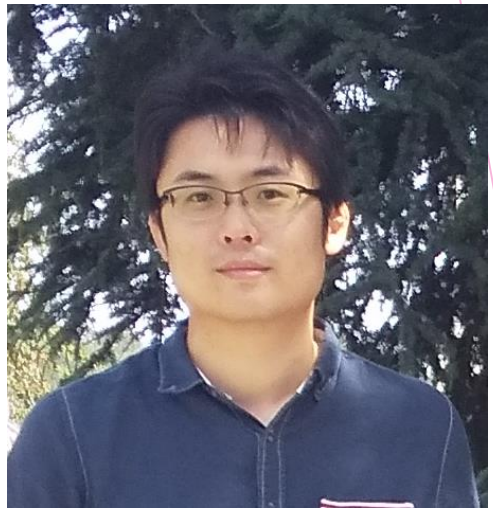
デジタル信号処理

2020年7月29日

本多良太郎（東北大学）

自己紹介

2014.09　　東北大学 理学研究科 学位取得
2014.10-2017.03 大阪大学 理学研究科 ポスドク
2017.04-　　東北大学 理学研究科 助教



専門
- 原子核物理・ハドロン物理

やっていること
- 主にFPGA内の回路や基板設計
- ソフト開発も少し

最近コロナのせいで飲み会が無くて非常に寂しい

講義全体の内容

1. デジタル技術概要
   - アナログ回路とデジタル回路
   - フロントエンド回路の役割
   - 2進数と論理演算
2. デジタル回路
   - 信号処理
   - 組み合わせ回路
   - 記憶素子
   - 順次回路
   - 同期回路のタイミング
   - メモリ
   - ステートマシン
3. データ通信技術
   - システムバスの概要
   - データ通信の基礎
   - 装置内通信
   - 装置間通信
4. デジタル回路の例
   - Common-start-type single-hit TDC

１．デジタル技術概要

2020年7月29日
本多良太郎（東北大学）

デジタル技術概要の内容

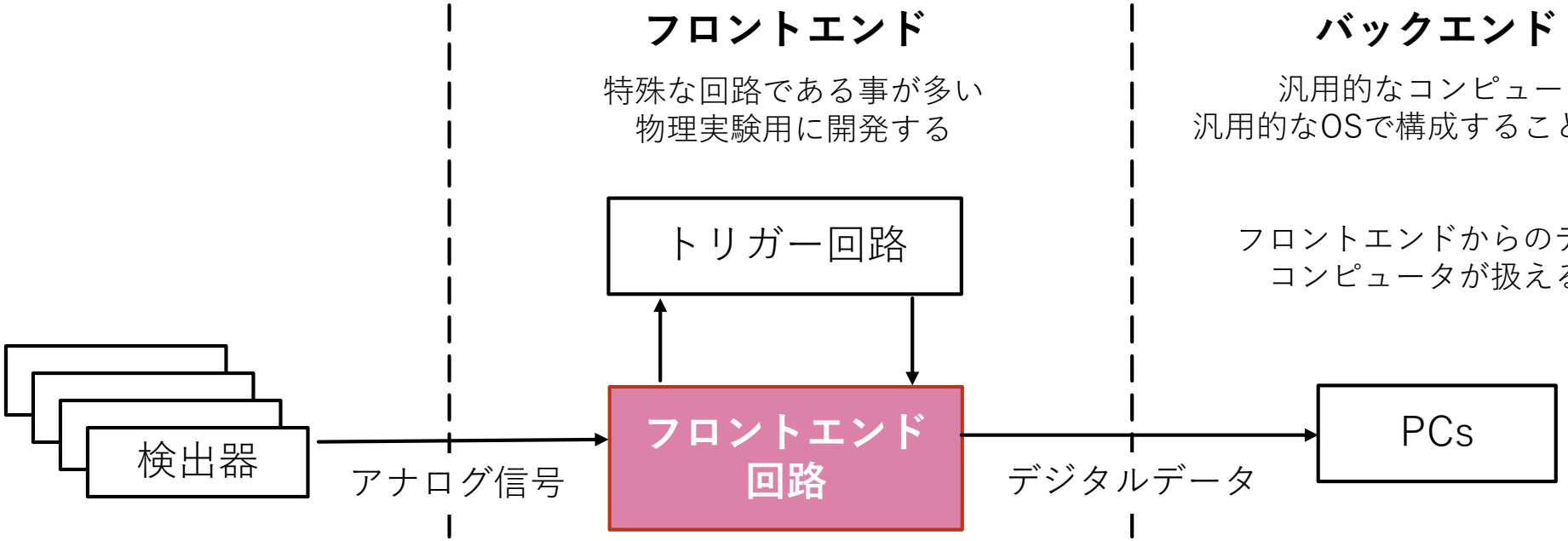
まず最初にデジタルとは何であるか？を学びます

- デジタル化が必要な理由
- 予備知識として2進数の処理について

1.1　アナログ回路とデジタル回路

データ収集（DAQ）システム

検出器のアナログ信号をデジタル化しコンピュータへ保存する前の一連のシステム

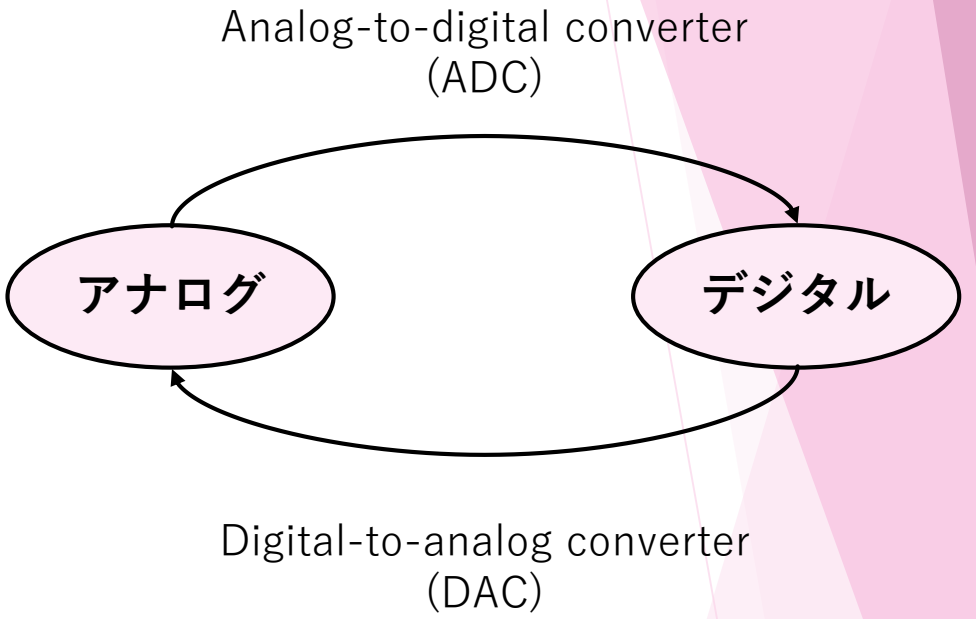


アナログ信号をデジタル化
1つの回路でやることもあるし複数に分けることもある
ところで、そもそも**デジタル化**って何だろう？

アナログとデジタル

アナログ
- 物理量（例：電圧）を連続量のまま取り扱う

デジタル
- 離散値（2進数値）を取り扱う



世の中で進むデジタル化
- レコード CD
- 測定器
- テレビ放送

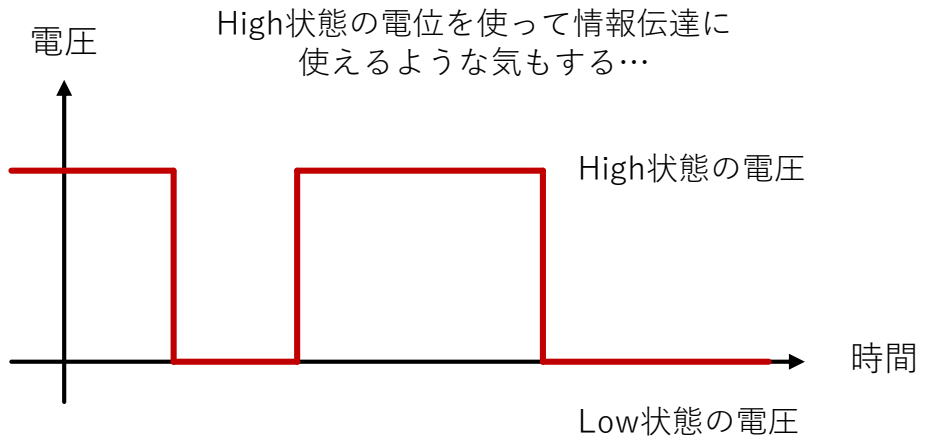
可能な限りデジタルで処理し**アナログの範囲を小さくする**のがトレンド
検出器システムも例外ではない

どうして？

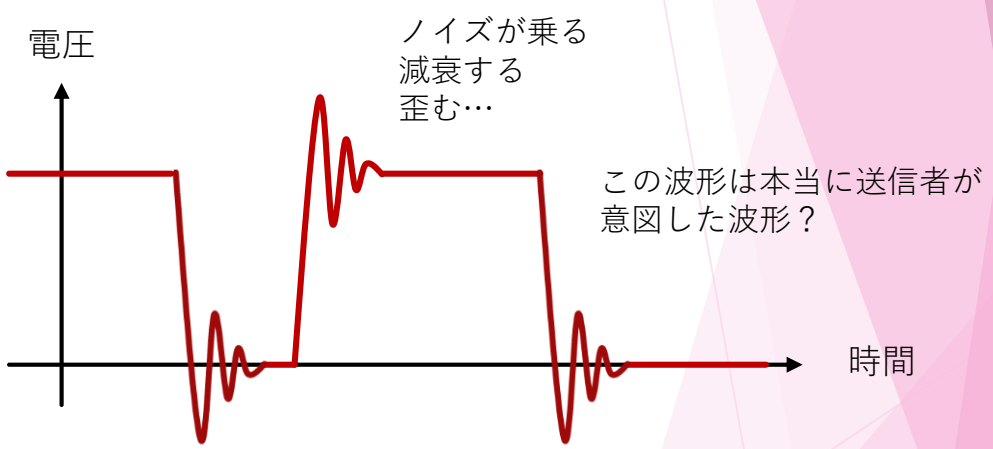
デジタルはノイズ (歪み) に強い

アナログ的に考えてみる

理想的な信号波形



実際の信号波形

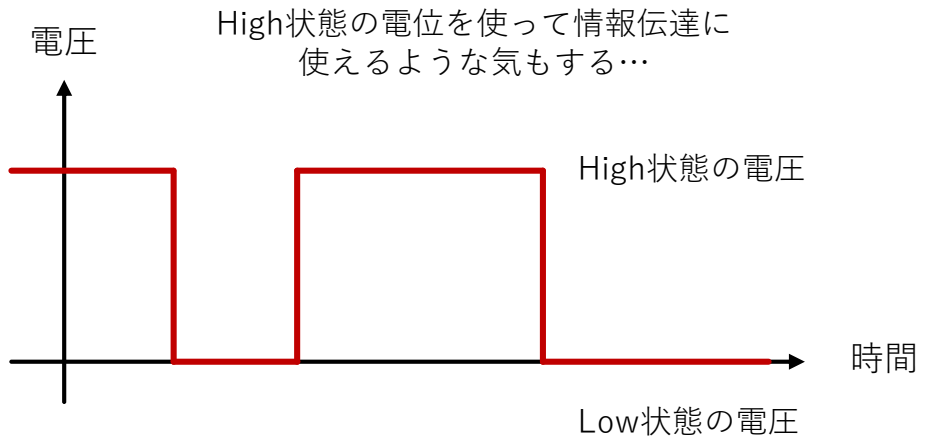


アナログでは信号形状そのものに意味がある！
ノイズや減衰で形が変わってしまったらそれだけで情報を失う！

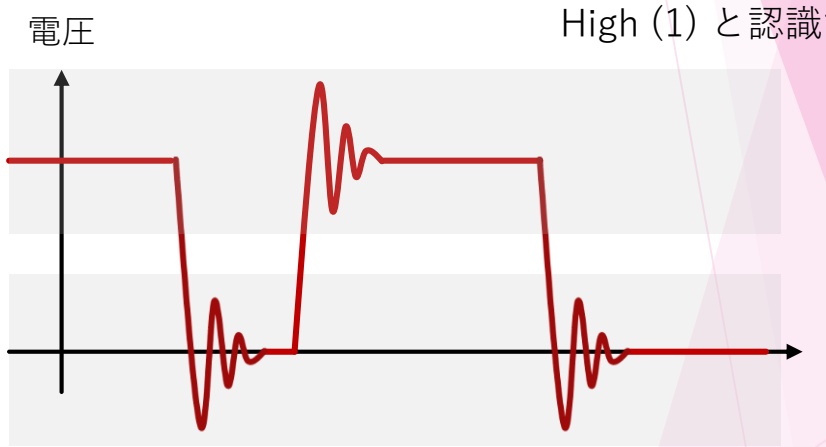
デジタルはノイズ（歪み）に強い

デジタル信号では

理想的な信号波形

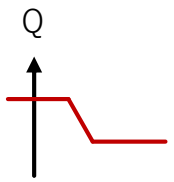


実際の信号波形

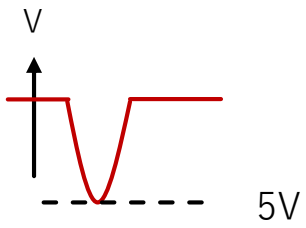


デジタル信号では多少信号波形が崩れても
間違った情報を伝えることがない！

物理実験の検出器では

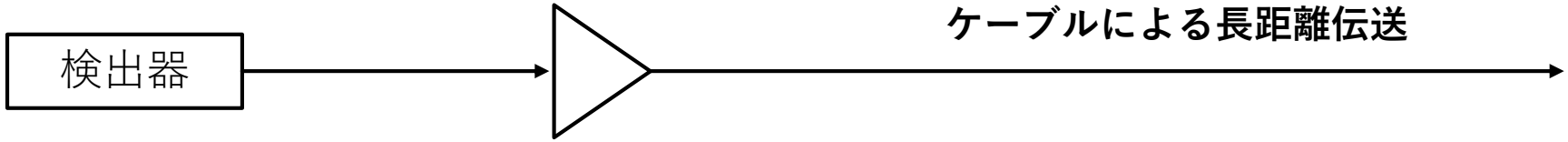


検出器からの微弱な信号
すぐに増幅したい



成形増幅後の
扱いやすい信号

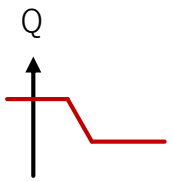
減衰により元の形が変わってしまった
ノイズも拾うかもしれない



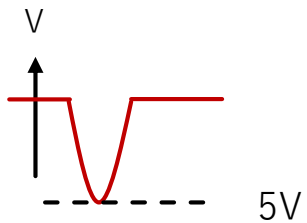
アナログ信号処理

- 成形
- 増幅

物理実験の検出器では



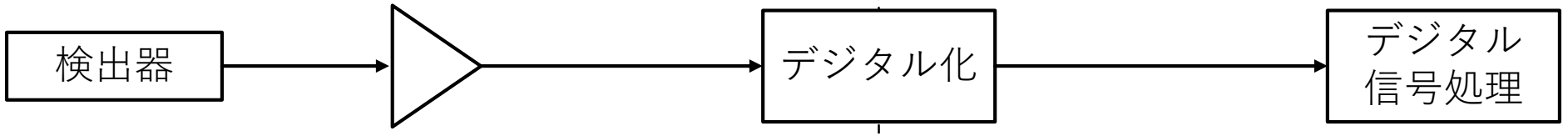
検出器からの微弱な信号
すぐに増幅したい



成形増幅後の
扱いやすい信号

ADC data
b0101

成型増幅直後の波高情報
減衰・ノイズの影響を受けない



アナログ信号処理
- 成形
- 増幅

ADC
- 多値へ変換

コンパレータ（ディスクリ）
- ヒット判定
- タイミング信号

- トリガー処理
- イベント選択
- デジタルフィルタ
- データ転送

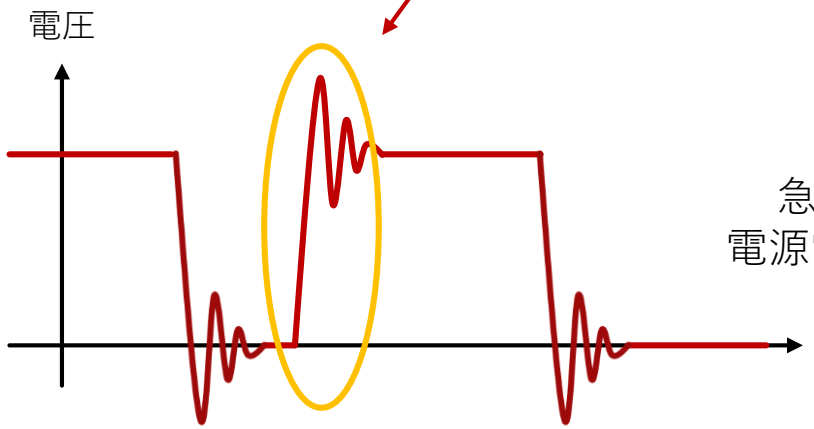
アナログ回路
出来るだけ小さく≒On-detector

デジタル回路

なぜ近年デジタル化が進んだ？
集積回路の技術進歩
高密度・小面積・高速動作
が可能になった

デジタル回路の欠点

信号を高速にスイングさせるために
ノイズが発生しやすい

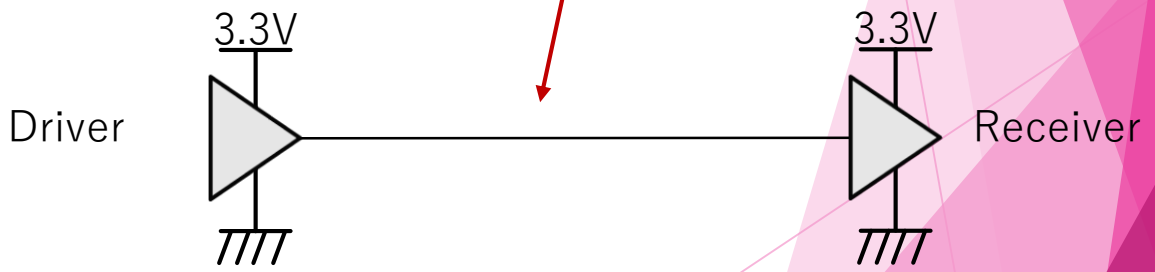


高速に動けば動くほど
単位時間当たりの電流変化が大きい

伝導ノイズ＋放射ノイズ

急激に電流が流れるために
電源電圧が（局所的に）降下する

急に電流が変化するため
電磁波の放射が発生

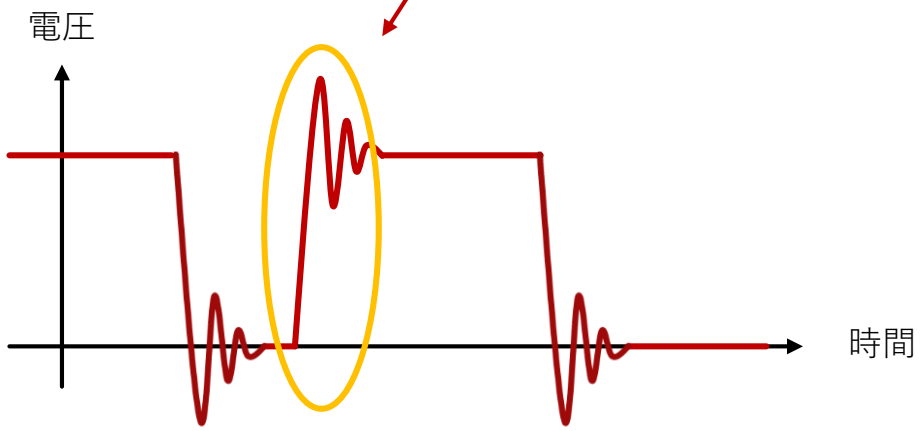


電源バイパスコンデンサ（パスコン）

信号を高速にスイングさせるために

ノイズが発生しやすい

（電圧降下）



高速に動けば動くほど
単位時間当たりの電流変化が大きい

急激な電流変化に耐えられるように
ICの電源ピン近くにエネルギーを蓄えておく。
電源インピーダンスを下げるという。

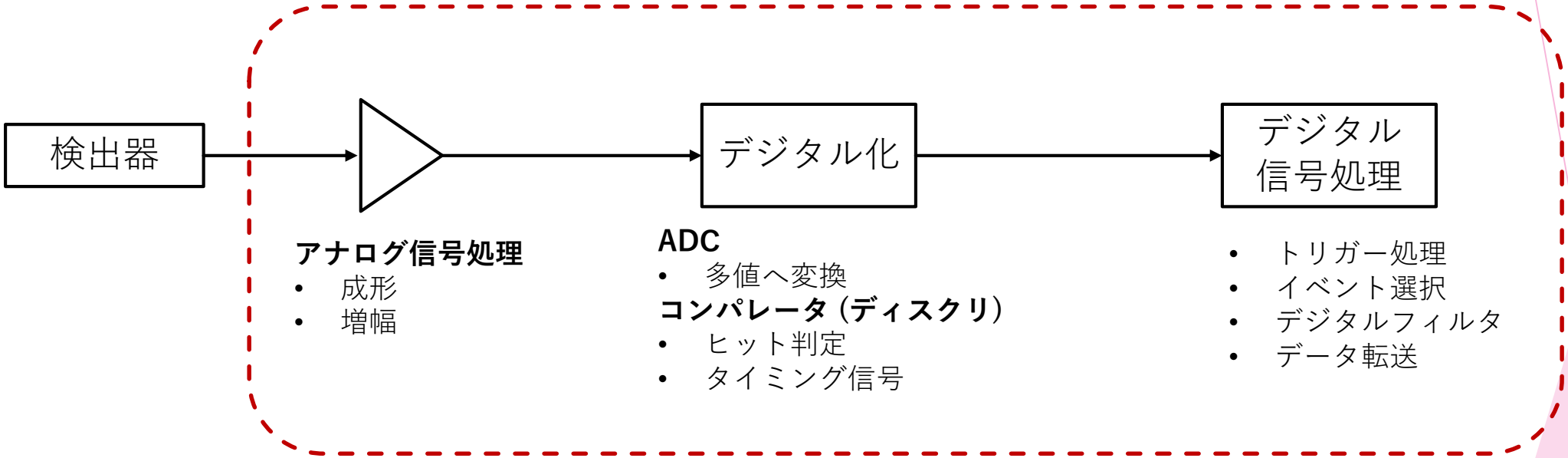
デジタル回路の得意なところ

- 大容量の記憶装置を使う事が出来る
  - 過去の情報をもとに動作を決定することができる
  - 情報を一時保存しておいて後から必要になったら取り出す
  - アナログ情報をそのまま記録するアナログメモリも存在するが大型化は難しい
- 情報の取捨選択が用意
  - 要らないデータは即破棄できる
  - アナログ情報を破棄するには減衰時間が要ることが多い
    - 強制短絡させるとノイズが出る
- 計算テクニック（数学）を使ってアナログ回路ではできない処理ができる
  - デジタルフィルタやエラー訂正など

