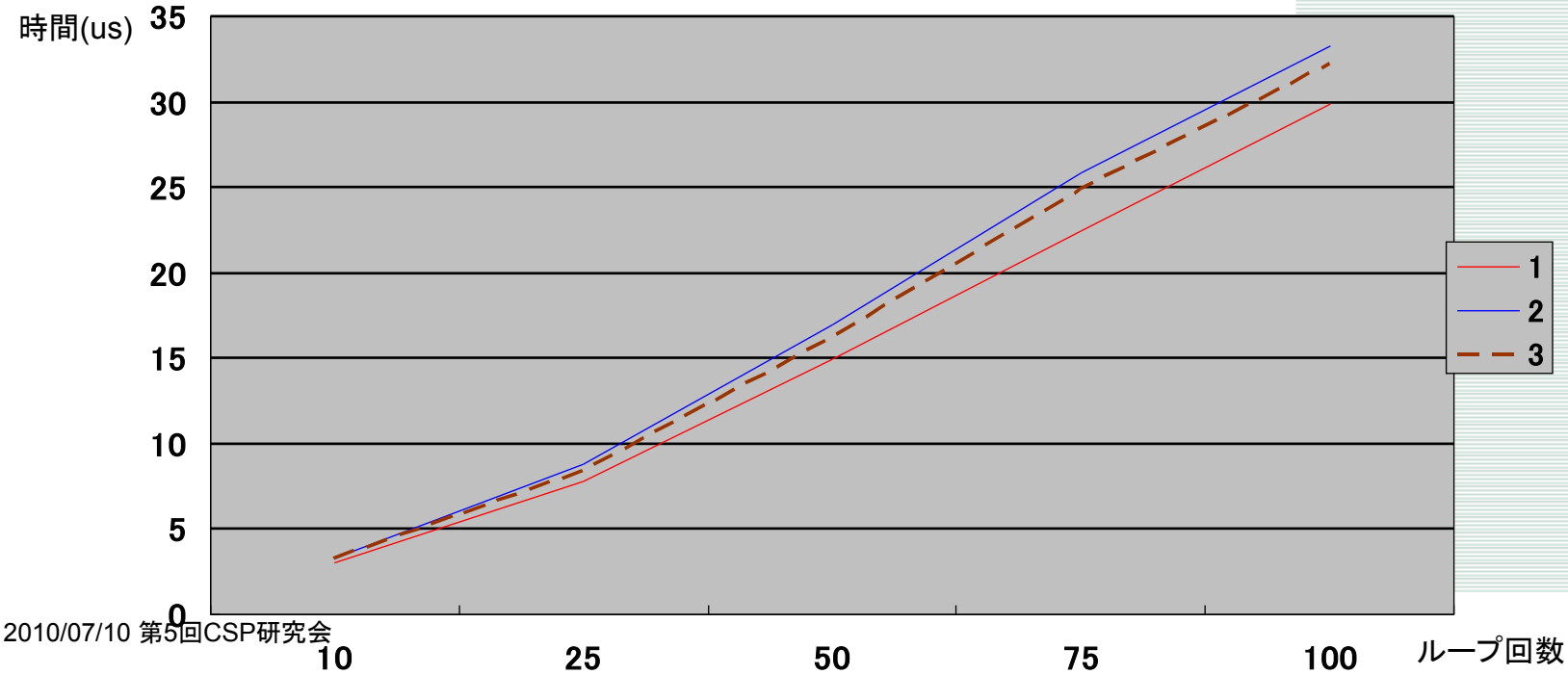


乗算器と加減算器を分割

WORK計算時間

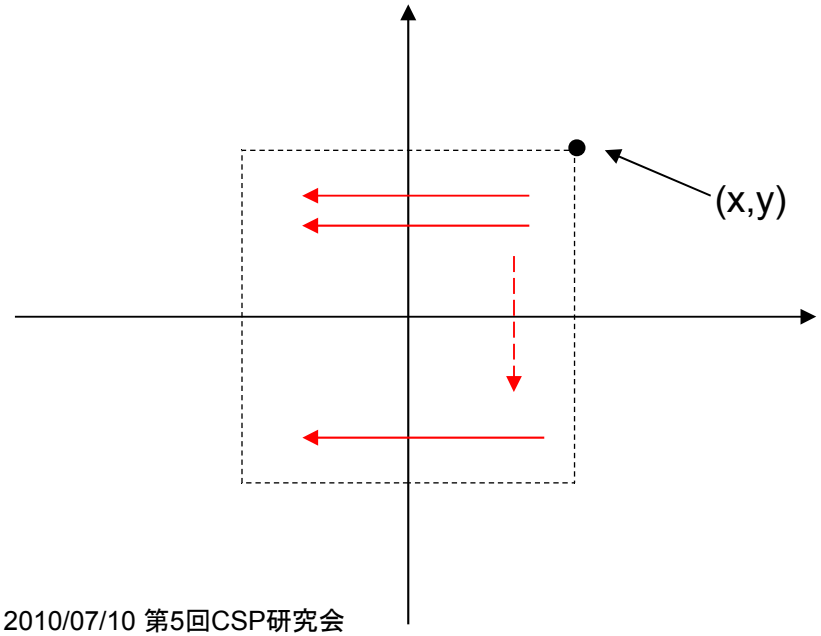
- WORK部分を
  1.完全同期回路(150MHz)
  2.乗算・加減算器に分割し両方を150MHzで動かす
  3.300MHz,150MHzで動かす

で比較



FARM

- 基準点(x,y)から(-x,-y)の長方形領域を間隔dで引き算して座標生成
- 加減算器一つであるなら完全な逐次処理となるので同期回路で実装



点線領域内をひたすら矢印に沿って引き算

複数の加減算器を使えば行ごとに座標を計算できる

まとめ

- イベント制御素子を作ることでCSP理論を回路設計に応用できる
- 非同期IFで同期回路も非同期回路の利点を引き継げる
- 非同期回路に不向きな所は同期回路に任せることで柔軟に回路が作れる

- 現状、非同期回路は回路図的に作成しているので非常にめんどくさい!