1.2　フロントエンド回路の役割

フロントエンド回路

これ全体をフロントエンド回路と呼ぶ



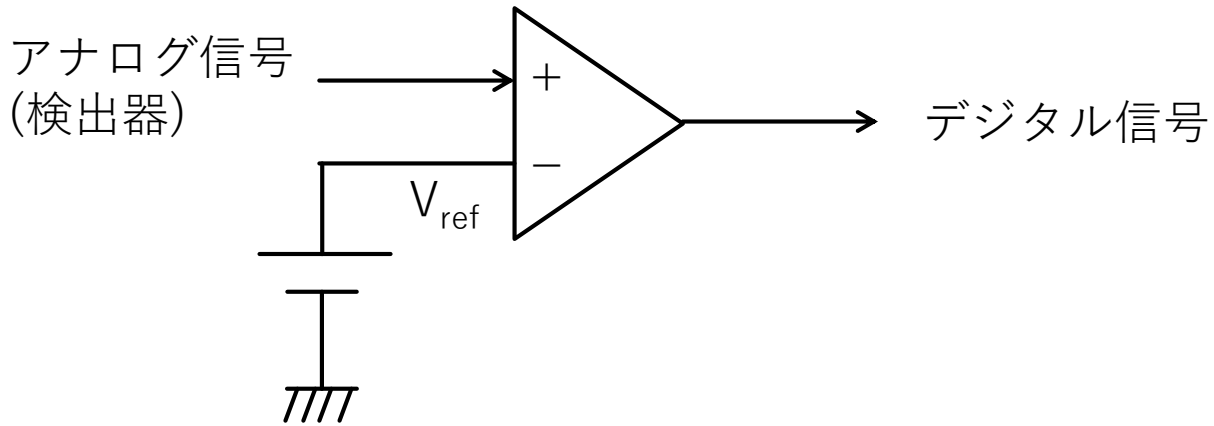
1つの基板で全部行うこともあれば
複数に分けることも

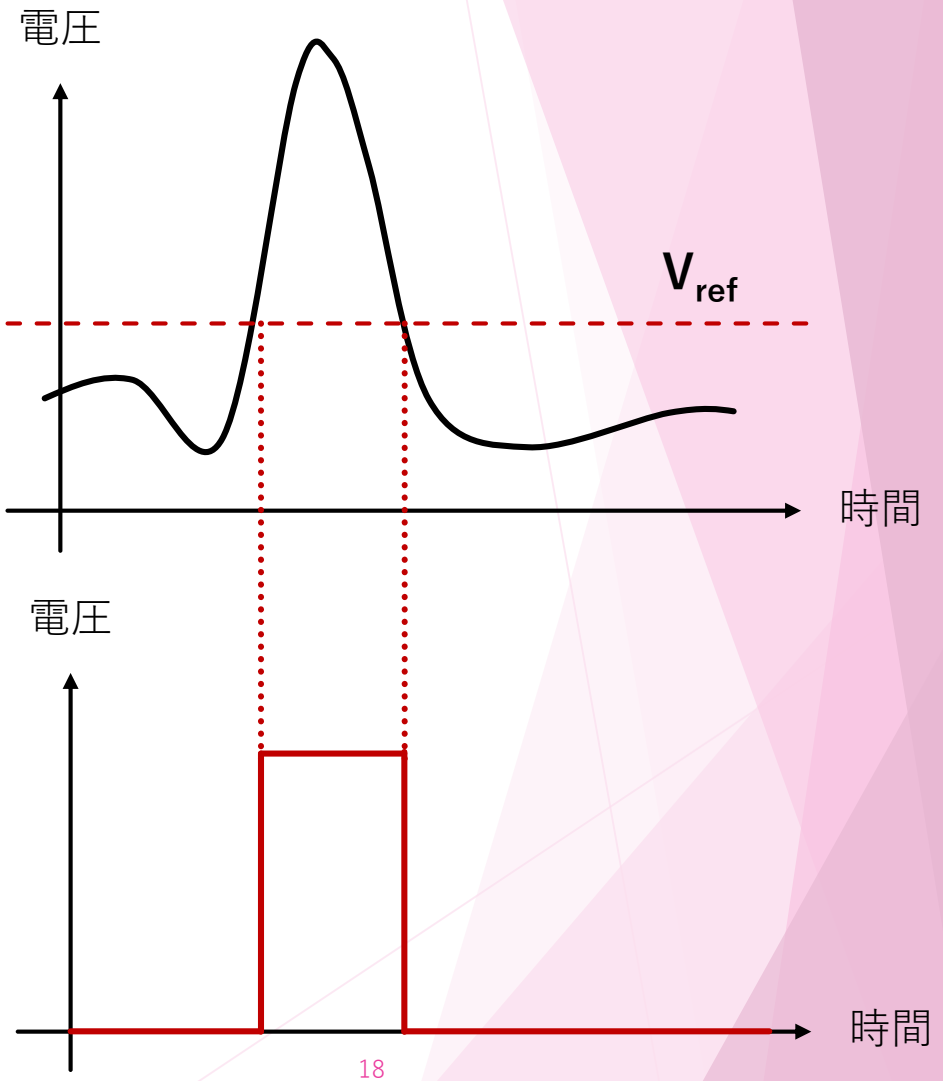
デジタル化：コンパレータ（ディスクリミネータ）

信号が基準電圧を超えたかどうかだけを調べる
- タイミング信号
- ヒット検出

などに使用する

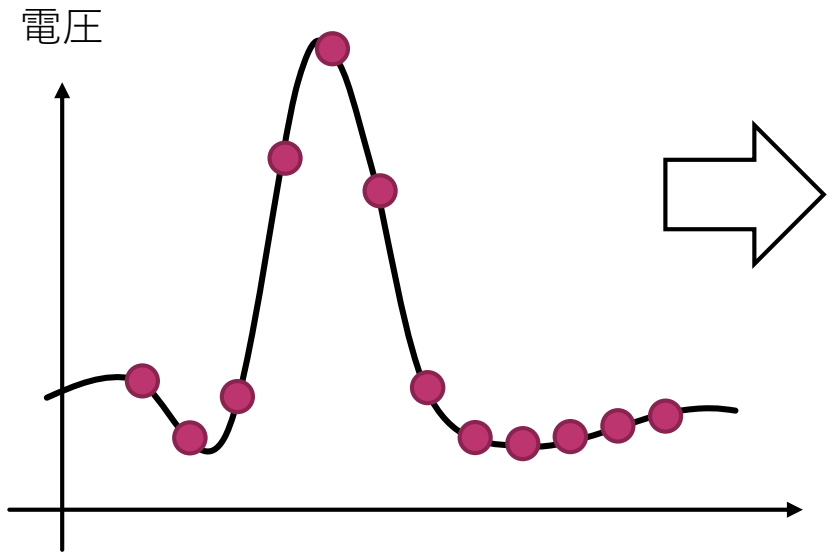
コンパレータ
（基準電圧と比較）





デジタル化：ADC

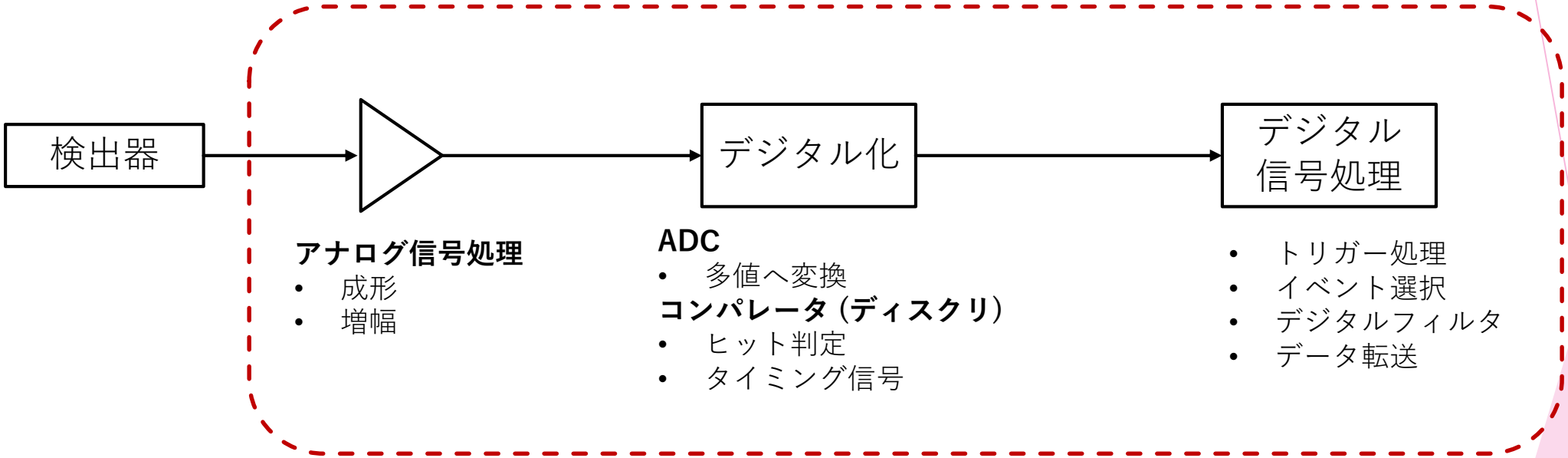
波形をあるサンプリング周期デジタル値に変換
（1つ前の講義で詳しくやったと思う）



各点の電圧をデジタル値へ変換
解析で波形情報を再構成

フロントエンド回路

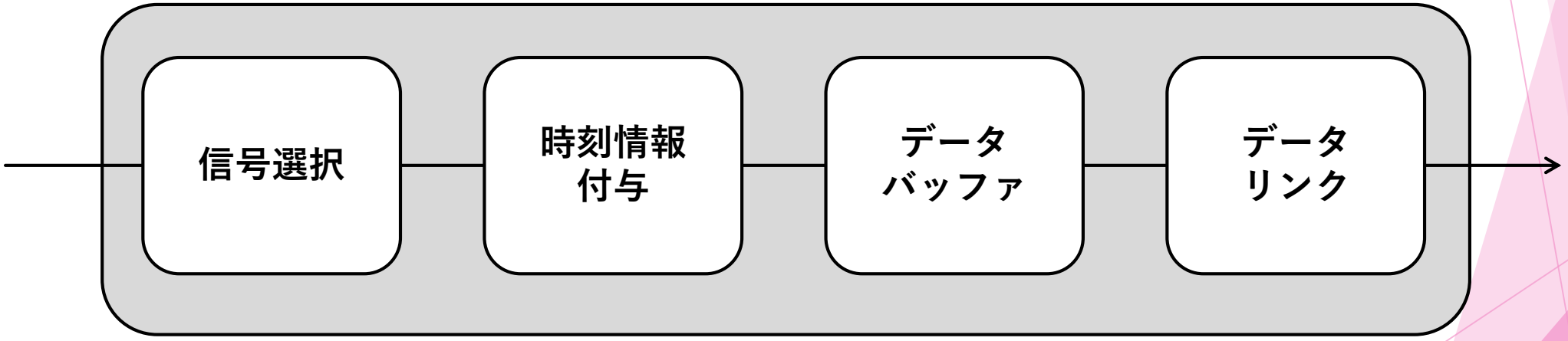
これ全体をフロントエンド回路と呼ぶ



1つの基板で全部行うこともあれば
複数に分けることも

デジタル信号処理

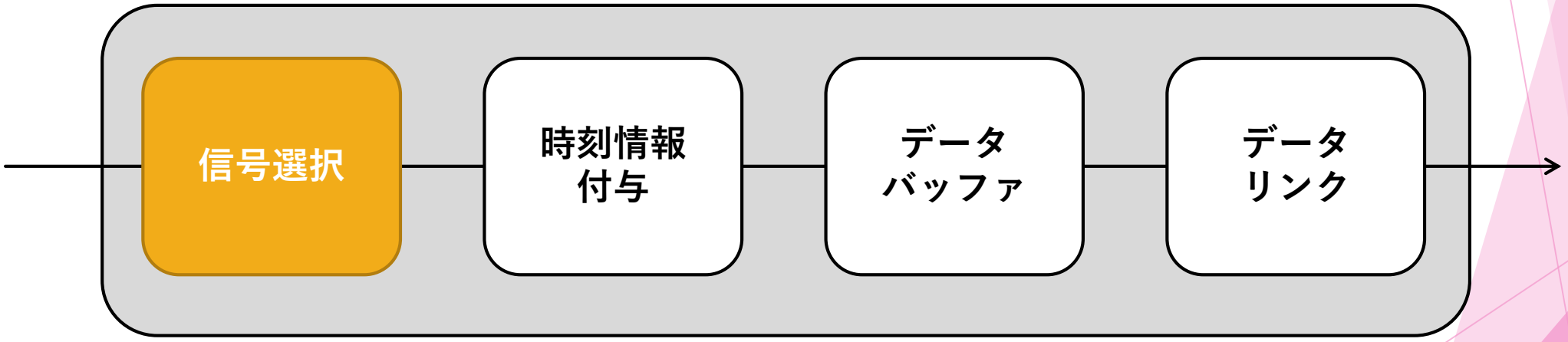
多くのフロントエンド回路に含まれている機能
（あくまで一例）



信号選択

データ量を削減する機能
- A/D変換したデータ全てが価値あるとは限らない
- データリンクの帯域やデータストレージには限りがある

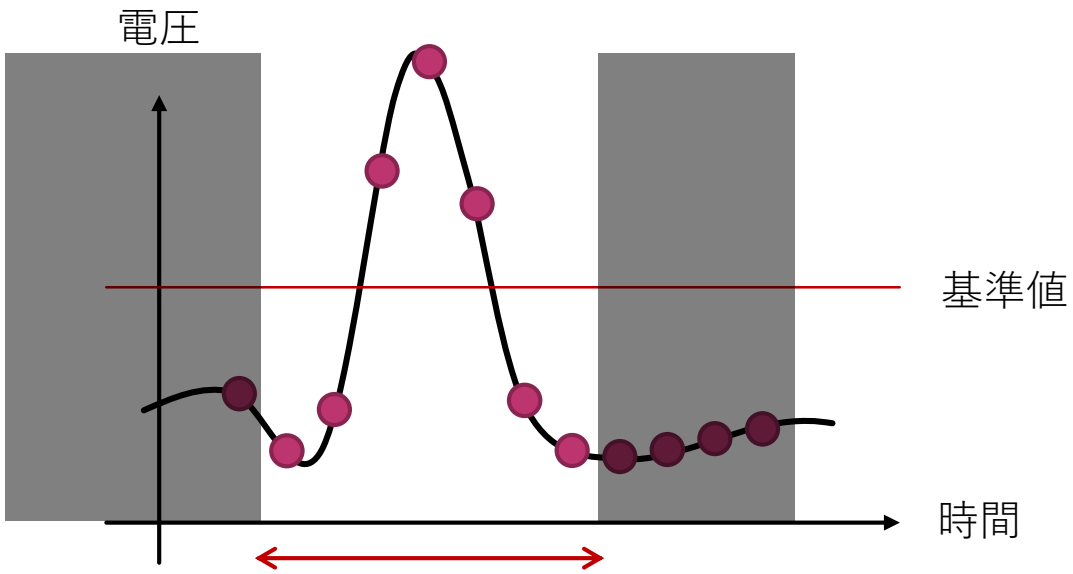
典型的には
- ゼロサプレス
- TDC window



信号選択：ゼロサプレス

例えば…ベースライン情報には意味が無い場合
- 基準ADC値を超えた部分だけデータ転送すればよい

基準値を超えた点の前後2点まで
データ転送する場合



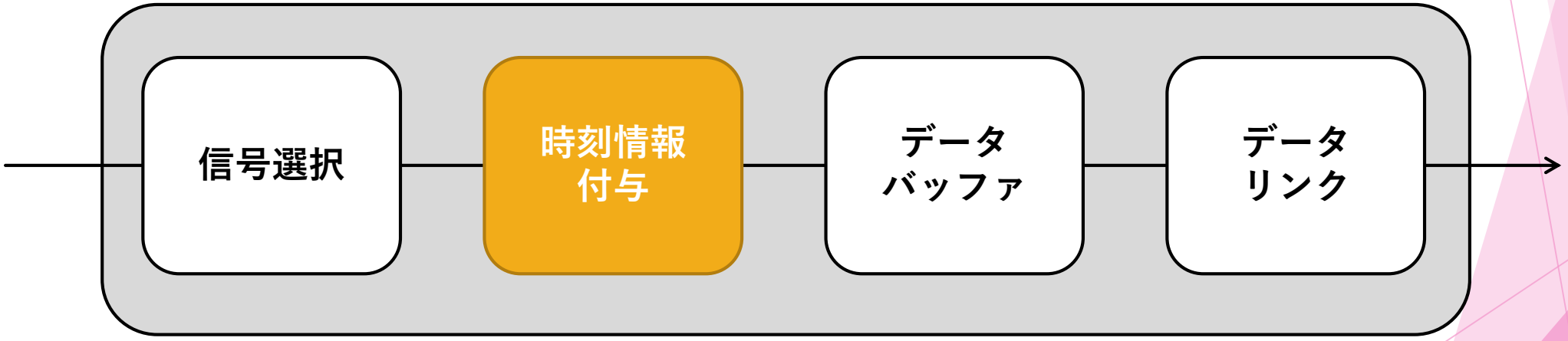
波形データは莫大なデータ量を生み出す
データリンクが全てのデータを
転送できるとは限らない

ここだけデータ転送する⇔他の点はA/D変換されたが捨てられる

時刻情報付与

物理解析では各回路のデータを集約して、それがいつ起こったか？を再構成しないといけない
ところで…全員同じ速度で動いてる？
- サンプリング周波数が回路間で違ったら大変

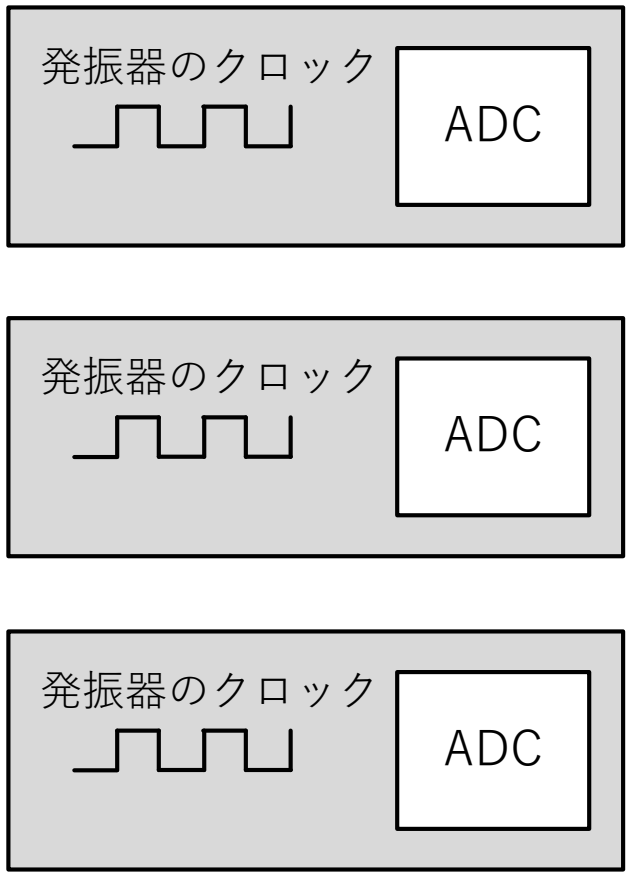
ところで…このデータって何番目のバンチクローシングで出たデータ？
- ADCはそんなこと知らない



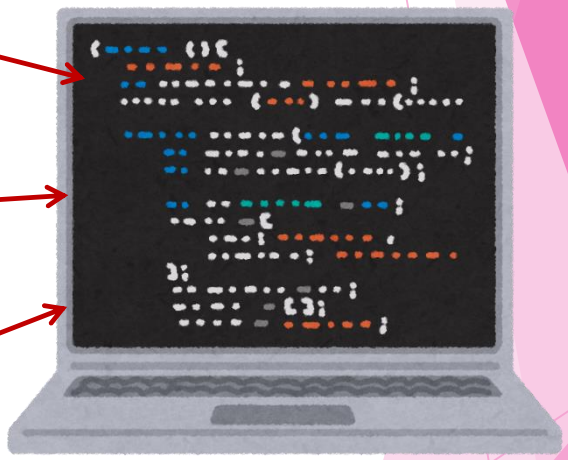
時刻情報付与

発振器のクロックは同じにならない

サンプリング周期が異なる



時刻の再構成ができない！



この波形データがいつのデータなのか

保証する物がない

時刻情報付与

マスタークロック分配

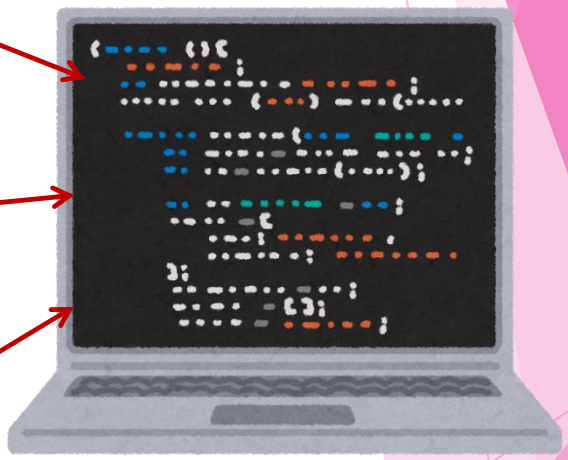
グローバルタグ
も配ろう

ADC

ADC

ADC

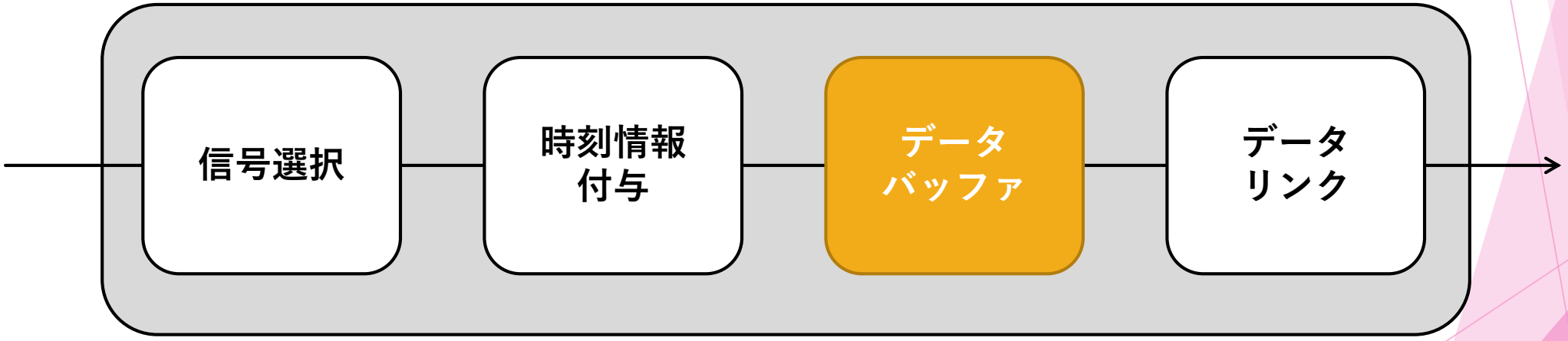
サンプリング周波数が揃っているから安心！
グローバルタグを頼りにどの事象に属しているのか調べられる



データバッファ

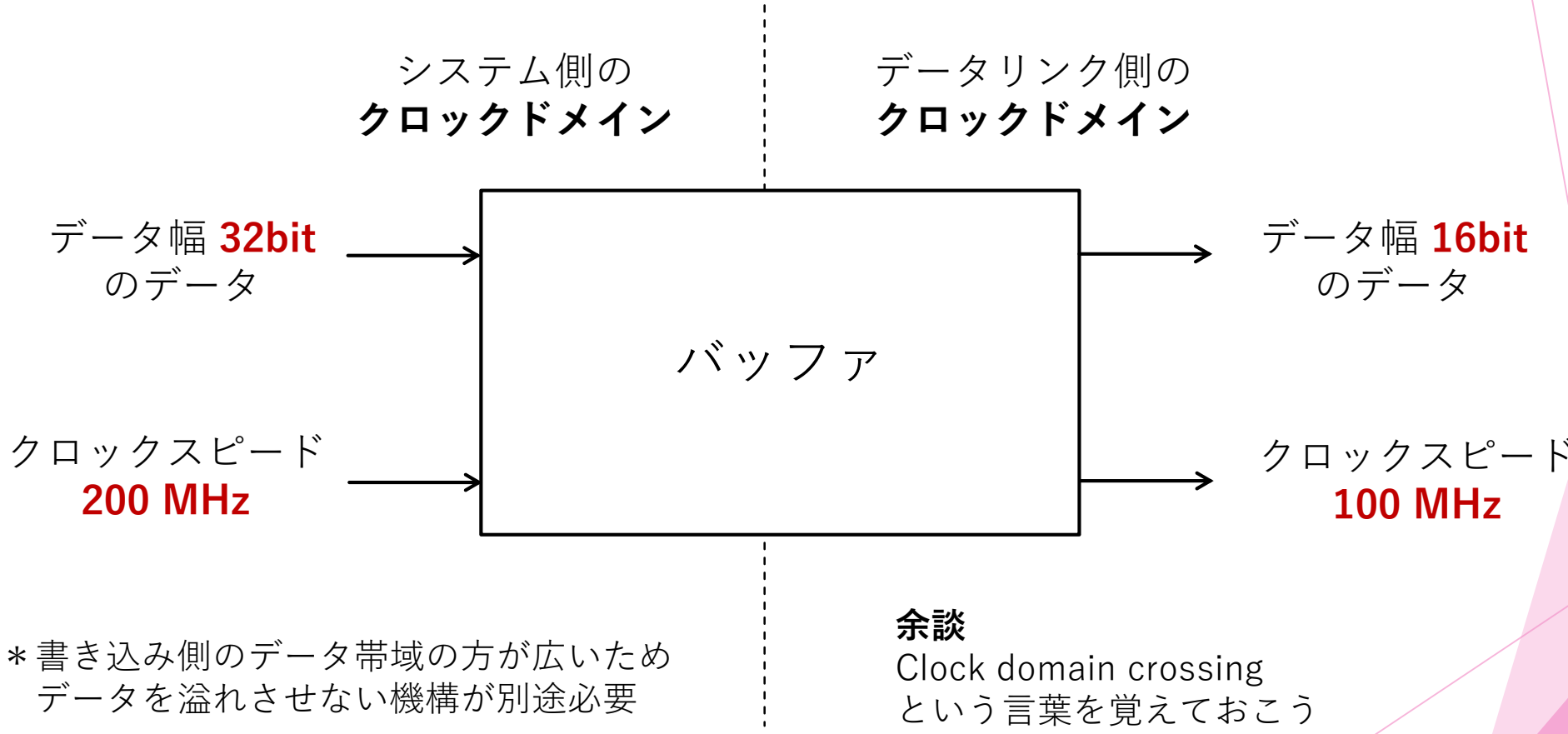
データを1時的に保存しておける機能

データバッファの役割は2つ
- データ転送速度の変換
- 待機機能



データバッファ

データの転送速度変換の例



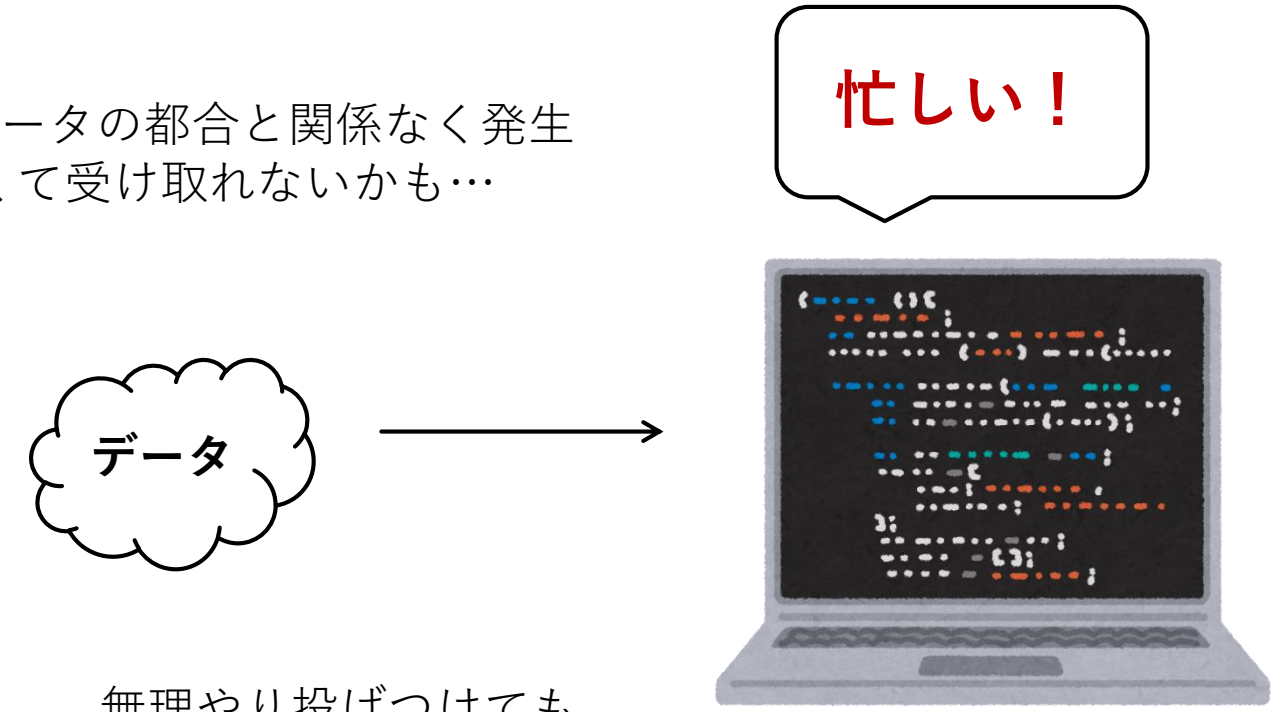
＊書き込み側のデータ帯域の方が広いため
データを溢れさせない機構が別途必要

余談
Clock domain crossing
という言葉を覚えておこう

データバッファ

待機機能の例

物理事象はコンパレータの都合と関係なく発生
たまたまPCが忙しくて受け取れないかも…

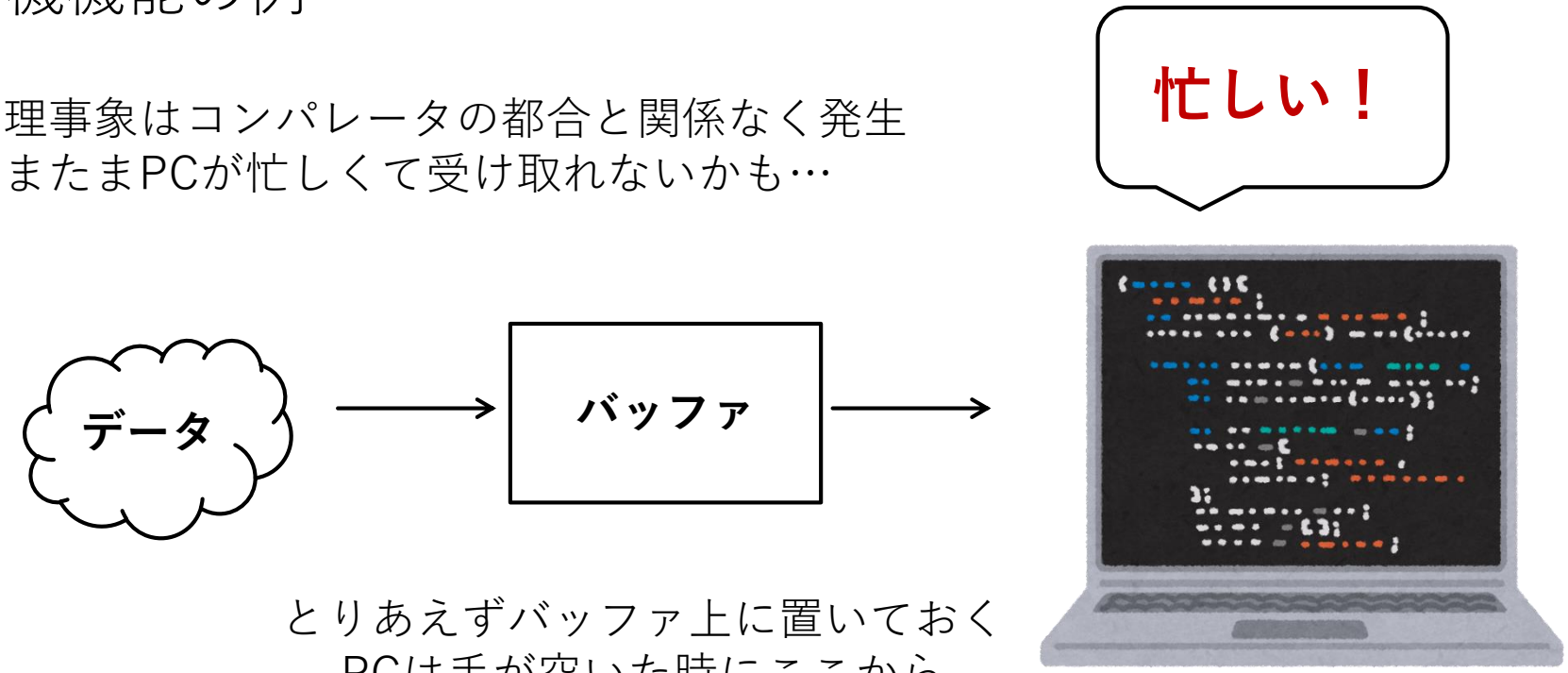


無理やり投げつけても
捨てられるだけ

データバッファ

待機機能の例

物理事象はコンパレータの都合と関係なく発生
たまたまPCが忙しくて受け取れないかも…

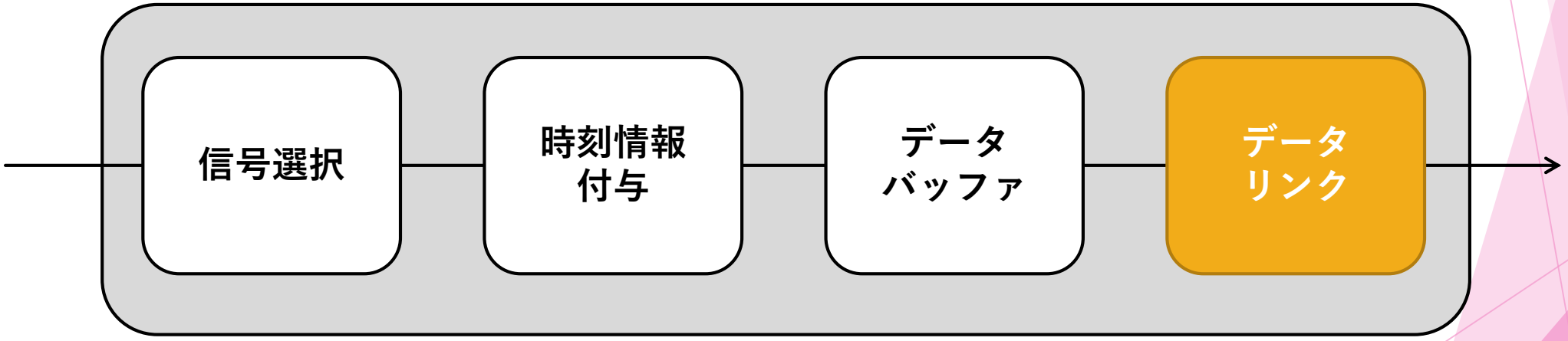


とりあえずバッファ上に置いておく
PCは手が空いた時にここから
読み出していく

データリンク

PCと話をするための機能
標準化されているI/Fを使用します

- Ethernet
- USB
- PCI express
- など



1.3　2進数と論理演算

2進数

1つの桁で表現できる状態は2つ (0および1)
10進数は1つの桁で10個表現できる (0-9)

0b = 2進数表記

10進数	2進数
0	0b0000
1	0b0001
2	0b0010
3	0b0011
4	0b0100
5	0b0101
6	0b0110
7	0b0111

10進数	2進数
8	0b1000
9	0b1001
10	0b1010
11	0b1011
12	0b1100
13	0b1101
14	0b1110
15	0b1111

16進数

10は2の整数冪でないので繰り上がりの周期が合わない
可読性を上げるために16進数を用いることが多い

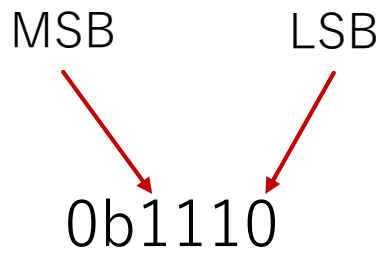
0x = 16進数表記

10進数	16進数	2進数
0	0x0	0b0000
1	0x1	0b0001
2	0x2	0b0010
3	0x3	0b0011
4	0x4	0b0100
5	0x5	0b0101
6	0x6	0b0110
7	0x7	0b0111

10進数	16進数	2進数
8	0x8	0b1000
9	0x9	0b1001
10	0xA	**0b1010**
11	0xB	0b1011
12	0xC	0b1100
13	0xD	0b1101
14	0xE	0b1110
15	0xF	0b1111
16	0x10	**0b10000**

用語

- **Byte (バイト)**
  - 8桁の2進数
- **Nibble (ニブル)**
  - 4桁の2進数 (あまり目にしない表現。16進数の1桁)
- **Least significant bit (LSB)**
  - 最低桁のビット
- **Most significant bit (MSB)**
  - 最高桁のビット



2進数の演算

AND
論理積

O = A & B
A = 1かつB = 1の時のみO = 1

OR
論理和

O = A | B
A = 1またはB = 1の時のみO = 1

NOT
論理否定

O = ~A
Aの反転

A, B, 及びOは1桁の2進数
複数桁の2進数同士の場合、各桁でそれぞれ演算
桁（ビット幅）が揃っていないと演算できない

2進数の加減算

加算

- 10進数と同じにように計算

```
    11 ←  繰り上がりの1
    0b0011
+   0b1011
-----------
   0b1110
```

減算

- このままでは計算できない場合がある

```
    0b0011
-   0b1011
-----------
       ?
```

符号が表現されていないので
答えがマイナスになる計算が出来ない

符号付き2進数表現

通常**2の補数**を使う
- ある2進数値をNOT演算し1を加える

10進数の7までを5ビット幅で2の補数表現した場合

10進数	2進数	2の補数
0	0b00000	
1	0b00001	0b11111
2	0b00010	0b11110
3	0b00011	0b11101
4	0b00100	0b11100
5	0b00101	0b11011
6	0b00110	0b11010
7	0b00111	0b11001

MSBが符号ビット
- MSB = 1の時は負の数を表す

2の補数を使った減算処理

2の補数の利点
- 減算を2つの値を加算することで実行できる
- ハードウェアの視点から見ると加算器のみで加減算できるようになり減算器を実装しなくてよくなる。

例題：7 – 5を2の補数を使い計算する
- 7および-5を4ビット幅で表現する
  - ＋7：0b0111
  - ー5：0b1011
- 加算を実行する
  - 0b0111 + 0b1011 = 0b10010
- 4ビット幅で定義したので下位4ビットを抽出
  - 答え：+2

四則演算を加算器で実装

乗算
- 複数回の加算へ変換する
- A×BはAにAをB-1回足す

除算
- 複数回の減算に変換する
  - A÷Bは2の補数を用いて負になるまでAからBを引く

加算器さえ実装すれば四則演算を行う回路が実現できる
（ただし乗除算は計算時間がかかるし、可変）

乗算除算回路は色々な実装方法があるので調べてみてください

履歴

資料原案

- 2018/7/27 第3.0版 内田智久 (Esys, KEK)

- 2020/7/29 v1.0 本多良太郎 (東北大)

2．デジタル回路

2020年7月29日
本多良太郎（東北大学）

デジタル回路構成要素

実際の回路を使って
デジタル回路の構成要素を見てみる

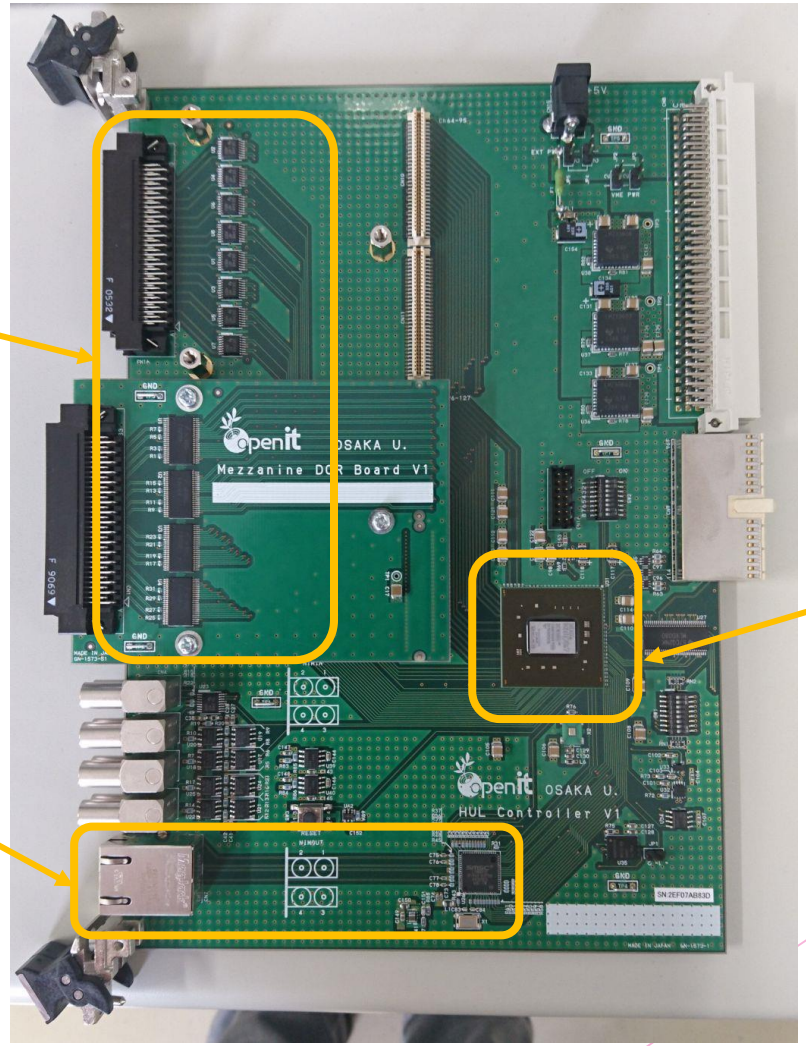
信号処理・IO標準規格
基板上を走る信号には規格がある

目的に合わせて信号規格を選ぶ

データ通信
- 装置内通信
- 装置間通信

Ethernet通信でPCと通信

汎用デジタル回路
Hadron universal logic module



デジタル信号処理
- ゲート素子
- フリップフロップ
- クロック同期
- パイプライン
- メモリ
- ステートマシン

FPGAがデータ処理の中心

2.1　信号処理

IO標準規格 (IO standard)

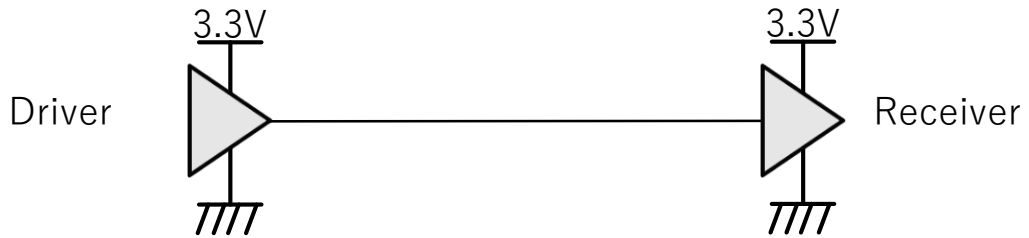
チップ間通信のために様々な規格が制定されている

回路設計をするうえでは
- 速度, 消費電力, 伝達距離などに応じて適切な規格を選ぶ
- 異なった規格同士では多くの場合で正しく信号の授受が出来ない
- IO standardが異なると電源電位も異なることが多い

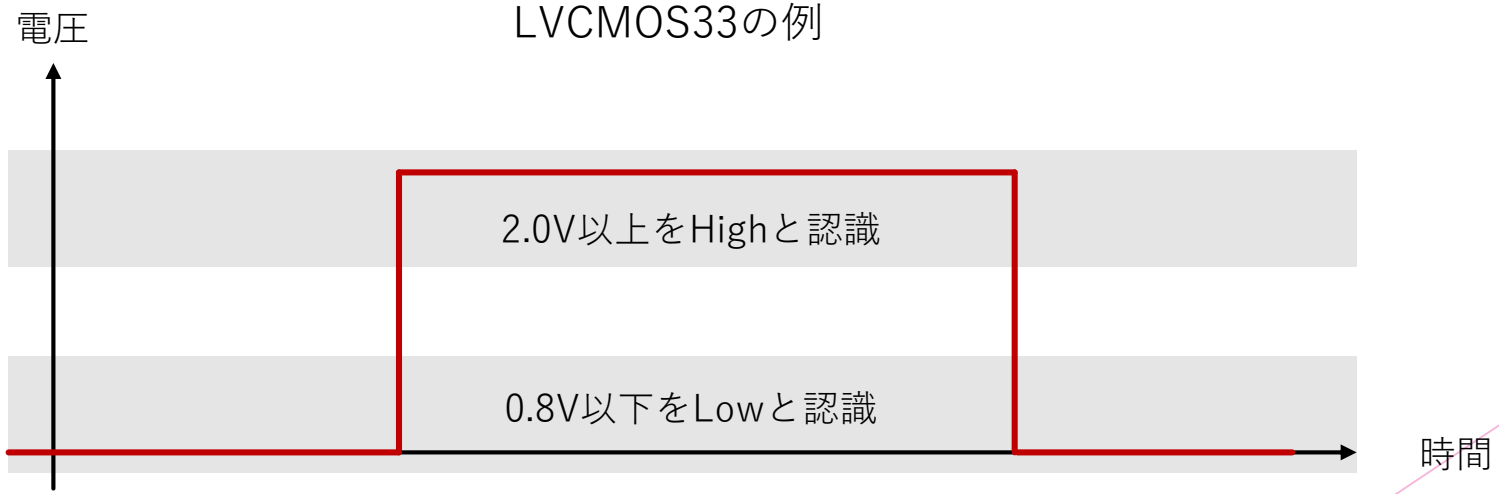
今回紹介するのは
- LVCMOS: 単一端 (single-ended) 信号
- LVDS: 差動信号 (differential signal)

LVCMOS

Low Voltage CMOS: CMOS技術によるデジタル回路の低電圧規格
- 3.3V, 2.5V, 1.8V…など複数の電圧ごとに規格が存在する
- 使う際には電源電圧も確認



LVCMOS33の例



余談
- TTL (transistor-transfer logic)は5V
- LVTTLは3.3V

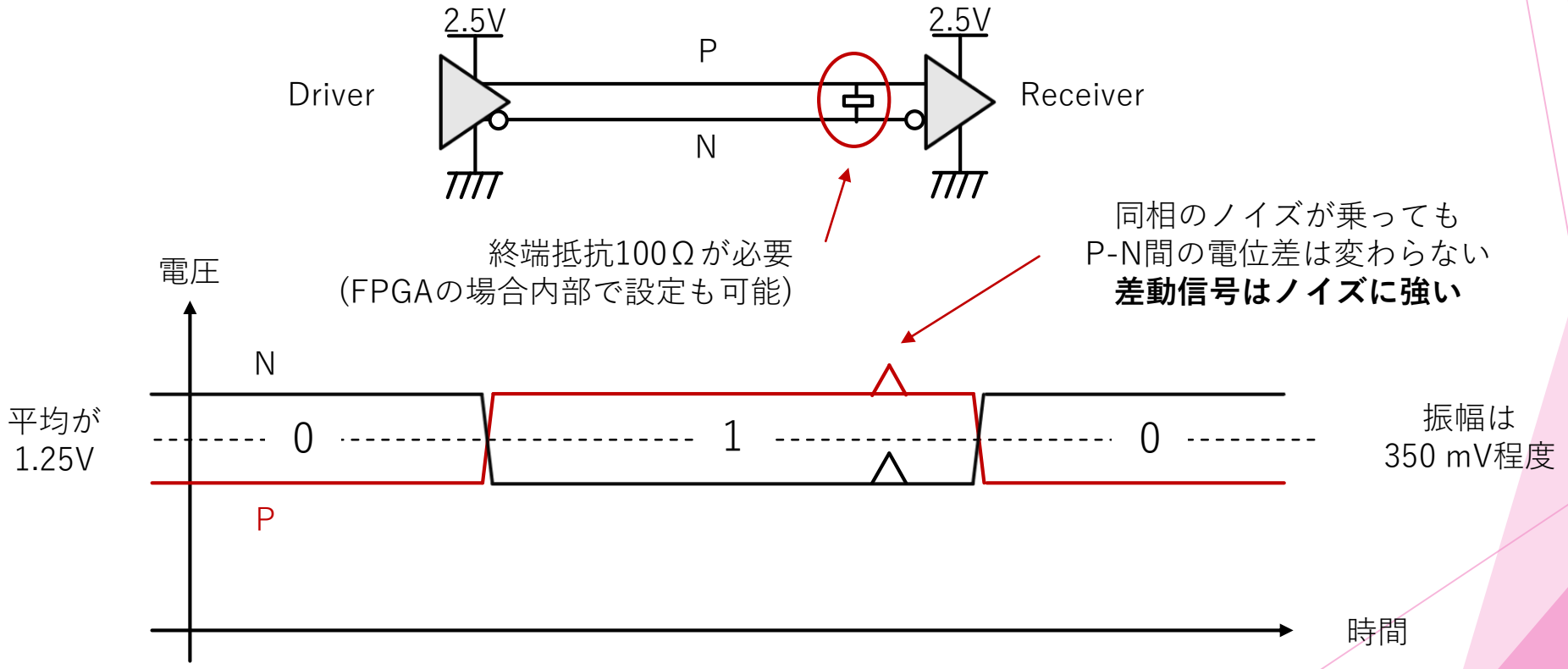
TTLは本来バイポーラトランジスタで構成した回路を指す

LVDS

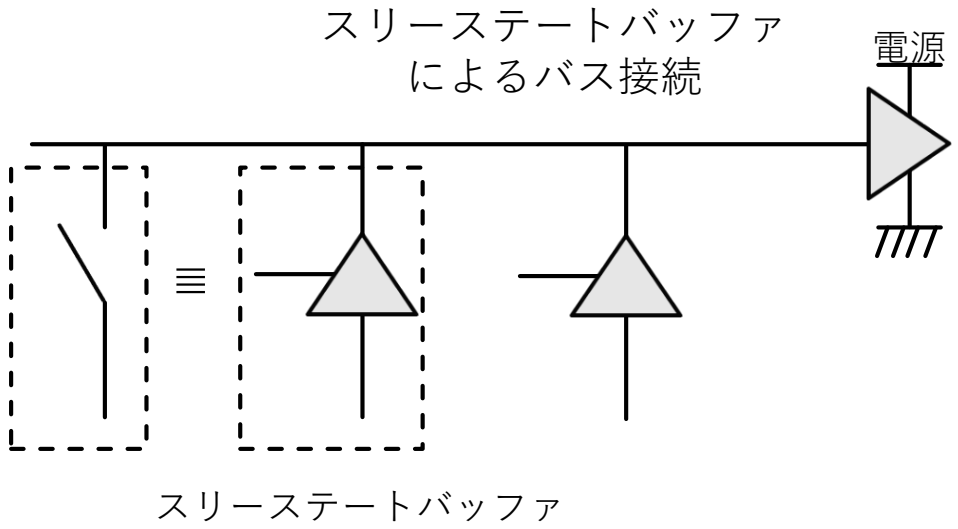
Low Voltage Differential Signal: 現代の差動信号の主流

差動信号

- 2本の信号 (P-N) 間の正負で0と1を判別する



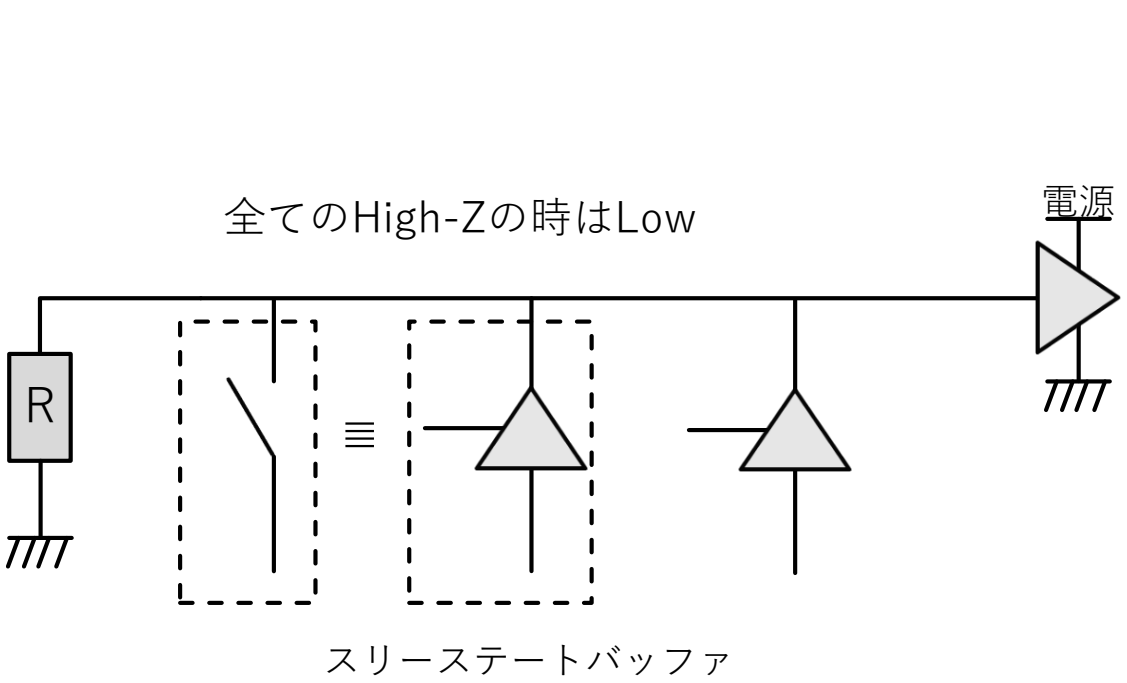
ハイ・インピーダンス状態

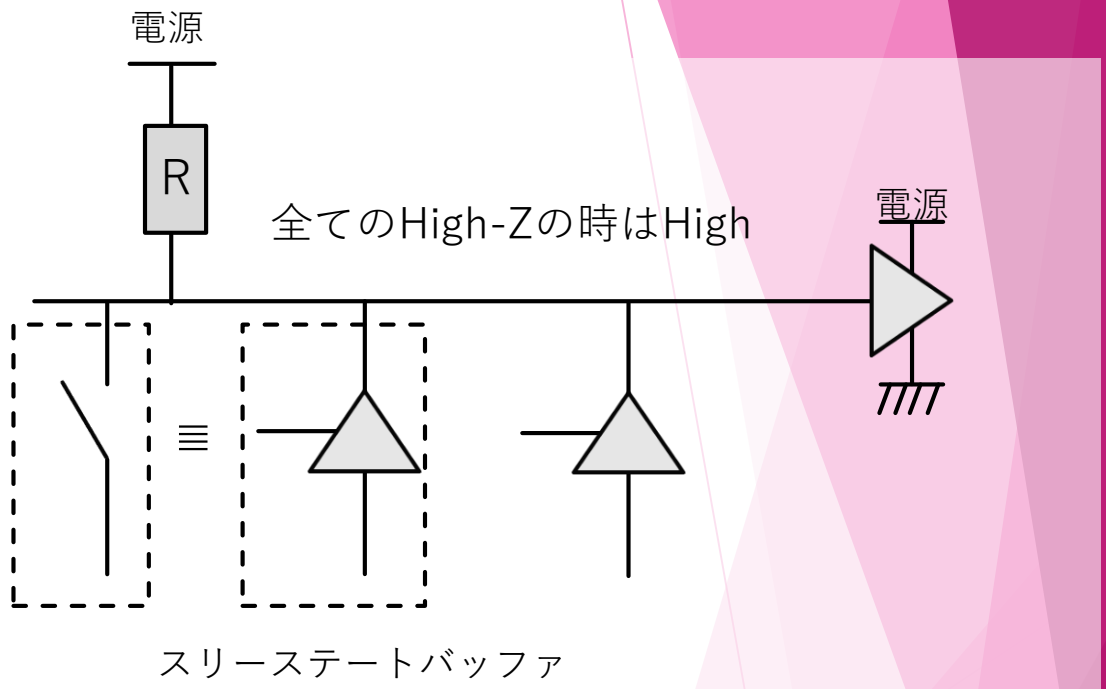


状態	
High	接続.
Low	接続.
Z (High-Z)	断線. ハイ・インピーダンス. 第3の状態

全てのバッファがHigh-Z状態だと
信号線の状態が不定になってしまう
どうしたらいいか？

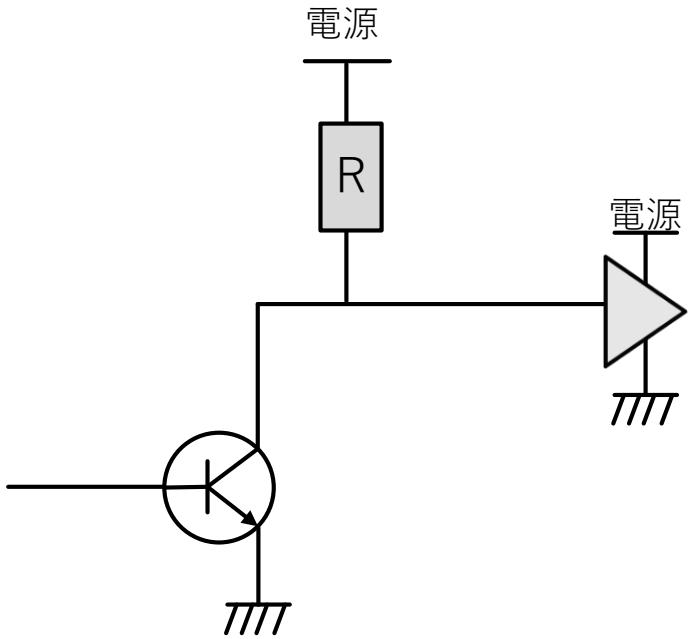
プルアップ・プルダウン





プルダウン抵抗・プルアップ抵抗により
High-Zの時の状態を定める
バッファ接続時はそちらが優先

オープンコレクタ



トランジスタがON
- 信号線はLow

トランジスタがOFF
- 信号線はHigh-Z：プルアップ電位で上書き

信号のレベルをレシーバ電源に合わせることが出来る
→**レベル変換**

MOS FETの場合オープンドレインになります
調べてみてください

2.2　組み合わせ回路

組み合わせ回路

記憶素子を使わない回路

- 基本ゲート (AND, OR, NOT) のみで構成
- **入力と出力の関係が一意に決まる**
- 入出力対応を真理値表で書くことが出来る

用途

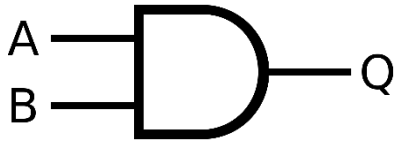
希望する状態を選択する

例：検出器のコインシデンスを取る

おさらい

AND
論理積

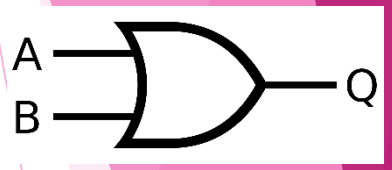
O = A & B
A = 1かつB = 1の時のみO = 1



MIL論理記号

OR
論理和

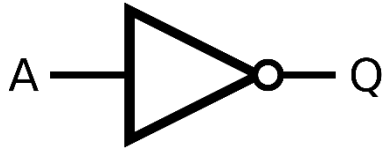
O = A | B
A = 1またはB = 1の時のみO = 1



MIL論理記号

NOT
論理否定

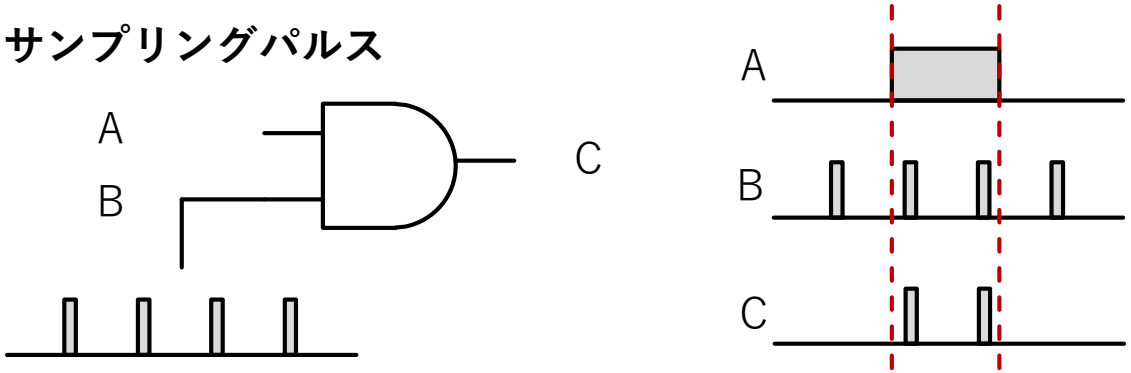
O = ~A
Aの反転



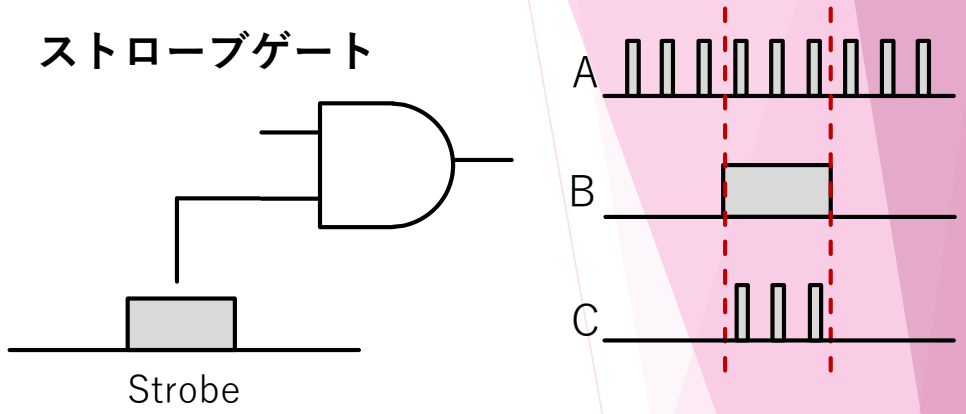
MIL論理記号

ANDゲートで用語の説明

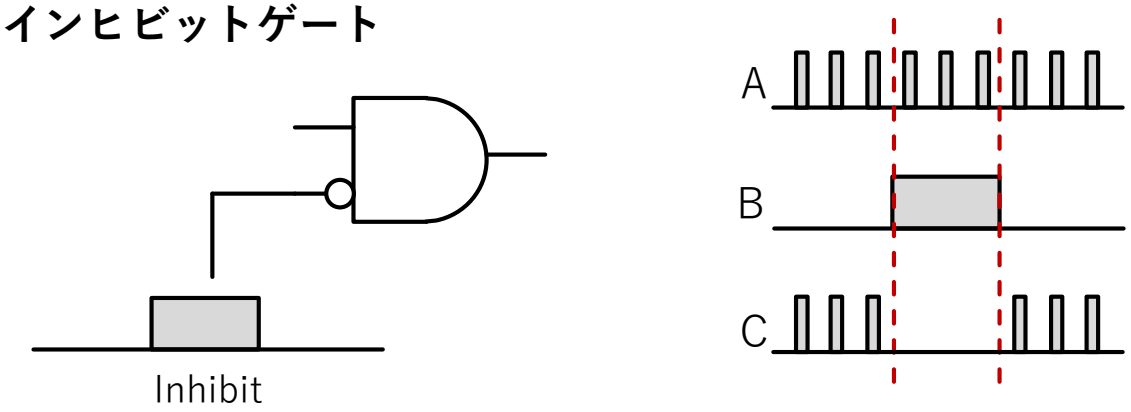
サンプリングパルス



ストローブゲート



インヒビットゲート



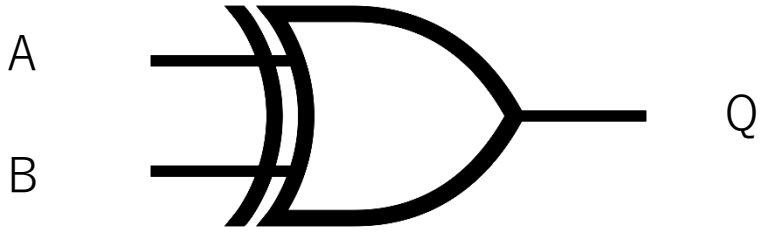
サンプリング
- ある時刻の入力のHigh/Lowを調べる

ストローブ（ストロボ）
- 信号を取り込むタイミングを知らせる

インヒビット
- 信号を受け付けられない状態を示す

排他的論理和 (XOR)

AとBの状態が異なるとQが1



XORの真理値表

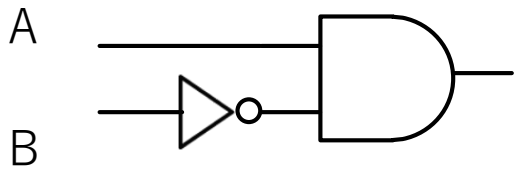
A	B	Q
L	L	L
H	L	H
L	H	H
H	H	L

H: High
L: Low

基本ゲートだけで作ると？

排他的論理和 (XOR)

片方がHだとQがHになる点に注目



右の赤枠の状態が再現できた
逆側も同様に作れるはず

XORの真理値表

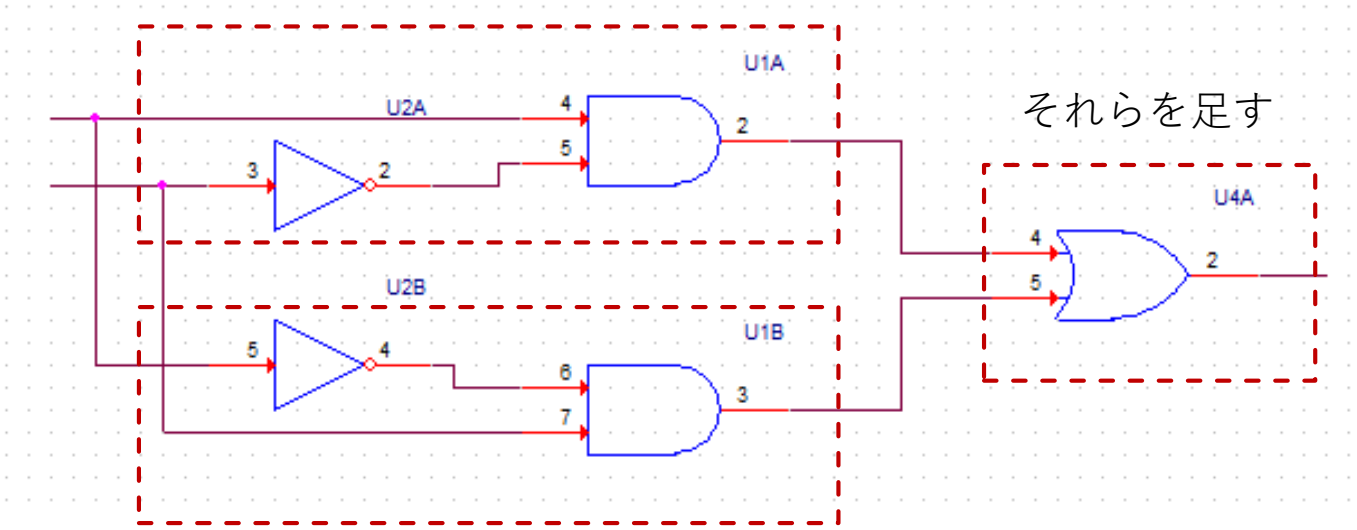
A	B	Q
L	L	L
H	L	H
L	H	H
H	H	L

H: High
L: Low

排他的論理和 (XOR)

A

B



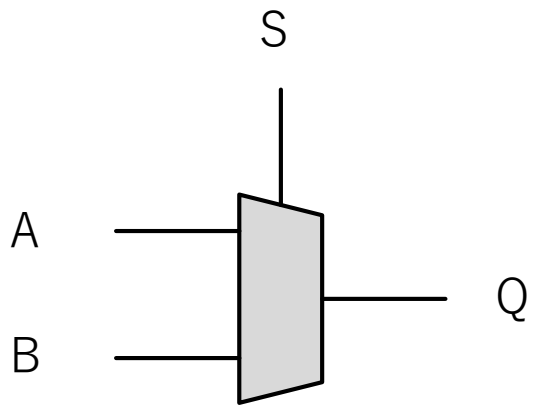
Q

特定の状態を再現する構成ブロック

組み合わせ回路の設計方針

1. 真理値表を書く
2. 真理値表の特定の状態が再現できる小さい回路を作る
3. それらを足す

1bit 2入力 Multiplexer/Selector



基本ゲートだけで作ると？
（考えてみよう）

2入力multiplexerの真理値表

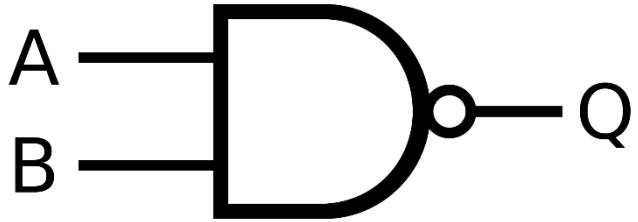
A	B	S	Q
L	X	L	L
H	X	L	H
X	L	H	L
X	H	H	H

XはDon’t care
入力がどんな値であれ
出力に影響を与えない

条件によって経路選択ができるようになった

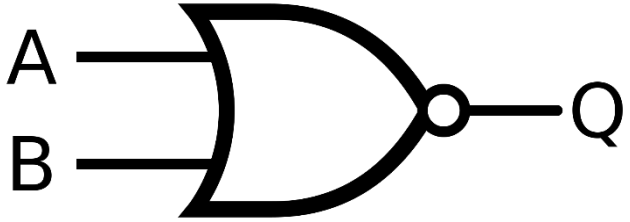
NANDとNOR

記憶素子で使うので紹介しておきます



NANDの真理値表

A	B	Q
L	L	H
H	L	H
L	H	H
H	H	L



NORの真理値表

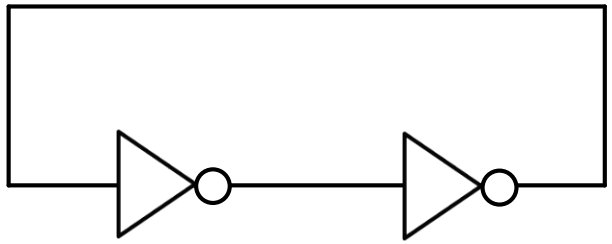
A	B	Q
L	L	H
H	L	L
L	H	L
H	H	L

2.3　記憶素子

記憶原理

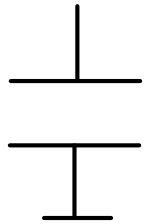
典型的な記憶原理

手法A



自己フィードバックループを作って
値を保持
こちらを見ていく

手法B

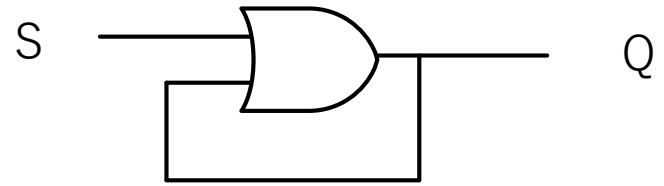


Vss (GND)

コンデンサに電荷をためる。
リーク電流で電荷が逃げるので
定期的な値の更新（リフレッシュ）が必要

ラッチ回路を作ってみる

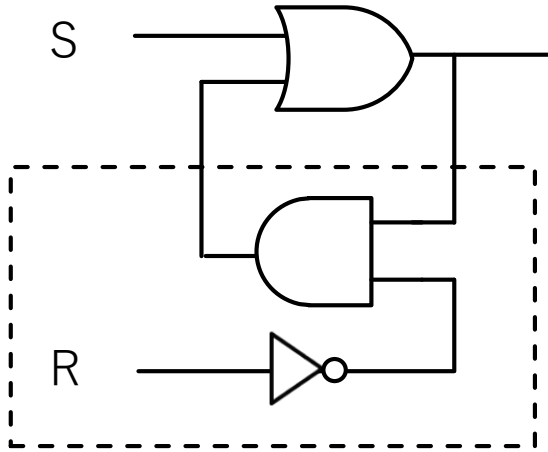
ORゲートを使って自己フィードバックさせる



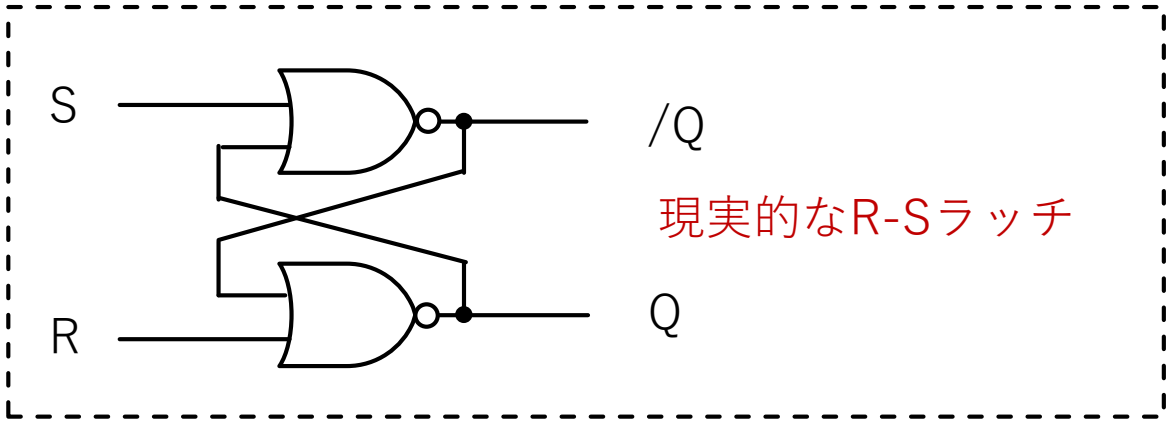
値は保持できるが二度と消去できない

ラッチ回路を作ってみる

値を消去できるようにする



R-Sラッチ:
セット (S), リセット (R) レベルにより動作するスイッチ



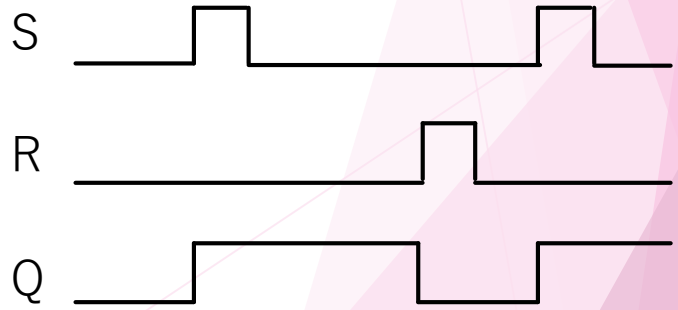
現実的なR-Sラッチ

R-Sラッチの真理値表

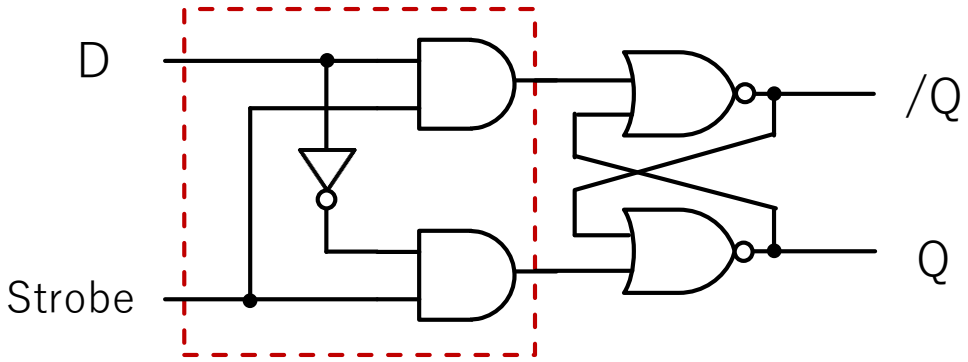
S	R	Q
L	L	Q_n
H	L	H
L	H	L
H	H	不定

Q_nは直前の値を保持
S/R両方HだとR-Sラッチの動作は不定

タイミングチャート
(伝搬遅延は無視)



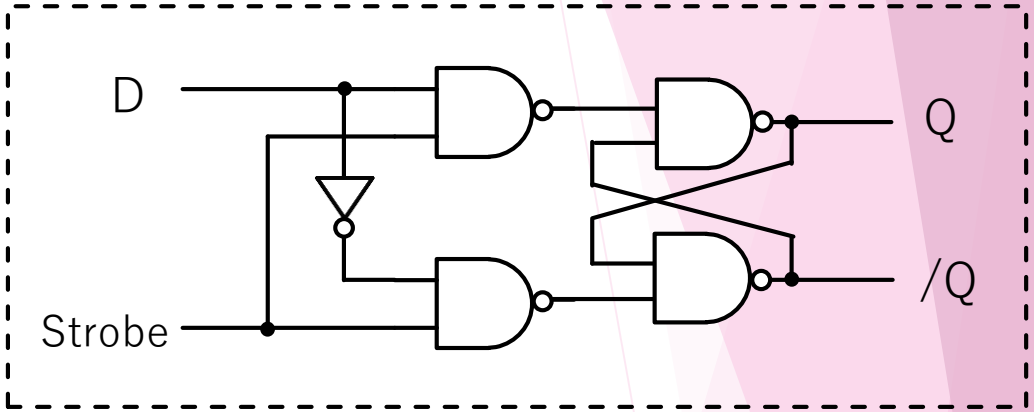
データを取り込むラッチ（Dラッチ）



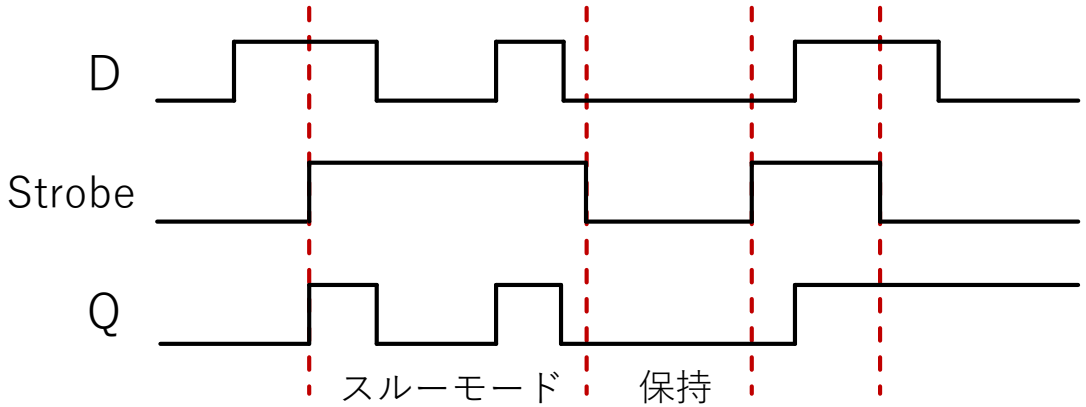
R-Sラッチの前に追加で
回路を取り付け

等価

どうして等価か
考えてみよう



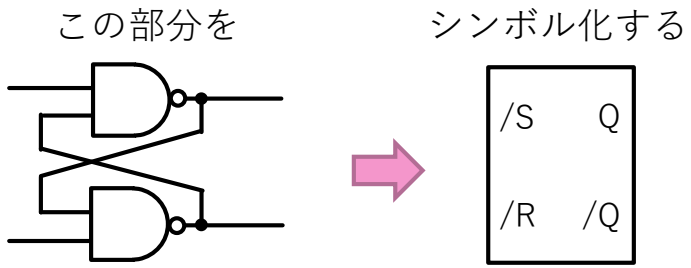
現実的なDラッチ
（Qの論理が反転してることに注意）

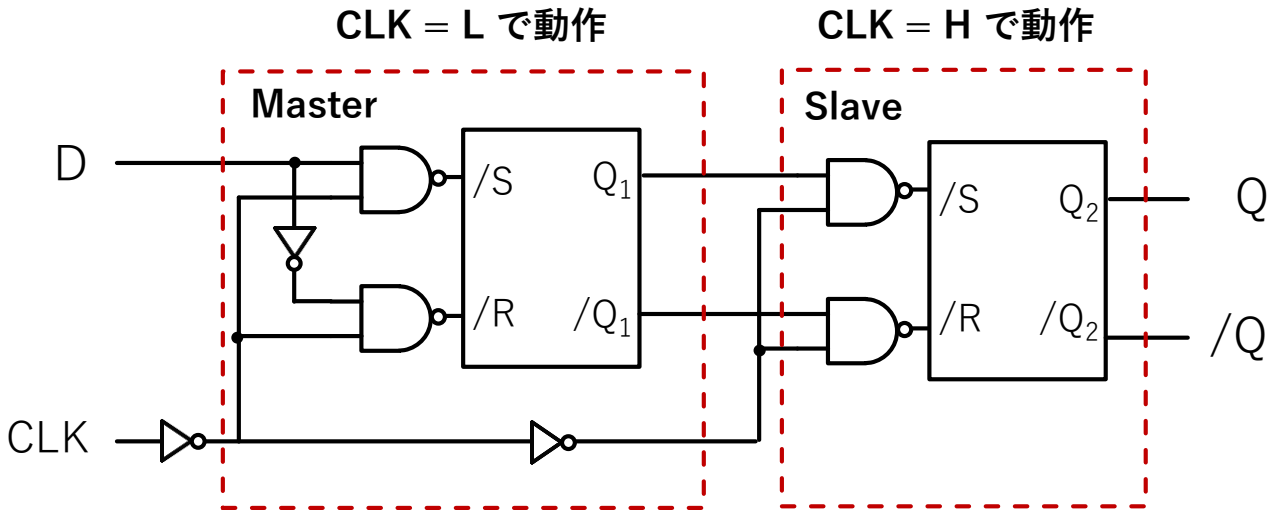


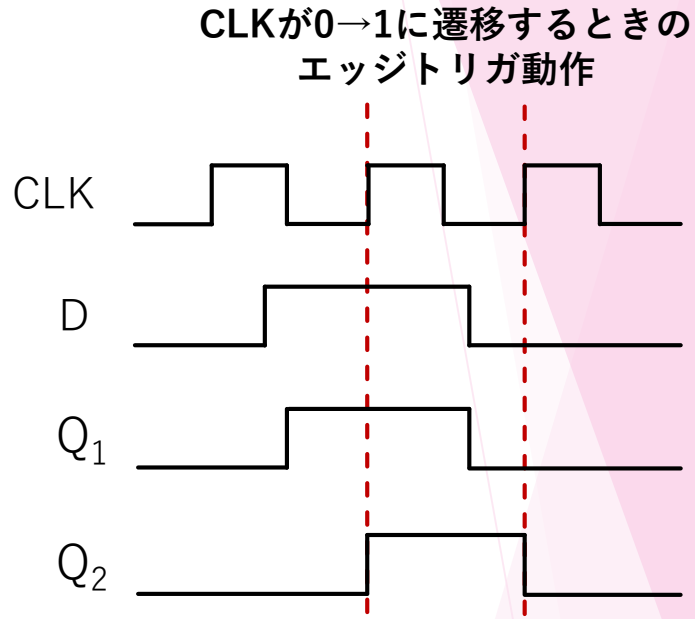
StrobeがHigh間データを取り込み
StrobeをLowにするとデータを保持できるようになった

ところが…
StrobeがHighの間にDが変わるとQも追随する
レベル動作
Strobeが遷移するタイミングの時だけ値を調べるようにできないか
エッジ・トリガ動作

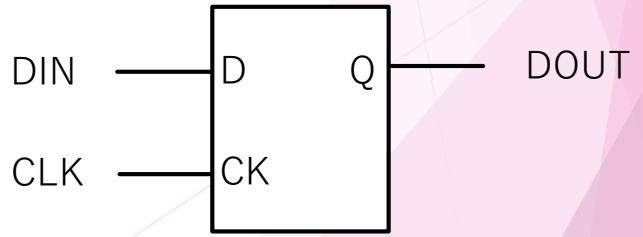
Master-Slave型フリップフロップ







回路シンボル
D-flipflop (D-FF)



余談: D-FFのDはデータとディレイのD

2.4　順序回路

内容

カウンター回路 (教科書で最初に出てくることが多い順序回路)

- 非同期カウンター
- 同期カウンター

上記2つの違いを利用して、デジタル回路の**タイミング設計**について学ぶ
クロック同期とタイミングはデジタル回路で極めて重要な部分！

順序回路

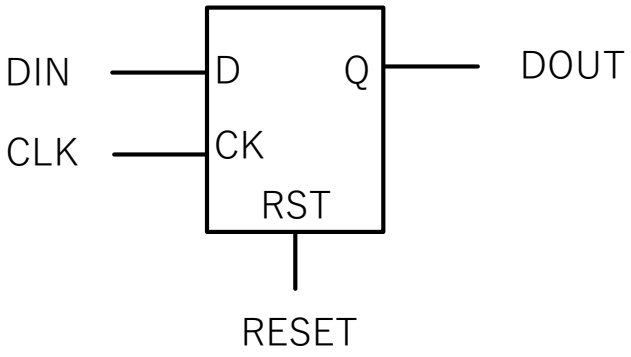
記憶素子を使う回路
- この講義の中では記憶素子：D-FF (Dフリップフロップ)
- **過去の情報を保持**することが出来る

過去の情報に依存して出力結果が変化する
- 例1：スイッチでLEDの点灯・消灯
- 例2：数を数える

復習：組み合わせ回路は入力から出力が一意に決まった

非同期・同期カウンタともにD-FFで構成される
まずはD-FFの動作を見てみよう

よくあるD-FFの動作



非同期リセットピンがついた
クロックに依存せずに
Qの値を強制的に0にするためにつかう

- 無くてもよい
- RST=0でリセットになる回路もある

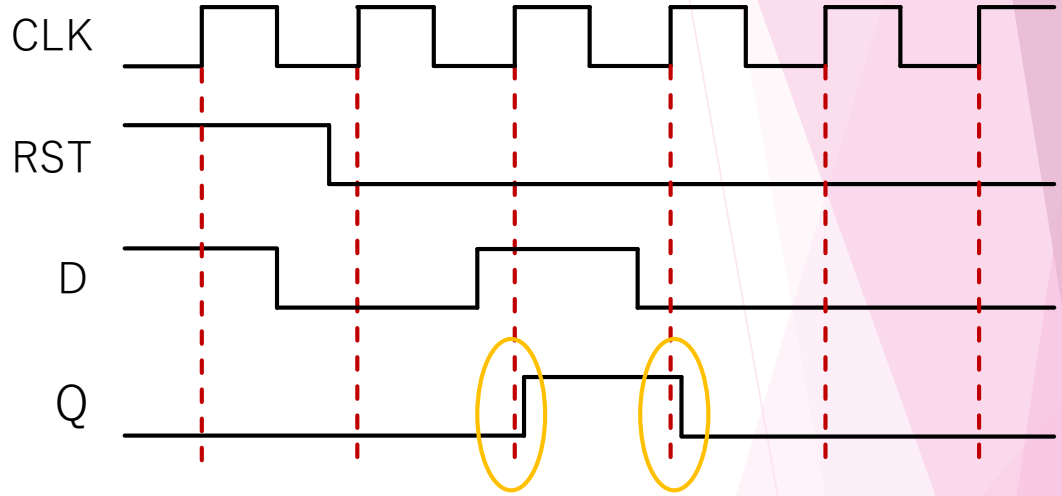
余談

- FPGAで非同期リセットを使うのはクリティカルな場所だけにするとよい。
- FPGAの配線リソースは限られているので必要ないFFにまでリセットを配ると配置配線で苦しむことがある。

ClockはCLKかCK
ResetはRSTと略することが多い

RST=1でQは0

DINはCLKに
同期してないとする



クロックエッジから**少し遅延してQが遷移**している。
クロックエッジから出力の状態が安定するまでの
時間を**clock-to-output delayと呼びTco**と書く。

D-FFの出力は必ず少し遅れて確定し
Tcoは素子に依存する！

カウンタ回路とは？

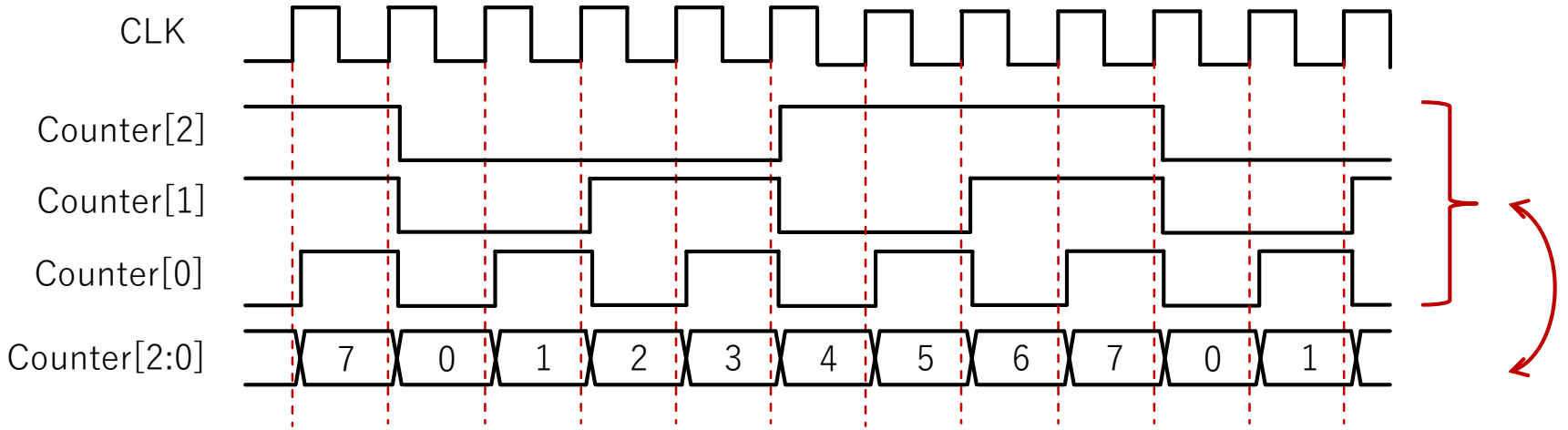
数を数える回路

クロックティック毎にカウンタが1増える
- 時間を測る
- クロックを分周する

EN (Enable) を付けてEN=1の場合だけ数を数える
- イベント数を数えるなど

2つの異なった実装方法でカウンタを作る
- 非同期カウンタ
- 同期カウンタ

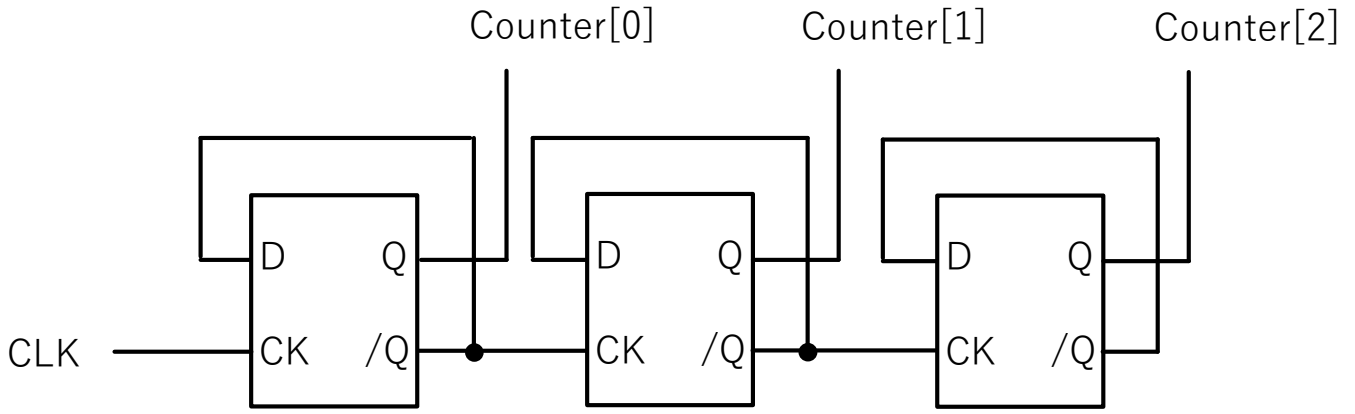
タイミングチャートにどういう違いが出るか？



同じ情報を示している
下のようにまとめ書く

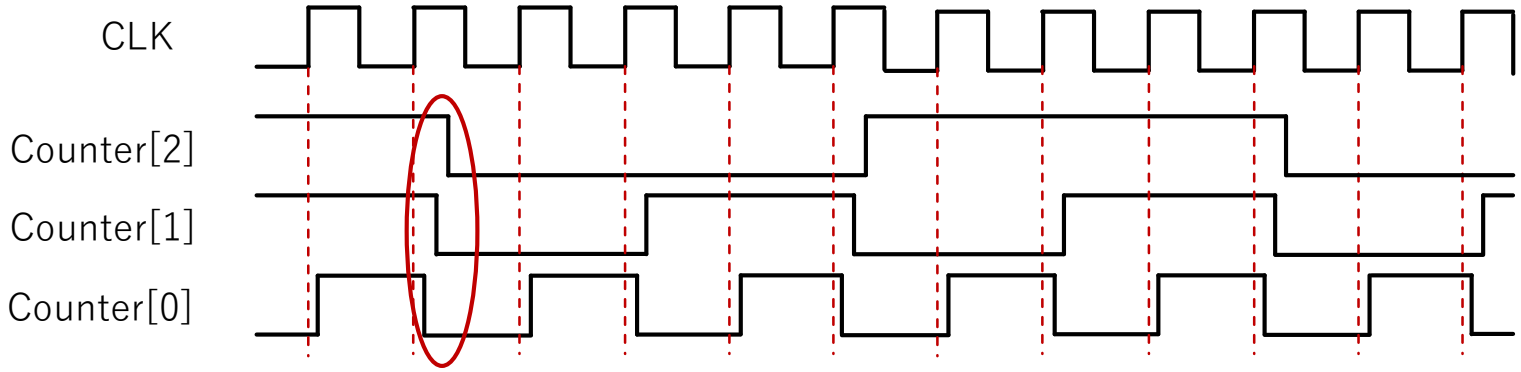
このタイミングチャートは同期カウンタ

非同期カウンタ



特徴

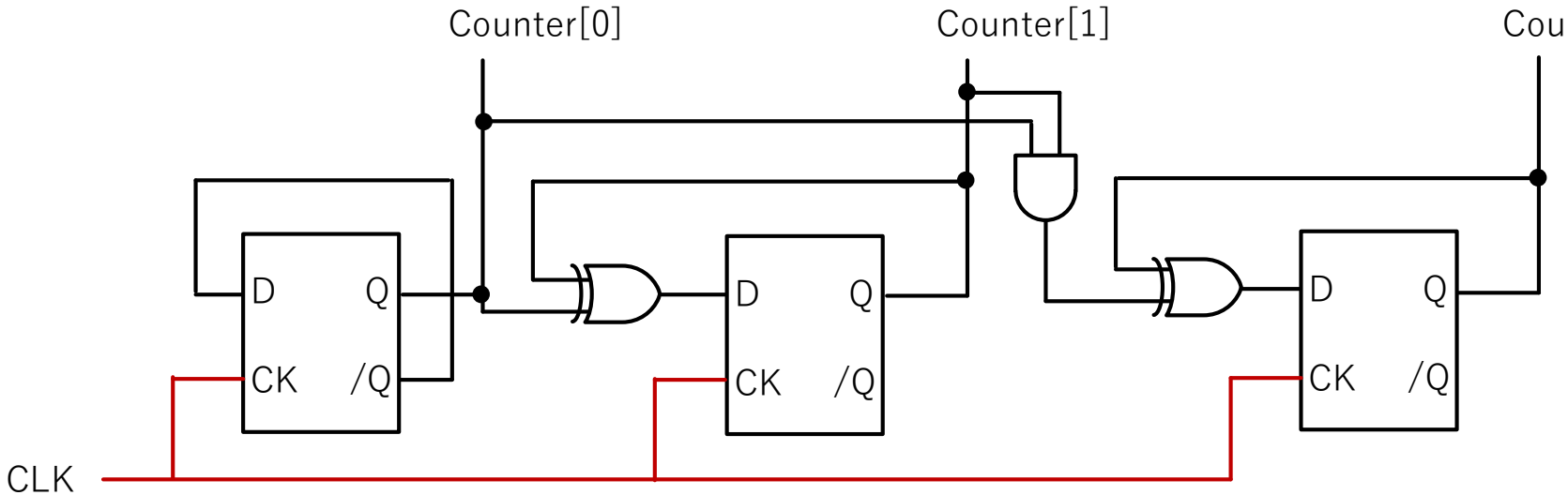
- 同じ構造のブロックをつなげていくだけでビット数が増える
- 構造が単純
- しかし最近の回路ではあまり使わない



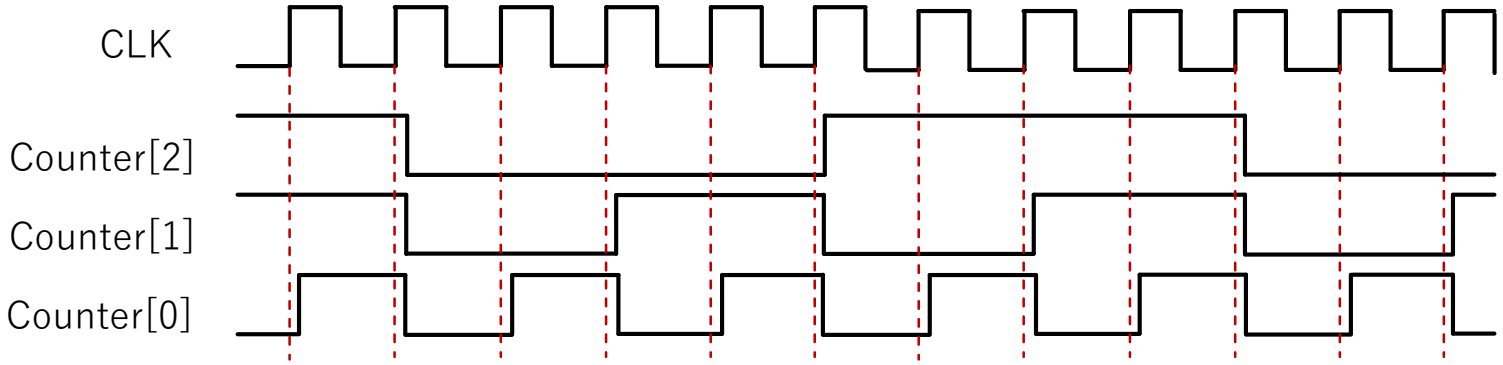
単一のCLKで同期されていないのでこのような回路を
非同期回路
と呼ぶ

次のFFのCKが前のFFの結果で駆動されるので遅延が積みあがっていく

同期カウンタ



経路長による**到達時間の差（クロック スキュー）**が
なるべく要求の範囲内に収まるように設計する



特徴

- 非同期カウンタより構造が複雑
- デジタル回路の主流
- FF駆動は同じクロック
- 遅延量が一定

単一クロックで同期されているため
同期回路
と呼ぶ

非同期回路はなぜ流行らなくなったか？

タイミング設計が非常に煩雑

- **遅延が積みあがる**ので入力から何秒後に結果が確定するかケースバイケースに考えないといけない。
  - 例：非同期カウンタの**ビット数が増えれば遅延量**も増える。
- FFの出力遅延だけでなく組み合わせ回路を通過する遅延量も加算される。
- 配線遅延も更に追加される。
- 素子の特性により遅延量は変化する。
  - 例：温度特性

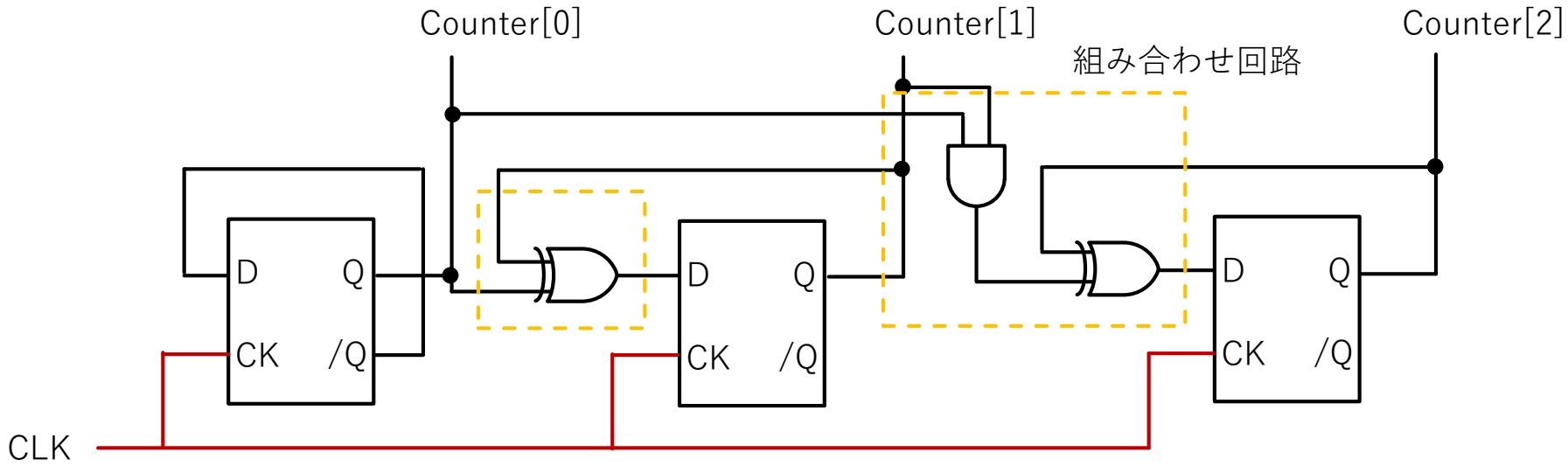
これらすべて考慮して回路全体にわたって**遅延量の足し上げを行うのは大変**。

- 非常にゆっくり動作させることが決まっている回路は別だが。

問題をシンプルにしたい！

同期回路

同期回路は何がいいか



全FFがスキュー調整されたクロックで動作する

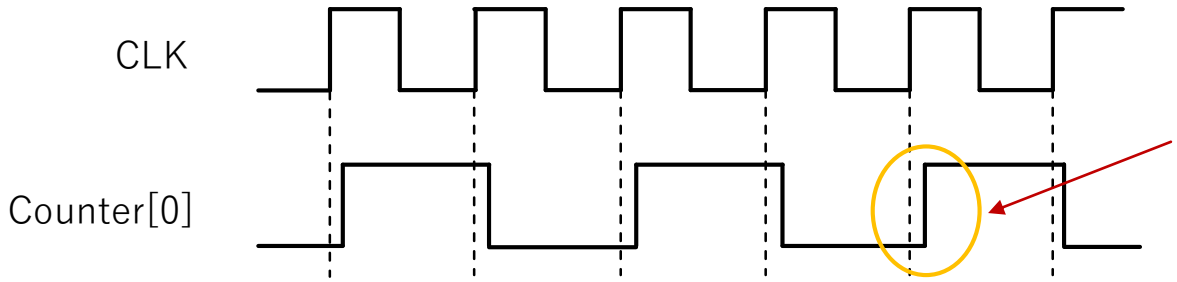
- 同時刻性
- 周期性

あるFFから出力された情報は次のクロックエッジまでに次のFFへ到達すればいい。

- 配線遅延
- 組み合わせ回路の遅延

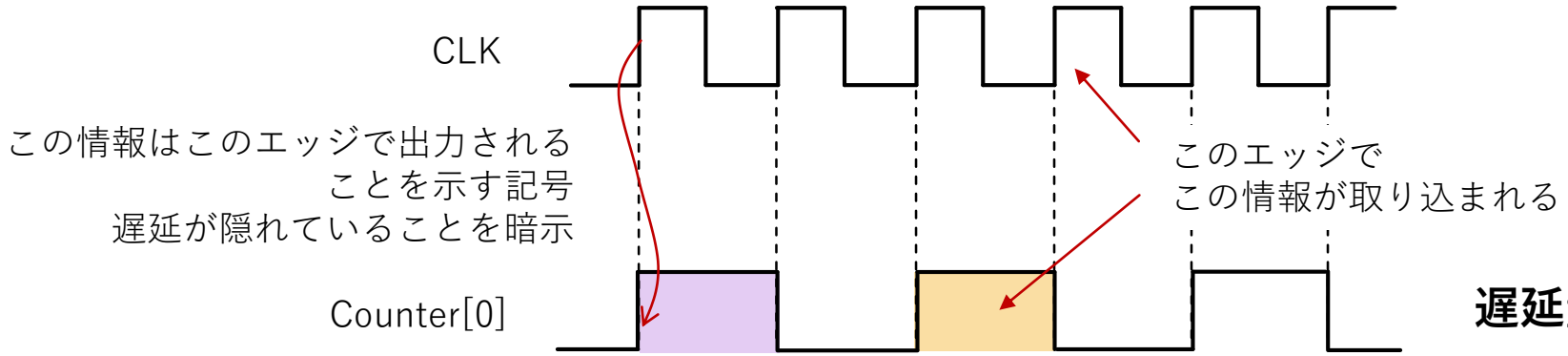
問題がシンプルになった！

同期回路のタイミングチャート



遅延量をあらわに書いていた

同期回路ではFFの出力遅延が同一なので
気にしなくていい



遅延量を書かないことが多い

同期回路のまとめ

- 現在は同期回路での設計が一般的です
- 特にFPGAは同期回路で設計する事を前提に設計されています
  - 同期回路を上手に設計する事でFPGAの性能を十分に発揮させることが出来る
- 非同期回路設計は一般に難しいです
- 慣れるまでは同期設計を採用してください

2.5　同期回路のタイミング

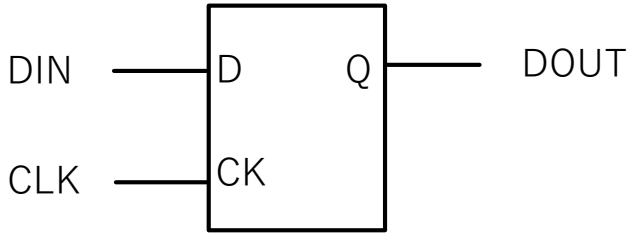
DFFの特性とタイミング

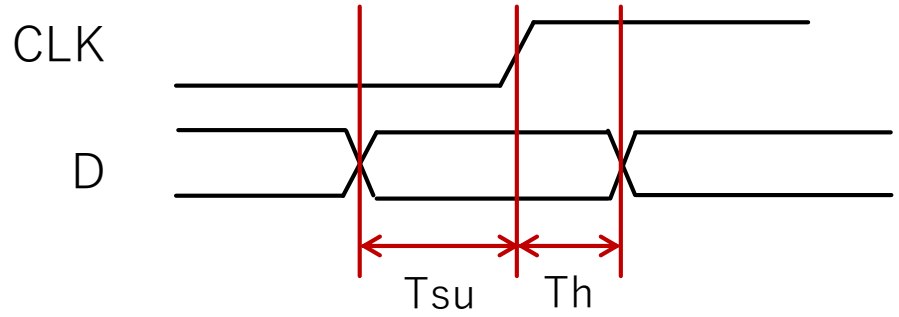
同期回路ではクロック周期内に情報が伝達できればいいと述べた
しかし1点重要なポイントを無視している

DFFは自己フィードバックループで情報を保持するため、
フィードバックループが安定する前に入力が変化すると
動作がおかしくなる

ここでは何に気を付けないといけないかを見ていく

セットアップ・ホールドタイミング

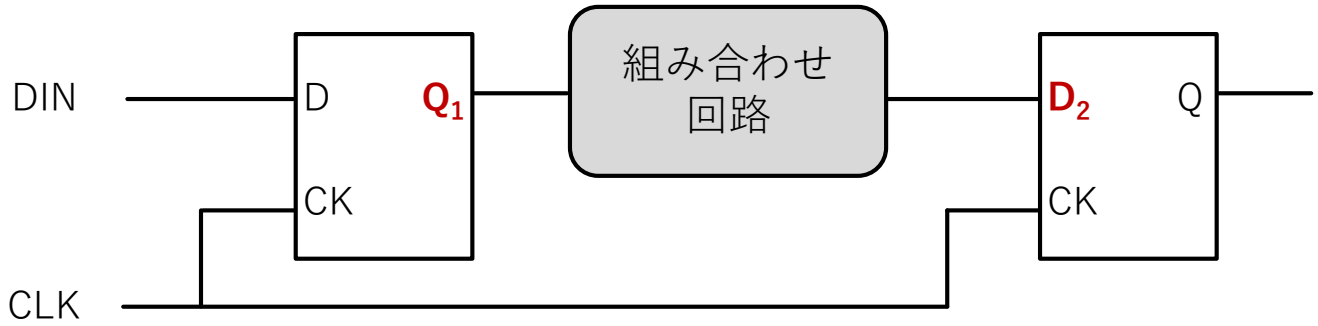


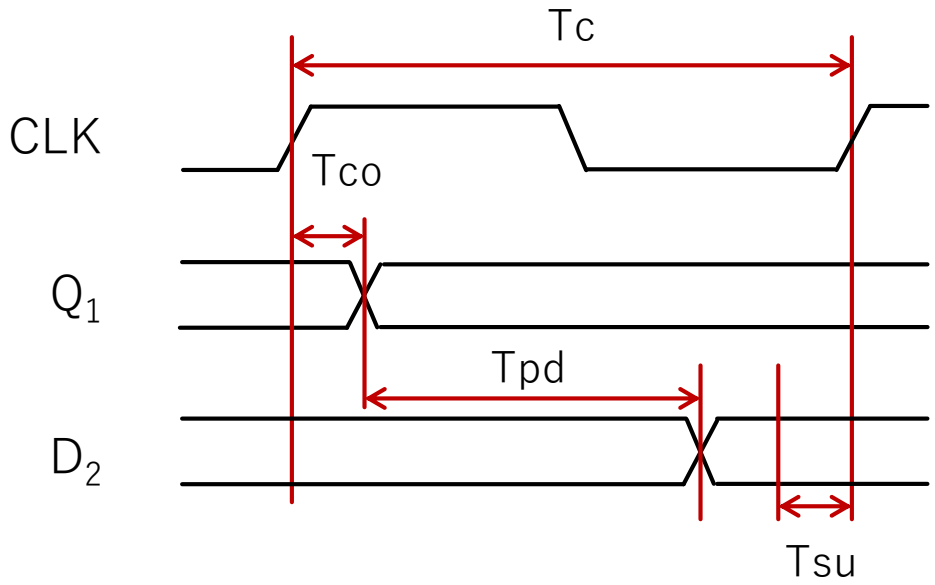


Tsu: セットアップ時間
Th: ホールド時間

正しい値をDFFへ保持させるためには
入力DはTsuとThの間の時間安定してなければならない（変化してはいけない）
TsuとThはプロセスによって決まっている

同期回路のタイミングを満たす条件





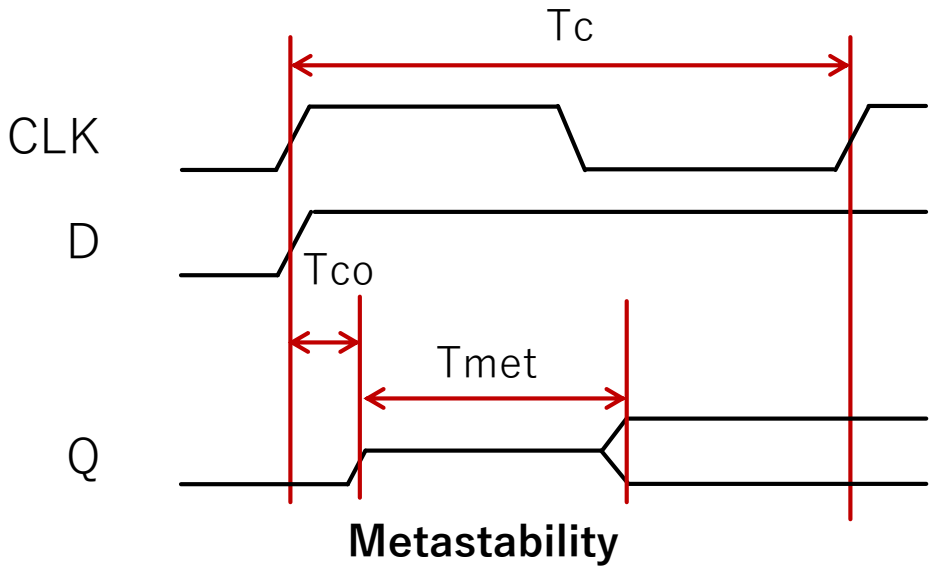
Tc: クロック周期

Tco: Clock-to-output delay
Tpd: Propagation delay
Tsu: セットアップ時間

$$Tc > Tco + Tpd + Tsu$$

DFFへの非同期入力

検出器信号などはクロックに同期してないのでランダムにやってくる
TsuとThの間にDが遷移してしまうとどうなるか？
答え：どうなるか分からない
Qが遷移しないかもしれないし、正常に動くかもしれないし、
中間電位で止まるかも、発振するかも



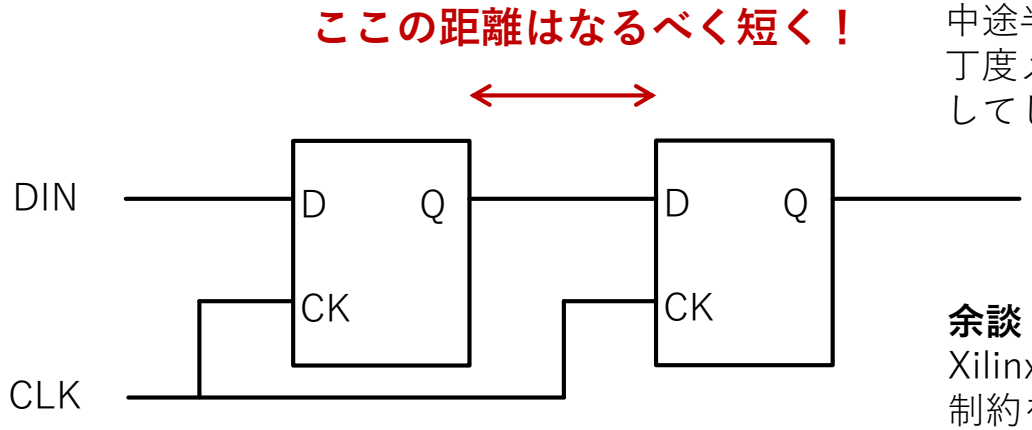
Tcoたった後も安定な状態にならず
安定状態に戻るまでに追加の遅延が発生する
- Tmet: セットリング時間 (落ち着くまでの時間)

安定状態とは言ってもHighに落ち着くかLowに落ち着くか分からない。
Tmetはプロセス技術や周辺環境に依存する。

メタスタビリティへの対処

そうは言っても検出器の信号は入れないといけない
対処方法：入力FFを二重化 (三重化)する

最初のFFでメタスタビリティが発生しても
セットリング時間後に次のFFでサンプリングしてしまえばよい



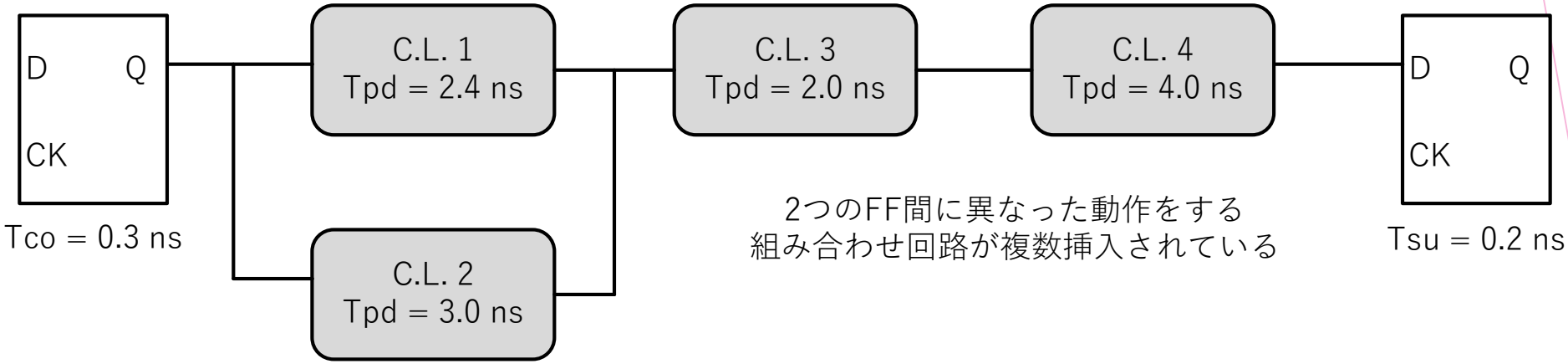
中途半端に配線遅延があると
丁度メタステイブル状態をサンプリング
してしまうかも

余談
Xilinx FPGAをやる人はaync_regという
制約を覚えておくといいです

2.6　パイプライン

高速化技術（パイプライン）

同期回路を高速動作させるための技術の1つです。
以下の回路をパイプライン化してみます。

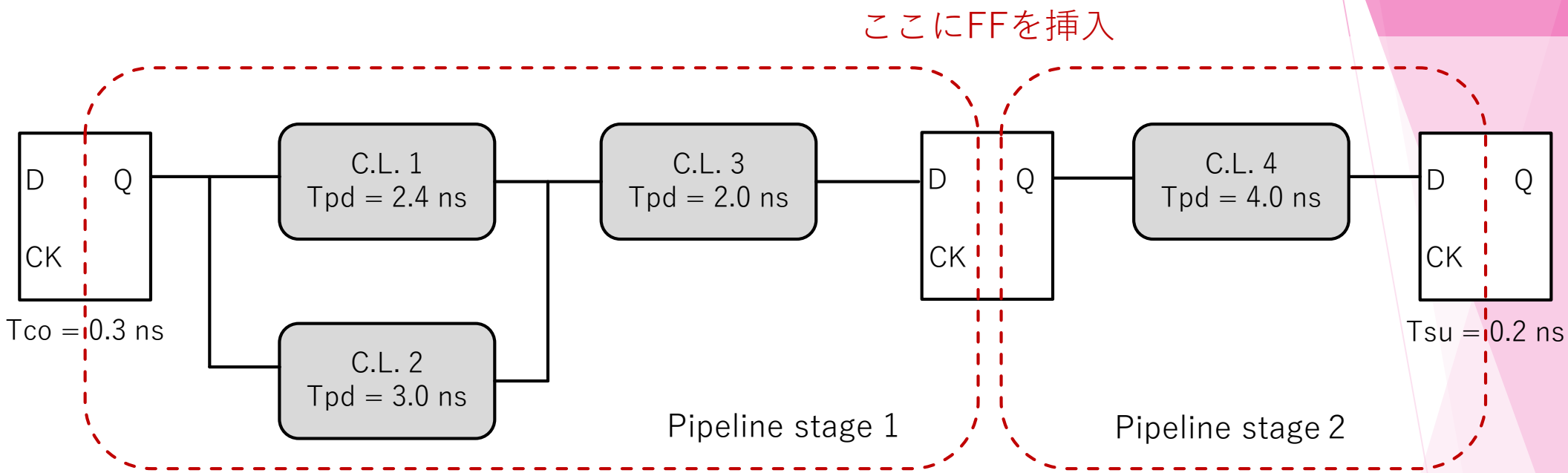


この回路の最大動作クロックは？

Fmax ＝ 1/(0.3 + 3.0 + 2.0 + 4.0 + 0.2)
＝ 1/9.5 ns
＝ 105 MHz

この回路は最高**105 MHz**で動作する

高速化技術（パイプライン）



Stage 1

Fmax = 1/(0.3 + 3.0 + 2.0 + 0.2)
= 1/5.5 ns
= 182 MHz

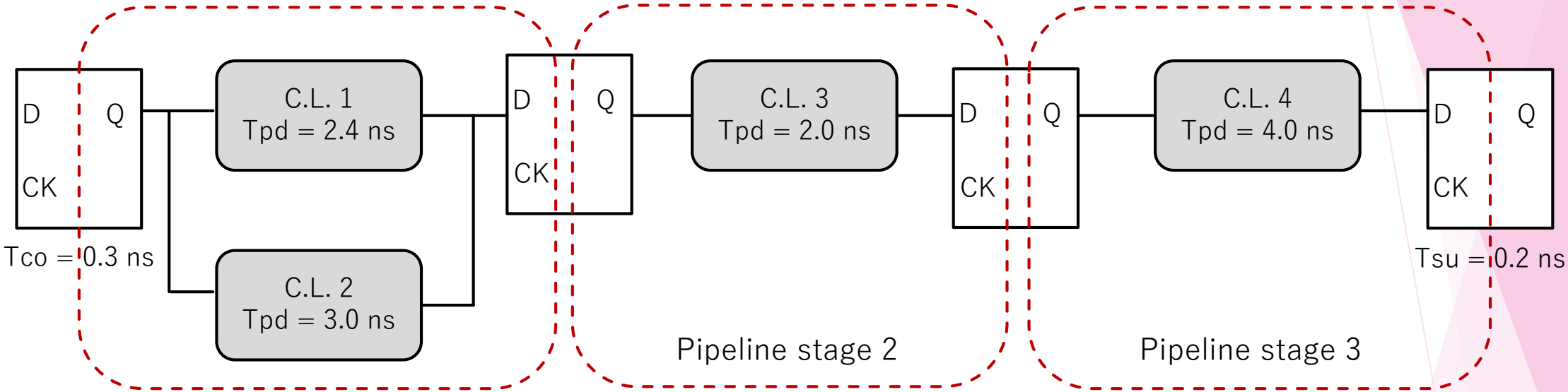
Stage 2

Fmax = 1/(0.3 + 4.0 + 0.2)
= 1/4.5 ns
= 222 MHz

182 MHzへ高速化された

高速化技術（パイプライン）

ここにもFFを挿入



Stage 1
Fmax = 256 MHz

Stage 2
Fmax = 400 MHz

Stage 3
Fmax = 222 MHz

222 MHzまで動作クロックがあがった！

作った回路を高速動作させたいけどタイミングを満たさない場合
パイプライン化できる場所が無いか探しましょう

パイプラインレイテンシ
3×4.5 ns = 13.5 ns

2.7　メモリ

メモリ

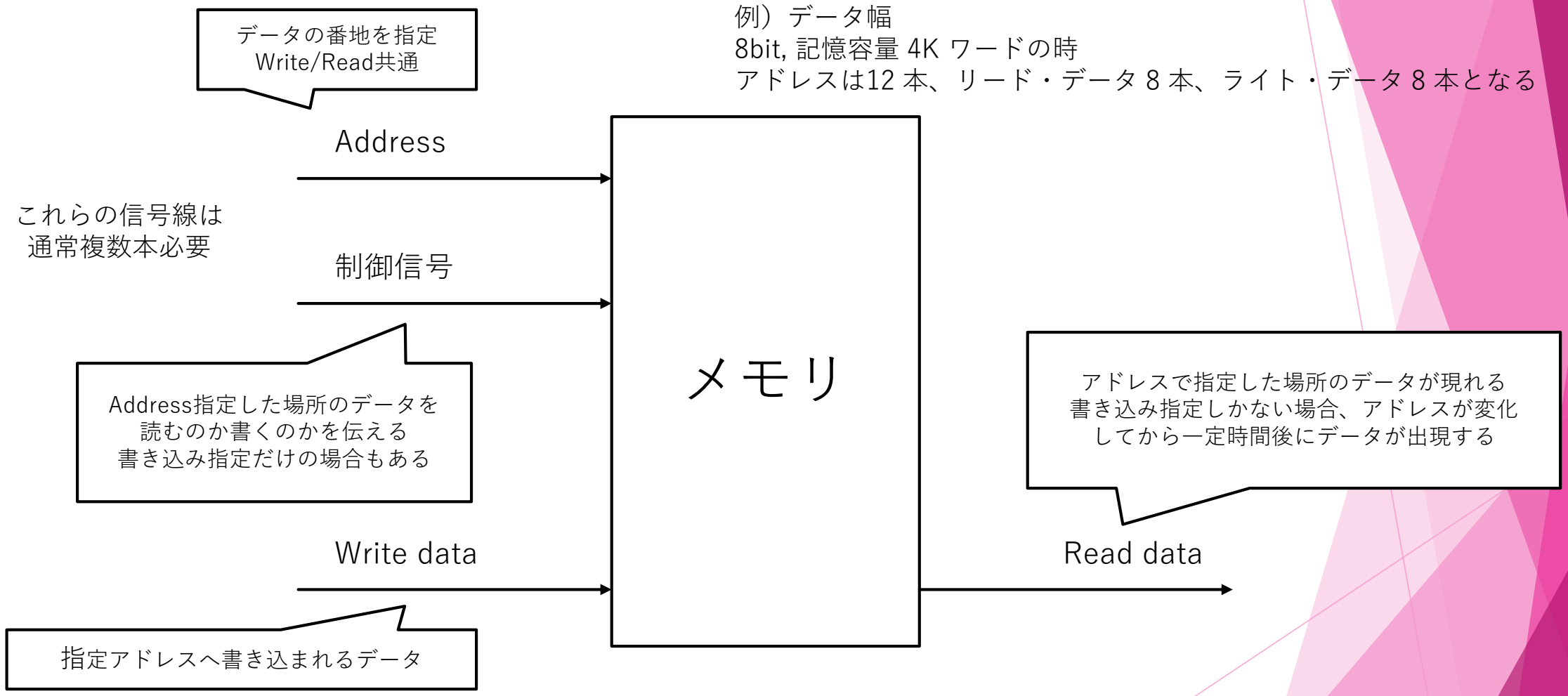
データ記憶に特化した回路
- 数千以上の記憶素子が並んだ回路です
  - D-FFも記憶素子だけど記憶できる情報は1ビットだけ
- アドレスを指定して情報を取り出す
  - アドレスを隠ぺいする場合もある。例：FIFO

必ずしもクロック同期回路ではない
- 現在PCのメモリとして使われるSDRAMはクロック同期だが元々DRAMは非同期だった

メモリの分類
- 揮発性・不揮発性
- 機能
  - ポート数、FIFOなど
- 記憶原理
  - DRAM, SRAM,　Flash ROM, PROMなど

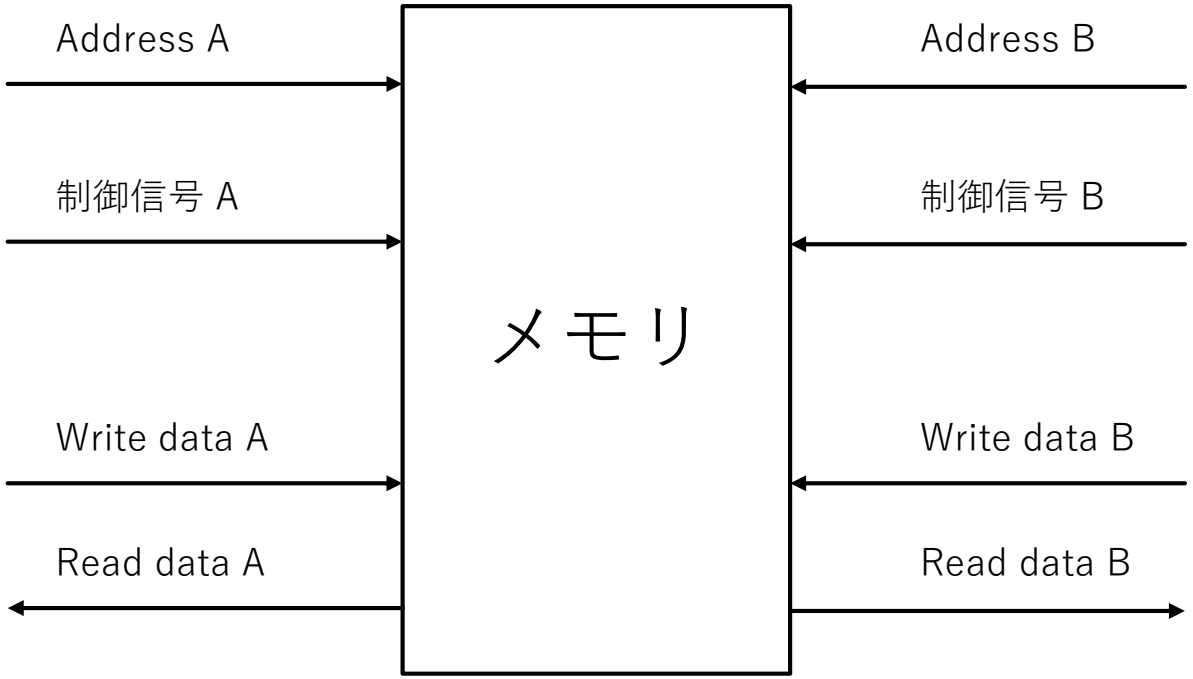
ここでは記憶原理には触れず
機能に絞ってみていきます

RAM (Random Access Memory) の基本構成



Dual-port RAMの基本構成

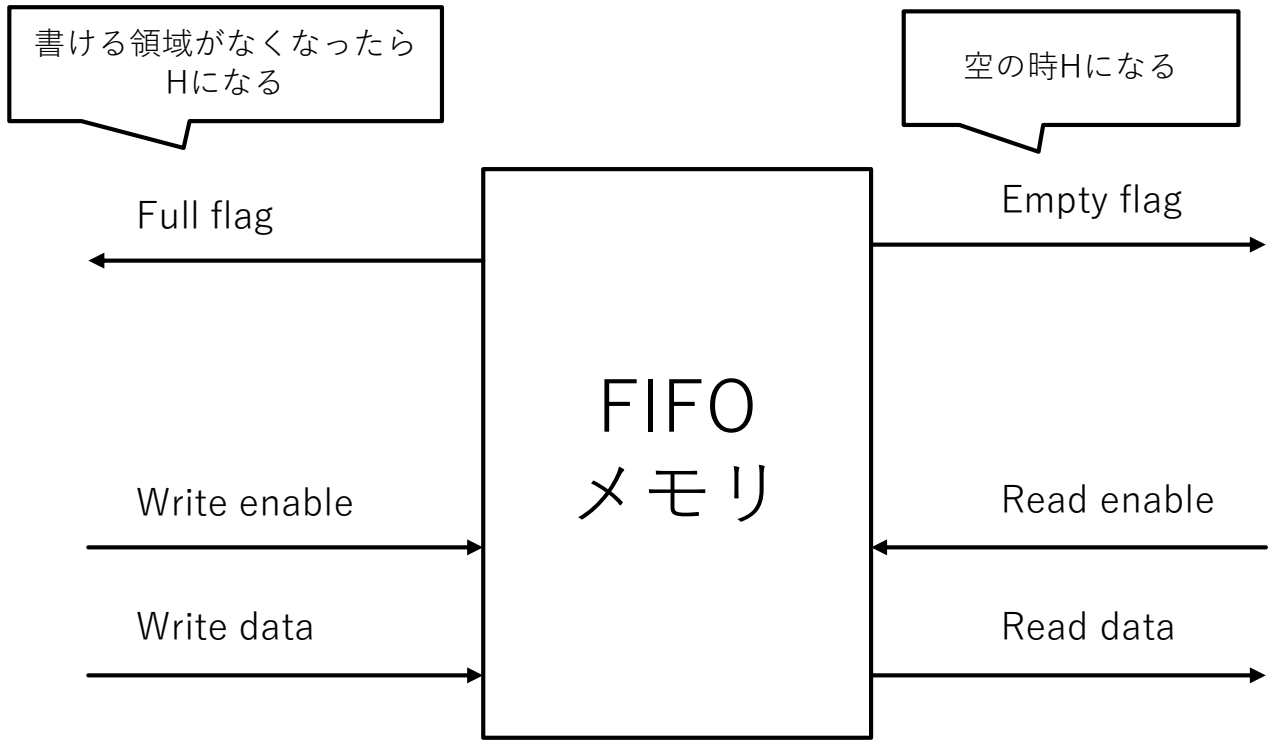
独立なアクセスポートが2つ存在する。記憶素子は共通。
2つのポートを独立に動作させることが出来る。



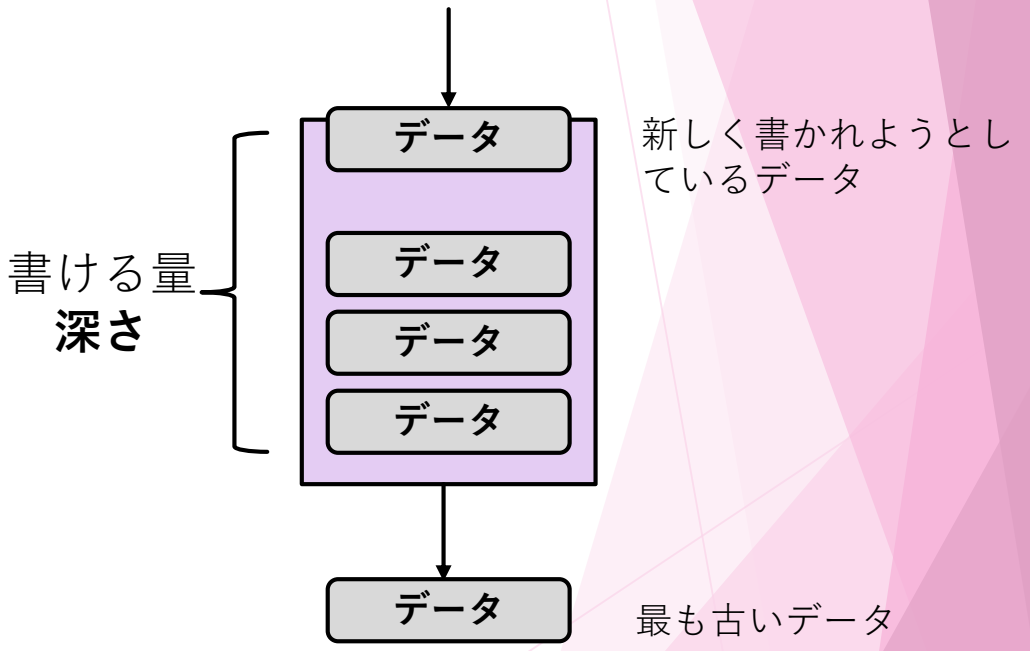
A側を書き込み専用、B側を読み出し専用とすれば
連続的に書きながら読み出すことが可能

FIFO (First-In First-Out)

最初の書いたデータが最初に出てくるメモリ
中身にRAMが入っているがアドレスは隠ぺいされている
一次的なデータ置き場、クロック乗り換えなど頻繁に利用される



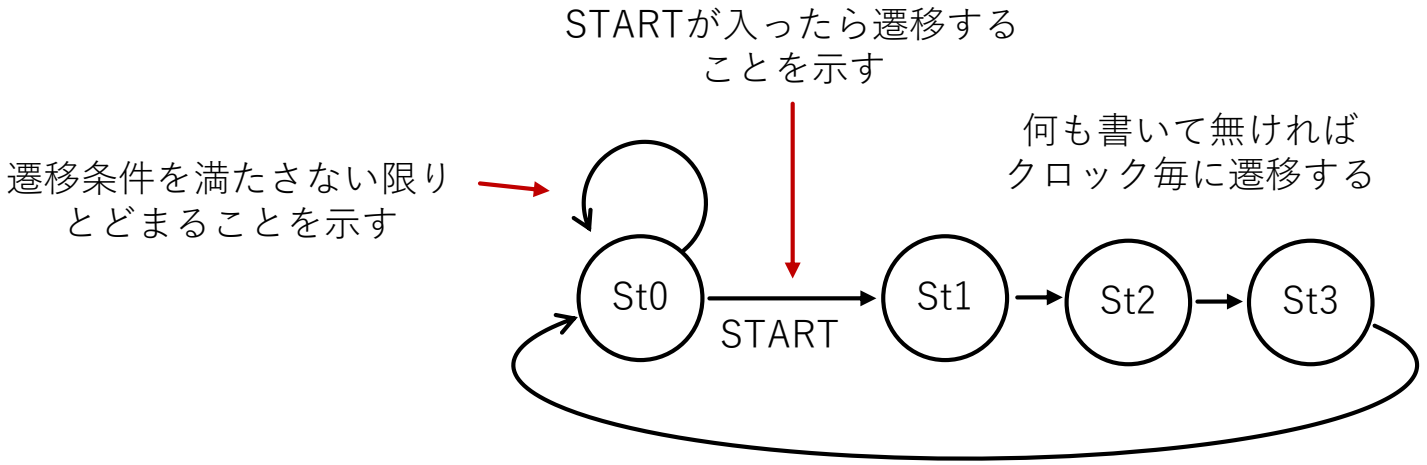
FIFOのイメージ



2.7　ステートマシン

ステートマシン (シーケンサ)

ステートマシンは**回路の制御**に用いられる
複数の状態を用意しておき**状態毎に違う動作を回路にさせる**
状態の出力と切り替えを管理するのがステートマシン



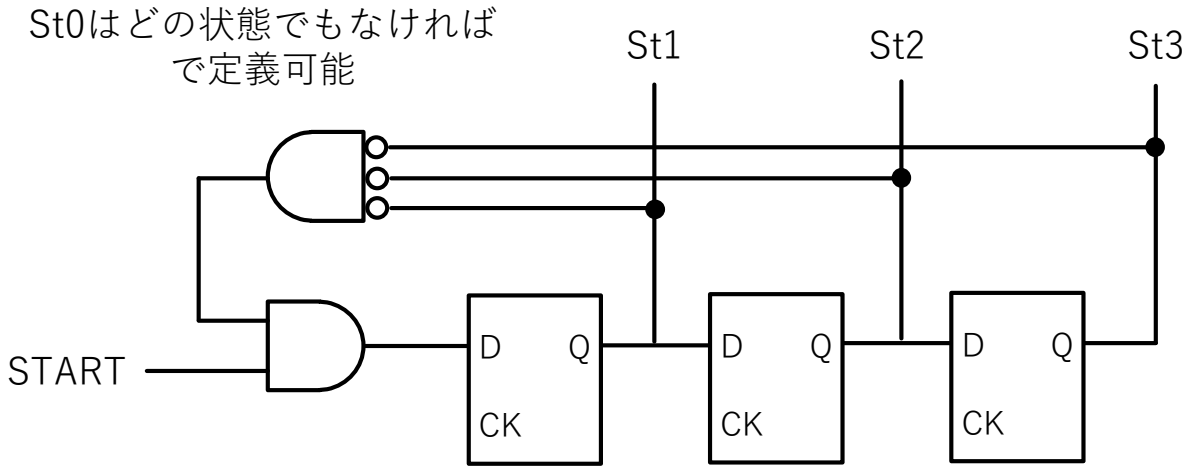
ステート図

それぞれのステートで
回路の動作を変えたい

ステートマシン (シーケンサ)

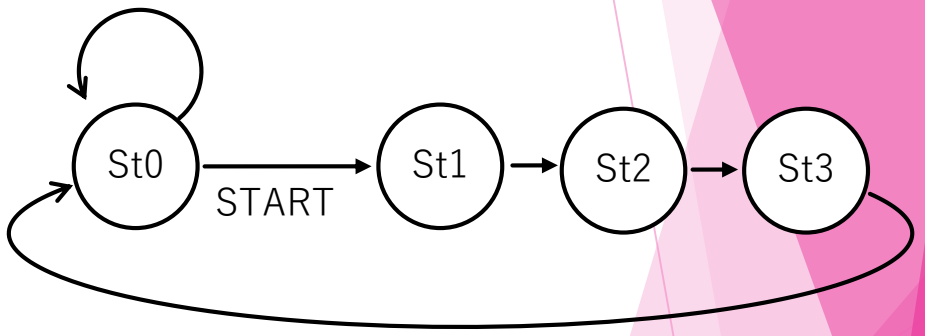
ストレートシーケンサ

- 分岐もウェイトもしない最も単純なシーケンサ



同期回路のためクロック省略

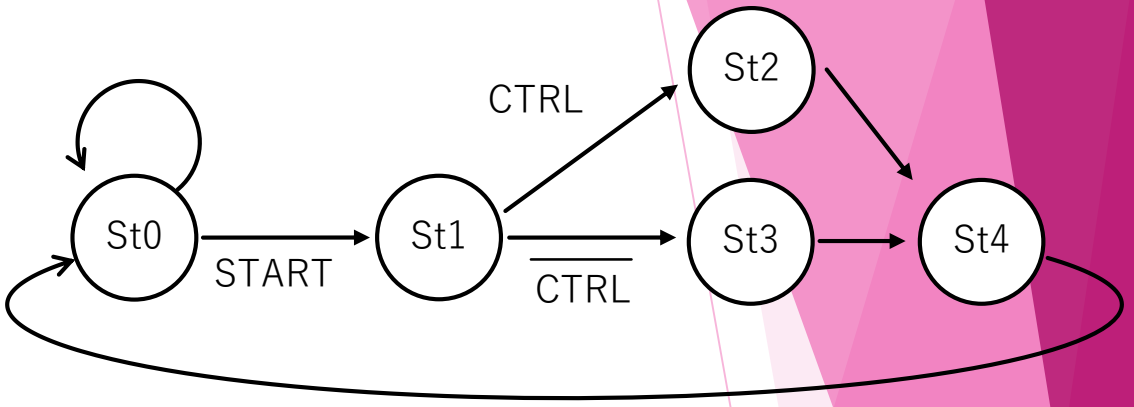
リセット経路も省略
(無いとこの回路は初期値不定)

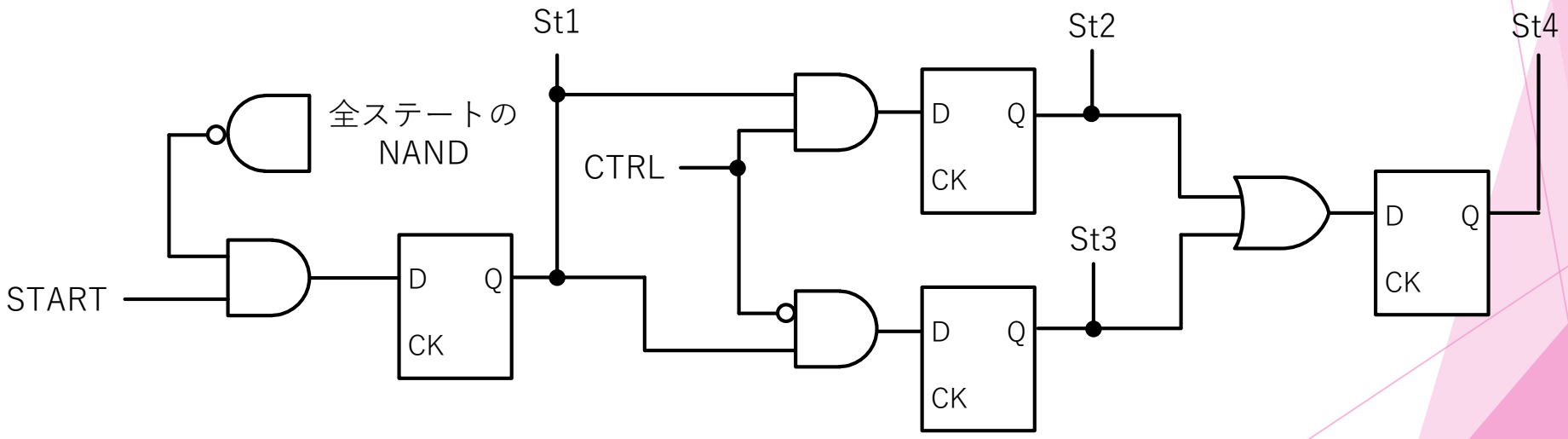


ステート図

ステート分岐

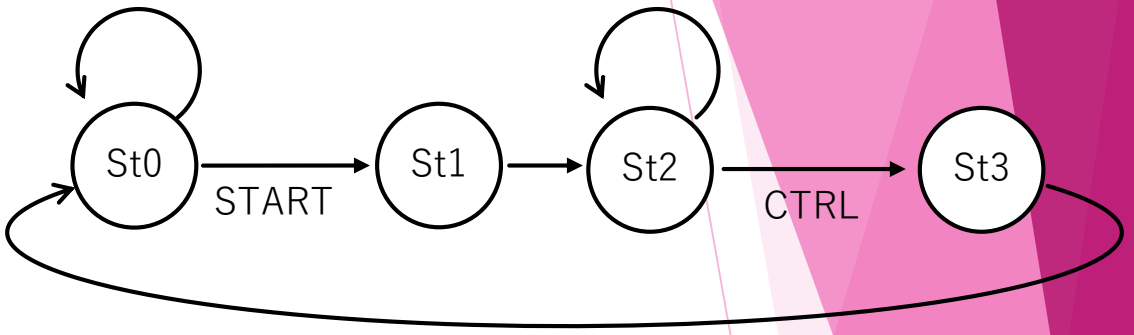
条件によってステートの分岐を与えてみる
- この例ではCTRLの状態によってSt2かSt3へ分岐する



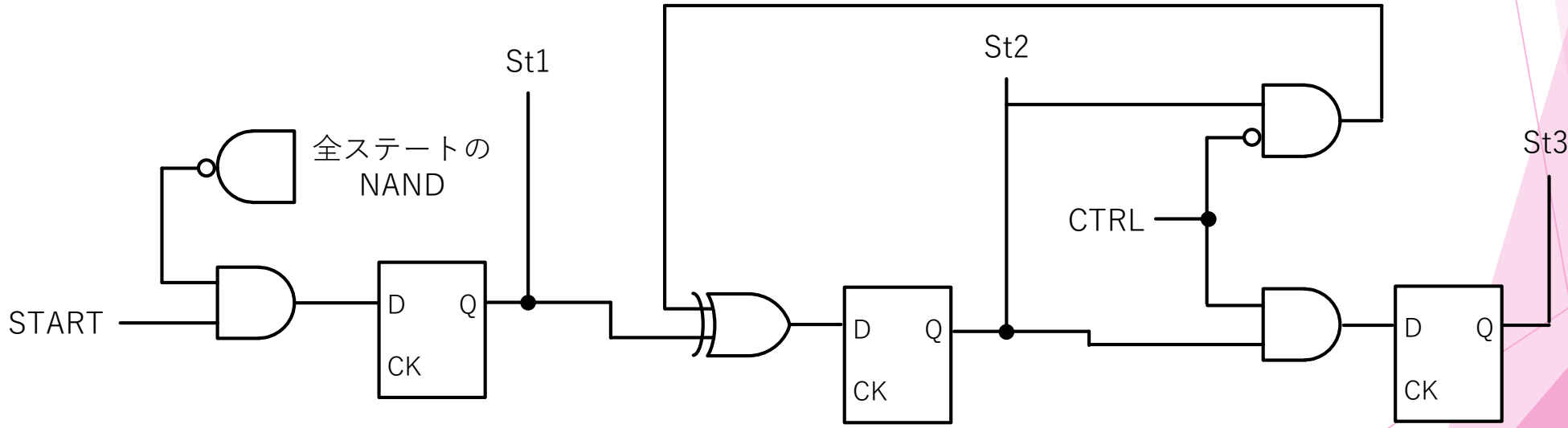


ウェイト

条件が整うまで次のステートに遷移させない
- この例ではCTRLがHIGHになるまでSt2で待機



分岐とウェイトがあれば大体の状態制御はできるはず



履歴

資料原案

- 2018/7/27 第3.0版 内田智久 (Esys, KEK)

- 2020/7/29 v1.0 本多良太郎 (東北大)

３．データ通信技術

2020年7月29日
本多良太郎（東北大学）

内容

ここまで学んだ内容はデジタル回路の構成ブロック
構成ブロック間、部品間、PC間で情報のやり取りをしないといけない
通信のルールを決めておかないと拡張性に欠けるし、
通信に失敗するかもしれない。
ここでは大別して装置内通信と装置間通信についてみていく。

3.1　計算機システムの構成と外部I/F

デジタル通信技術

計算機（CPU/ プログラム）が必要な情報をやり取りするために開発・発展

- **装置内通信**
  - CPU が装置内のメモリなどの装置内蔵デバイスとデータをやり取りするために開発・発展
- **装置間通信**
  - ある装置の CPU/ プログラムが他の装置の CPU/ プログラムとデータをやり取りするために開発・発展

装置内通信、装置間通信は求められる機能や性能が異なるので
それぞれ長所と短所がある

CPUとは？

CPUをハードウエアの視点から見ると

2進数の演算と
2進数の転送

を行っている、ように見える

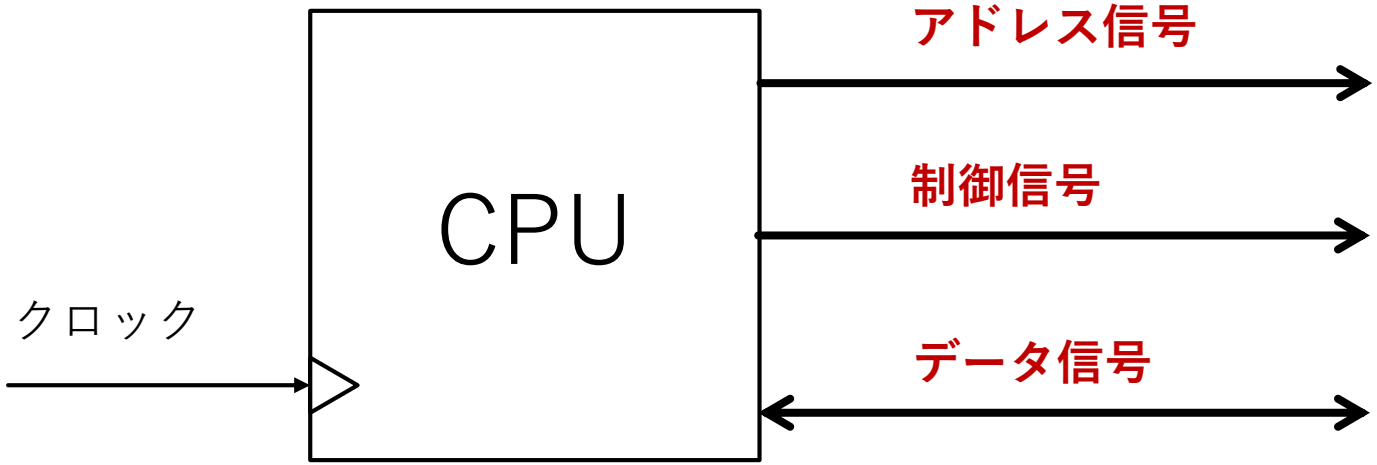
演算はCPUのみが行うので、
システム設計の観点ではCPUの周辺部品との**データ転送の仕組みが必要**

CPUのデータ転送機構

導入された転送機構

CPUが次の信号を制御することでデータ転送する

- どのデータかを指定するためにアドレス信号 (バス)
- データを送るのか受け取るのかを指定する制御信号
- データ値を転送するためのデータ信号 (バス)

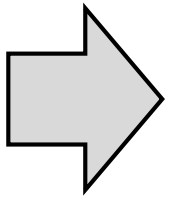


アドレス/データ信号線数は同じことが多い
64 bit CPUは64本の信号線がある

計算機システムの構成

好きな構成でシステムを作りたい
- CPUが処理する内容はユーザーに依存する
- ユーザー毎にCPUに接続される周辺機器（部品）が異なる
  - フロントエンドもCPUから見れば周辺機器の一つ
- どんな部品であれCPUから制御を受けないといけない

データを処理する物は決まっている
- **データをどのように使うのか知っているのはCPUのみ**
- CPUが周辺機器（部品）を指定して、その機器（部品）に対してデータの読み書きができればよい

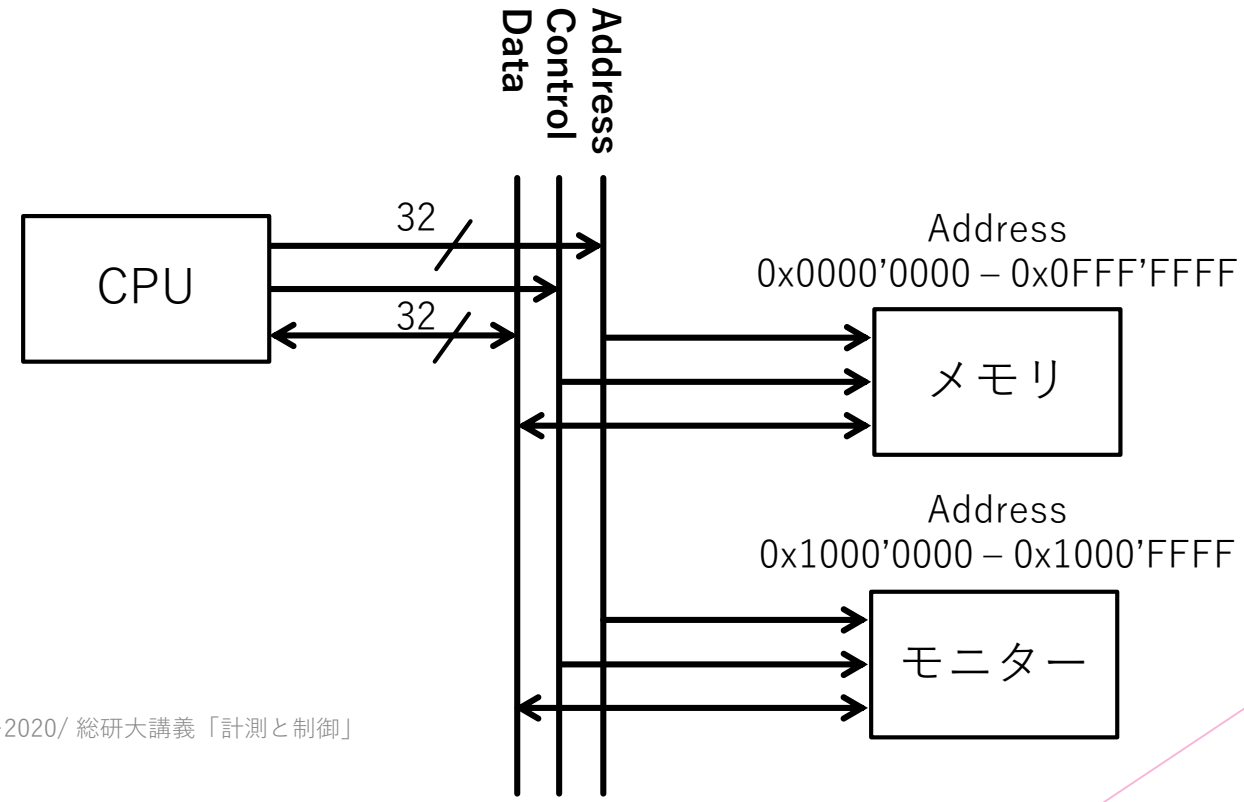


システム・バスの導入

システム・バス

- アドレス信号で対象装置（部品）を指定　　：**アドレス・バス**
- データ信号でデータやり取り　　：**データ・バス**
- 制御信号で読み書きを制御

複数の回路に接続され共有される信号線をバスを呼ぶ
現在では意味が転じて複数本で構成される信号をバス信号を呼ぶことも



参考文献

CPUに関する書籍

- 渡波郁、CPUの創り方、毎日コミュニケーションズ
- 伊藤剛浩、川田裕貴、独自CPU開発で学ぶコンピュータの仕組み、秀和システム
- パターソン&ヘネシー、コンピュータの構成と設計（第5版）、日経ＢＰ

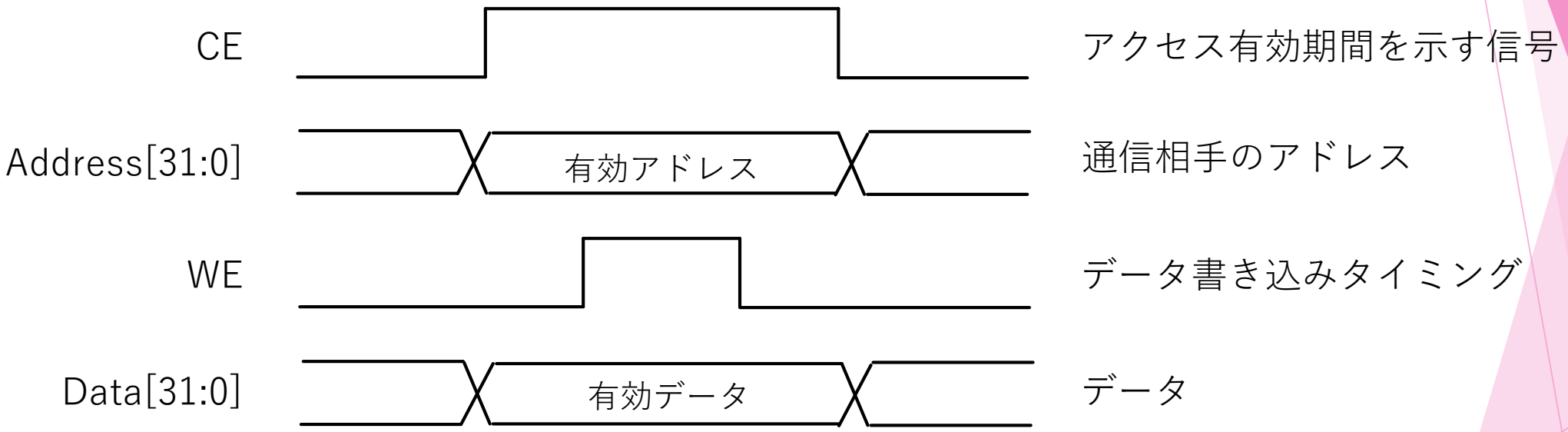
3.2　データ通信の基礎

データ通信の共通概念

- 通信開始者が居る
- 送信者と受信者が居る
- 送信者、受信者はアドレスで指定（区別）
  - 1:1の通信では転送先と転送元が一意なので使わない時もある

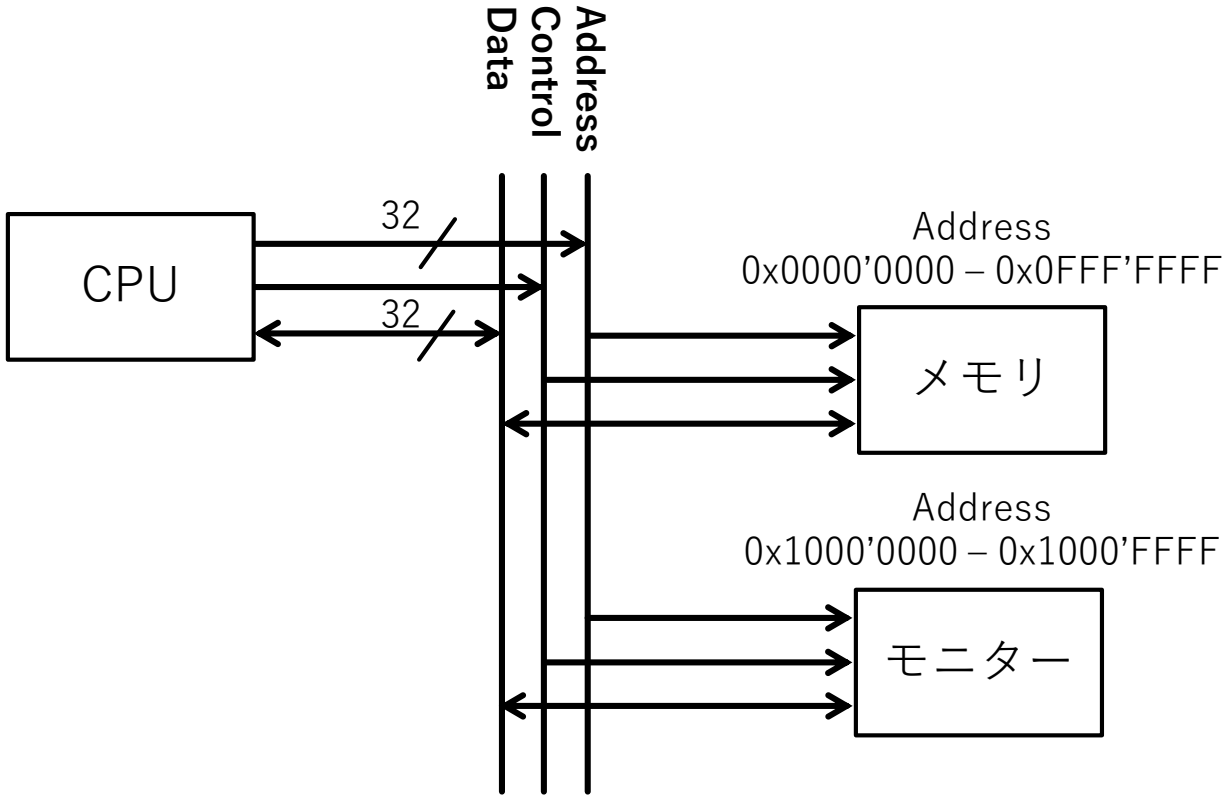
パラレル通信

データを複数の信号線を使って転送する手法



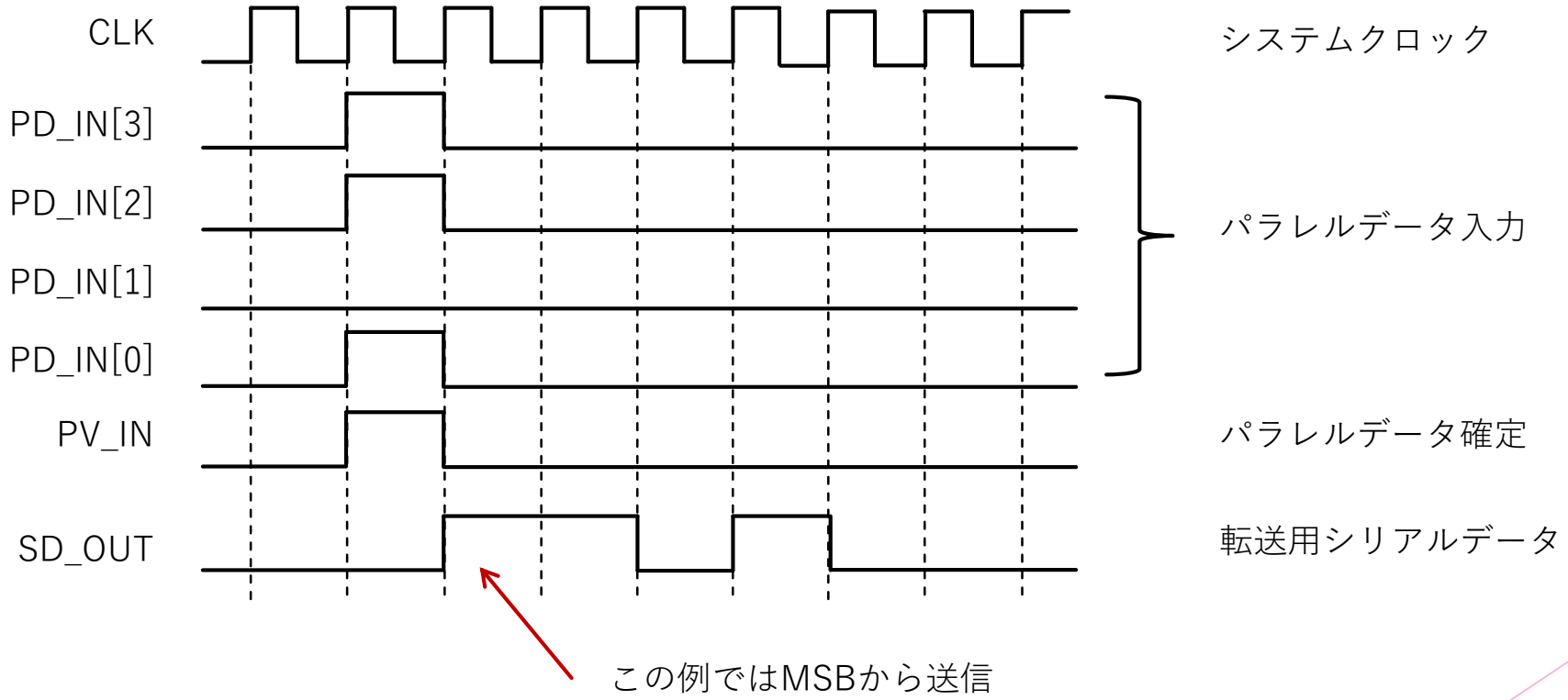
通信手順の例

パラレル通信の例


Address
Control
Data
CPU
32
32
Address
0x0000'0000 – 0x0FFF'FFFF
メモリ
Address
0x1000'0000 – 0x1000'FFFF
モニター

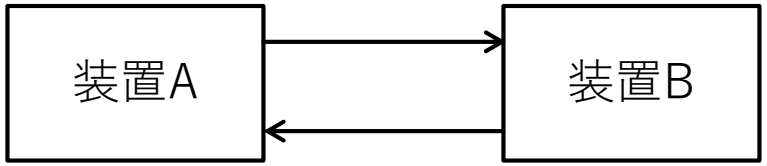
シリアル通信

多ビットのデータを時間方向に展開して転送する方法



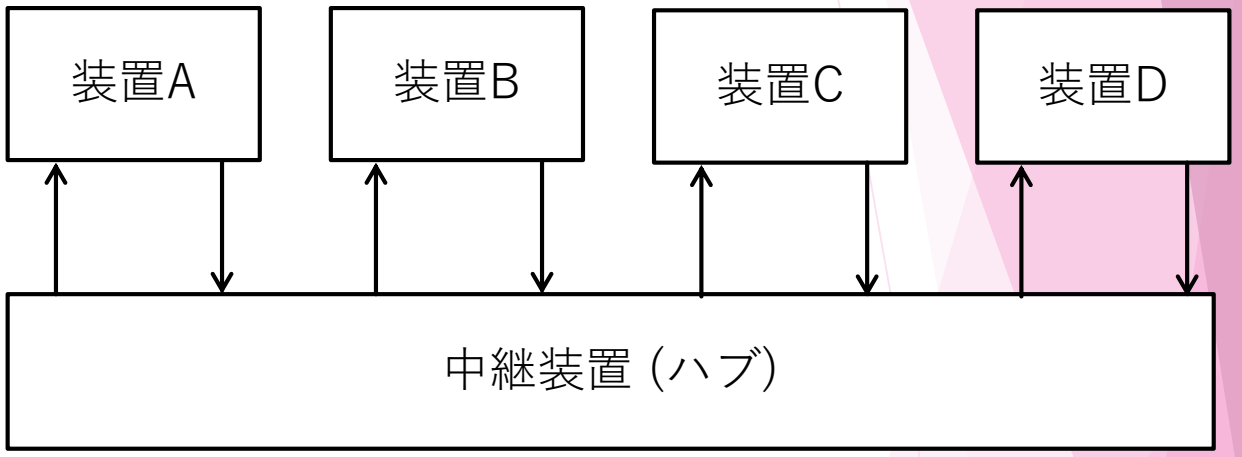
シリアル通信の構成例

1対1が基本



最低限2本線があればよい (送信と受信)

装置が複数ある場合
中継装置を使って信号を分配する



ところでアドレスはどうやって指定する？

パケット

シリアル通信はビットの時系列なので工夫しないと使いにくい

問題１：データの切れ目がない、分からない
問題２：誰から誰へのデータなのか分からない（アドレスがない）

パケット

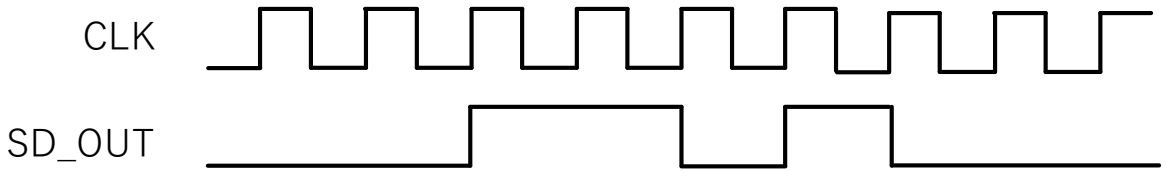
解決策1
通信開始時（または未使用時）に特別なビット列を送る
例えば、
- 未使用時は０を流し続ける
- 通信開始前に必ず１を一つ送る
- データ長は必ずNビット

解決策2
パケット
- アドレスなどの制御データを含むデータの集合
- アドレスやデータを格納する先頭からの位置を決めておく

パケットの概念を用いて定められたフォーマットを用いて通信する
例えば、
- 先頭から３２ビットは宛先アドレス、
- その次が送信元アドレス
- その次がデータの長さ
- その次が送りたいデータ、など

少し進んだ話

シリアル通信を採用する理由の1つが信号本数が少ないこと
クロックに1本信号線を使うのはもったいない



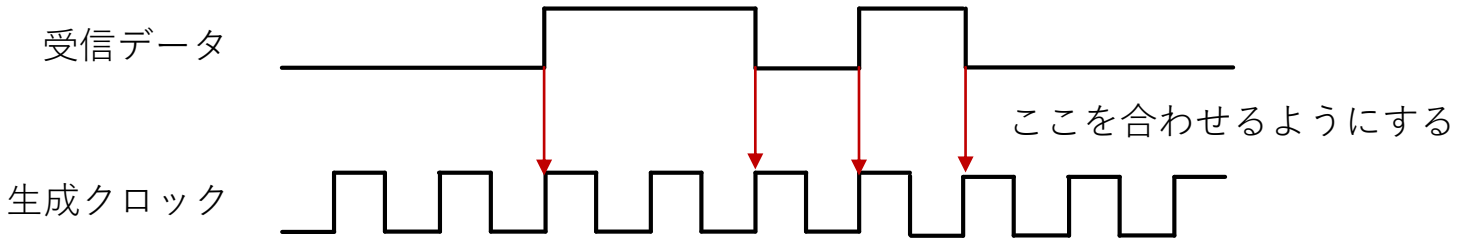
クロックを送らなければ1本減る

とはいえ…独立した装置間で完全に同じクロックを作ることは出来ない
正確に1ビット幅の時間を知るためにはどうしたらいいか？

クロック再生

データはクロックに依存して変化する
データの遷移するタイミングからクロックが再生できるはず！
(実用上問題ないレベルで再現できる)

受信したシリアルデータの変化点に生成したクロックの変化点を合わせるように調整する
こまめに調整する事でクロック精度を上げる作戦



余談
長期間データが変化しないと調整できないので必ずデータが変化するようにデータを加工して送信する事が多い。
これを変調と呼ぶ（4B5B/8B10B などが有名）。 0b0001 を 0b01001 へ変換するなど。

パラレル通信・シリアル通信どっちを選ぶ？

より多いデータをより速く転送したい
データ通信規格は年々**高速化**

パラレル通信
- 同一基板上など近距離通信向き
  - 高速動作では複数の信号線間で配線遅延を揃えるのが難しい
  - 2000年代前半に高速化の流れに置いてかれるように
- 近距離ならある程度高速化可能
- 回路は比較的簡単
- 占有スペースが大きい

シリアル通信
- 遠距離通信、または、高速通信向き
- 回路が複雑
- 一般にソフトウエアによる処理が必要
  - パケットを処理する為
- 小径ケーブルで接続可能

3.3　装置内通信

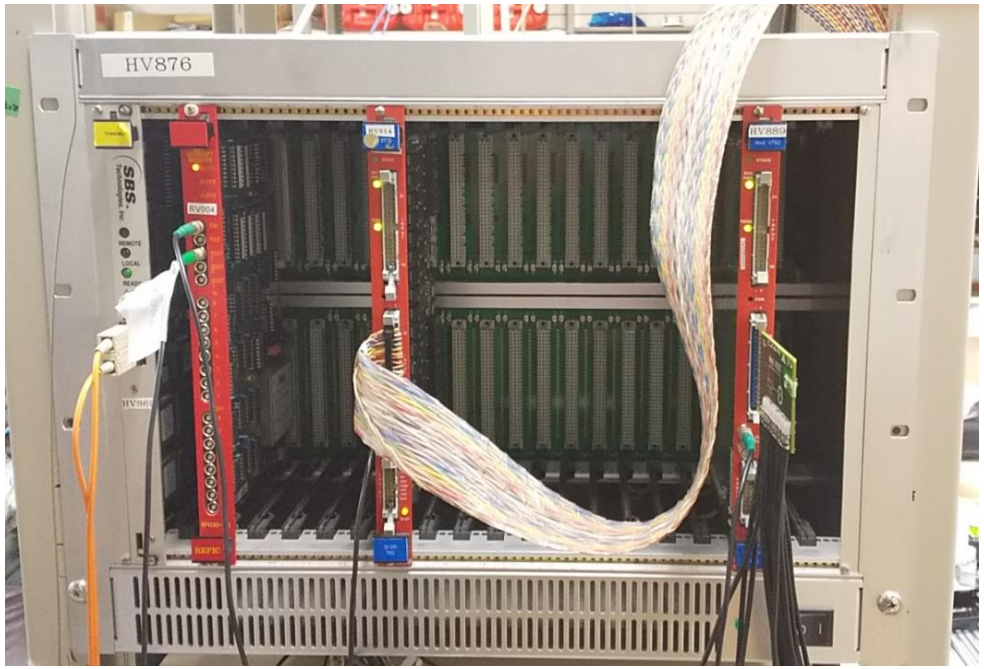
装置内通信

ここではPCインターフェースに焦点を絞る

- CPUと周辺機器間の通信として開発されたものが多い
- CPUバスの拡張
  - アドレス＋データ
  - CPUが全ての通信を管理
  - 全ての周辺回路はCPUが使うためにあると考えられている
  - 通信開始者は基本的にCPUのみ
- 短距離
  - 装置内なので短距離で十分
  - 長くても数メートル程度
- 代表例
  - VME bus, PCI bus, USB, Bluetooth

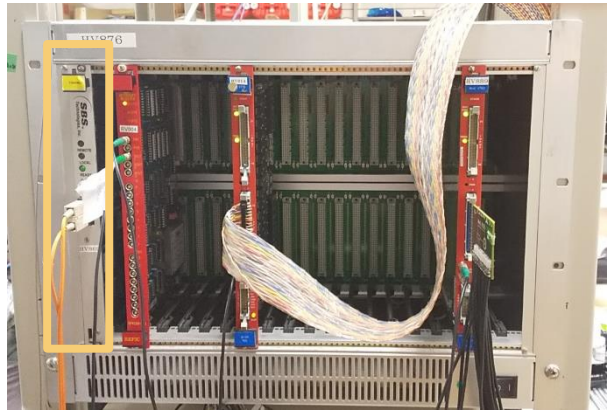
VME

- 1980年代に広く使われた規格
- 加速器科学分野では現在でも使われている
- クレートに各モジュールを差し込んでシステムを組み上げる方法



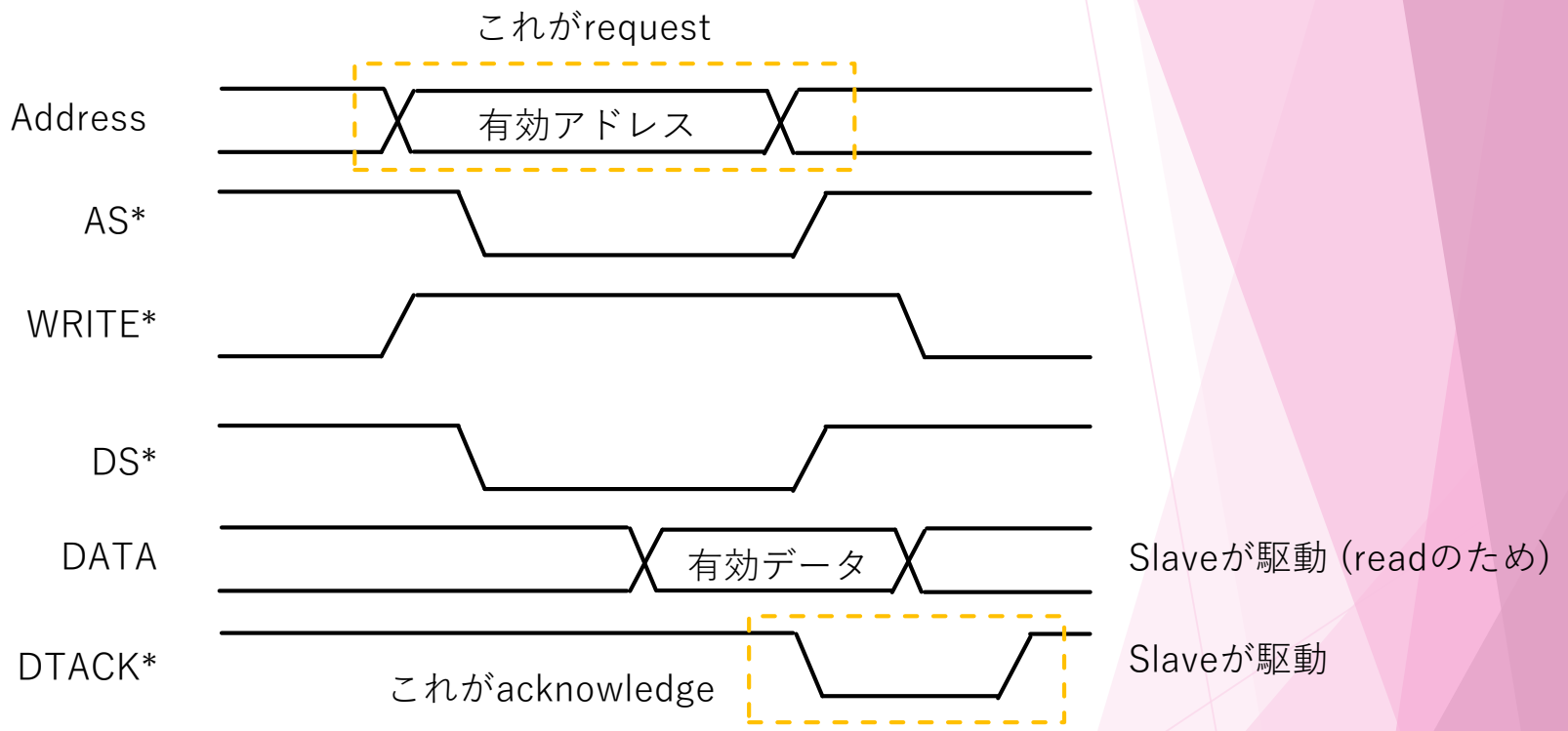
VMEのバスサイクル

CPUモトローラ 68000 系のバス拡張
非同期バス
Request-Acknowledge 型のプロトコル



バスマスタ
- 通信開始者

スレーブ
- それ以外



リードサイクルの例

VMEの参考文献

古い規格なので現在発売されている書籍は無い

VITAホームページ
- http://www.vita.com

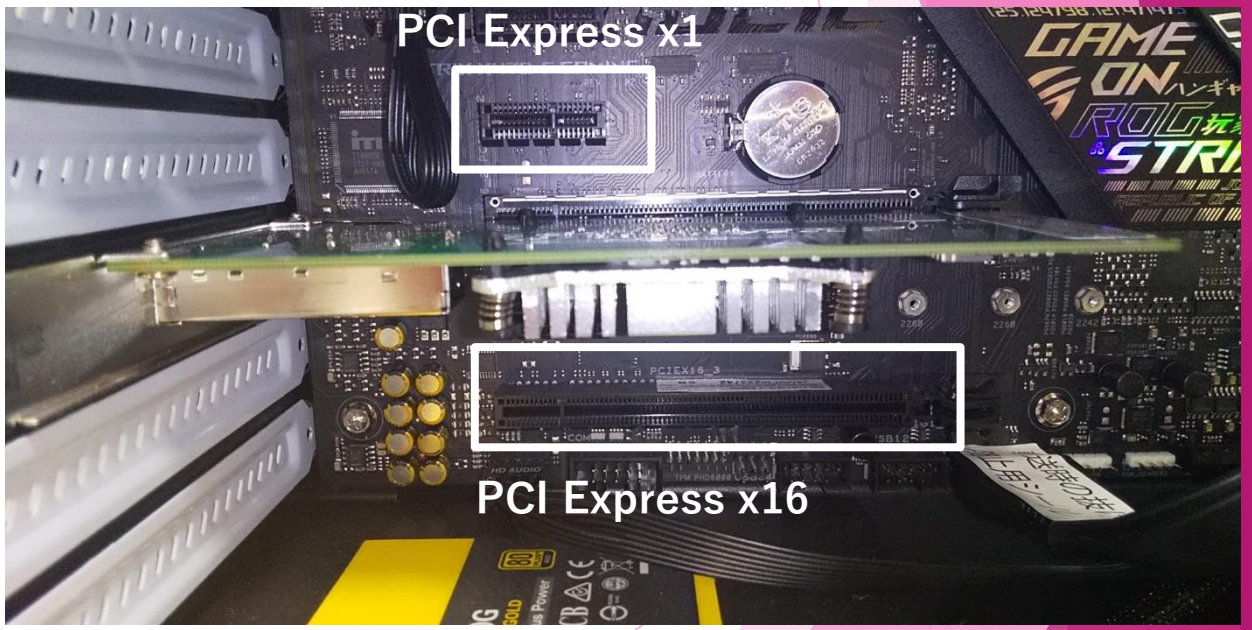
VME32
- 過去の遺産はVME32を採用していることが多い
- VMEbusアーキテクチャマニュアル、CQ出版、1986

PCI

Intel系CPUの拡張バス規格
ISAバスを置き換えるためにCPUアーキテクチャに
依存しない広帯域・汎用バスとして制定

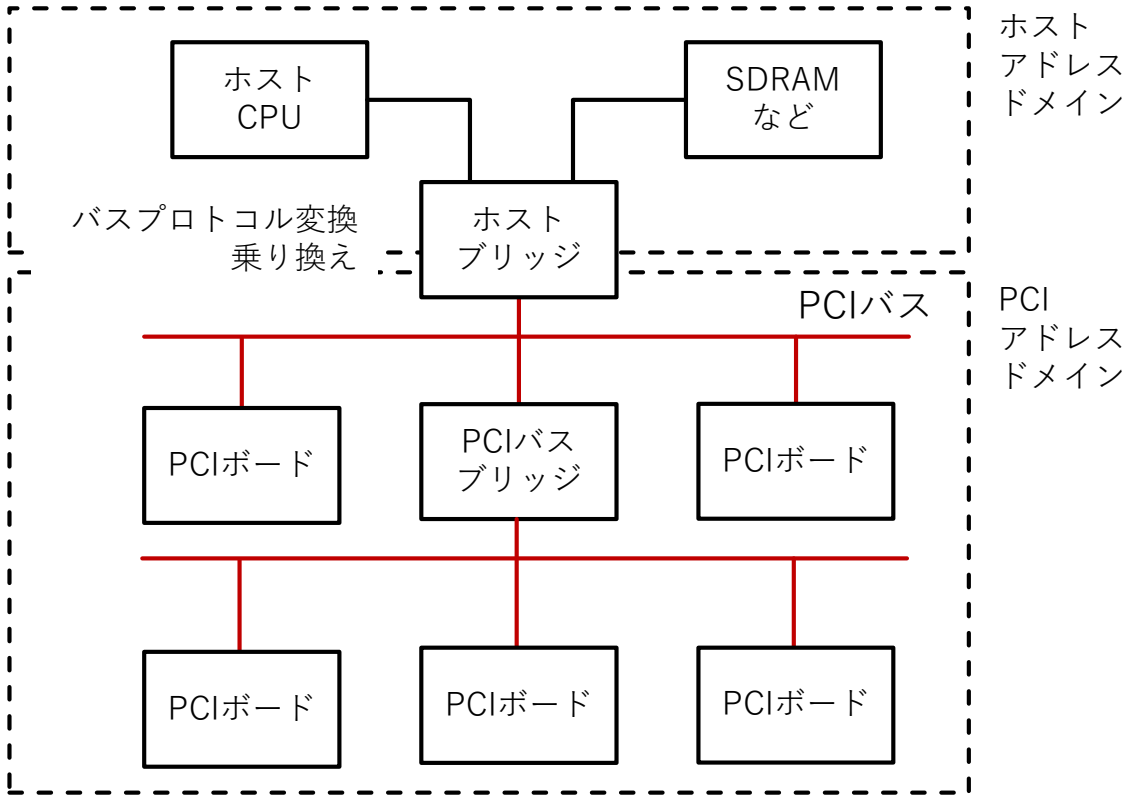
歴史的変遷
- ISA→PCI→PCI-X→PCI Express
- PCI-Xまでがパラレル通信、PCIeはシリアル

PC筐体内にカードを挿して使用する
- 付加機能をカードにより追加する

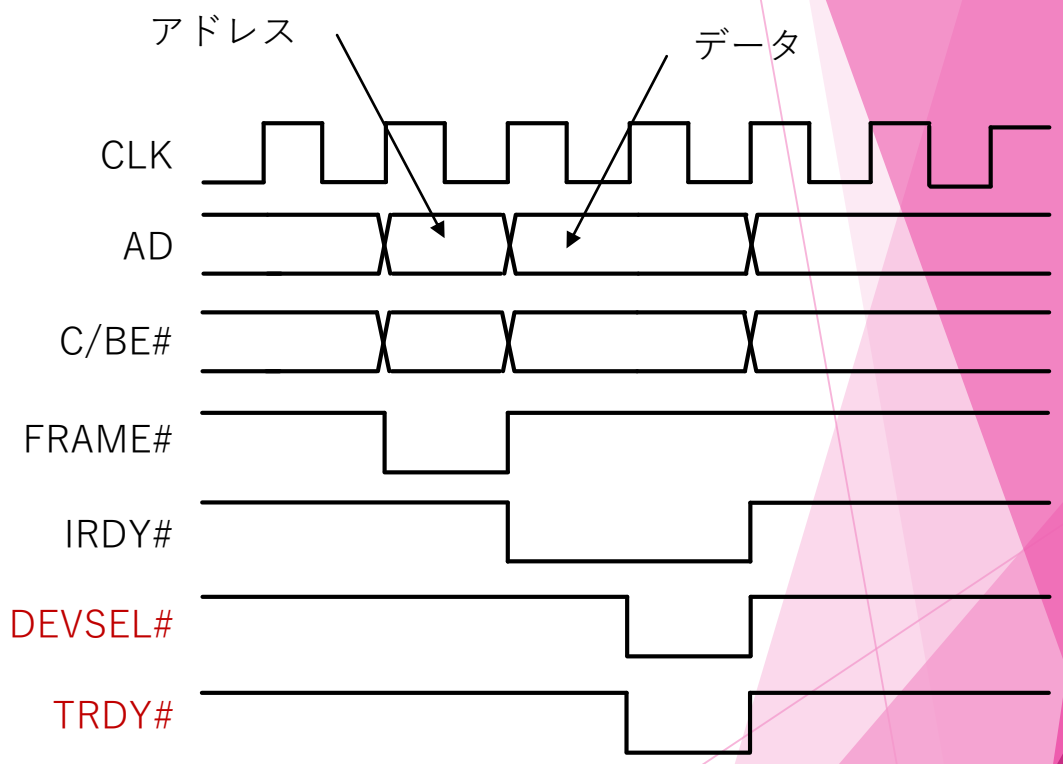


PCIの構成とバスサイクル

同期バス
アクセスを始めるものをInitiator
アクセスされるものをTargetと呼ぶ



アドレスとデータを時間分割して送るマルチプレクス方式



クロック周波数：33MHz

Counter[2]

PCI Express

シリアル通信
- 複数のシリアル信号を使うことができる
  - リンクが1本　　x1
  - リンクが16本　　x16

データ帯域 (Gen1)
- 物理層帯域　　2.5 GT/s/link (giga-transfer per second)
- データ層帯域　0.25 GB/s/link

アクセス方式はネットワークに近い
- パケットを用いる

PCIバスと互換性を有する
8b/10b変調を利用しているため物理層帯域とデータ層帯域で区別している

PCIの参考文献

PCI Expressの書籍は現在も販売されているがPCIについては絶版になっている

- PCIバス&PCI-Xバスの徹底研究、インターフェース増刊、CQ出版
- PCIデバイス設計入門、CQ出版
- 荒井他、PCI Express入門講座、電波新聞

3.4　装置間通信

装置間通信

- **全ての装置が通信を開始できる**
  - 装置内通信との大きな違い
  - 全ての装置が持つ権利は同じ
- アドレスにより相手を指定
- 長距離
  - 目的により長距離通信が可能
- 代表例
  - Ethernet, Wireless LAN

Ethernet

- **シリアル通信**
  - 送受信の2対が標準
  - 1対双方向も企画されているが現在はほとんど使用されていない
- ネットワーク通信規格のデファクトスタンダード
- 複数の転送レートに対応
  - 10Mbps, 100Mbps, 1Gbps, 40Gbps, 100Gbps
- 異なる転送レート間もHUBを介して接続可能
- メディアが複数規格化
  - 光ファイバ、UTP(Unshielded Twist Pair, いわゆるLANケーブル)など

Ethernetの構成例とフレーム

各装置が独立に動作することが出来る
データ流量を制御する管理者がいないので許容量を超えればデータは破棄される

- 通常は**TCP/IP**と呼ばれる通信規格をもちることで補完する
  - 通信手順やデータが無くなった時の回復方法などが決められている

フレーム構造 (DIXイーサネットの場合)

プリアンブル	宛先アドレス (MAC)	送信元アドレス (MAC)	タイプ	データ	FCS
8byte	6byte	6byte	2byte	46-1500byte	4byte

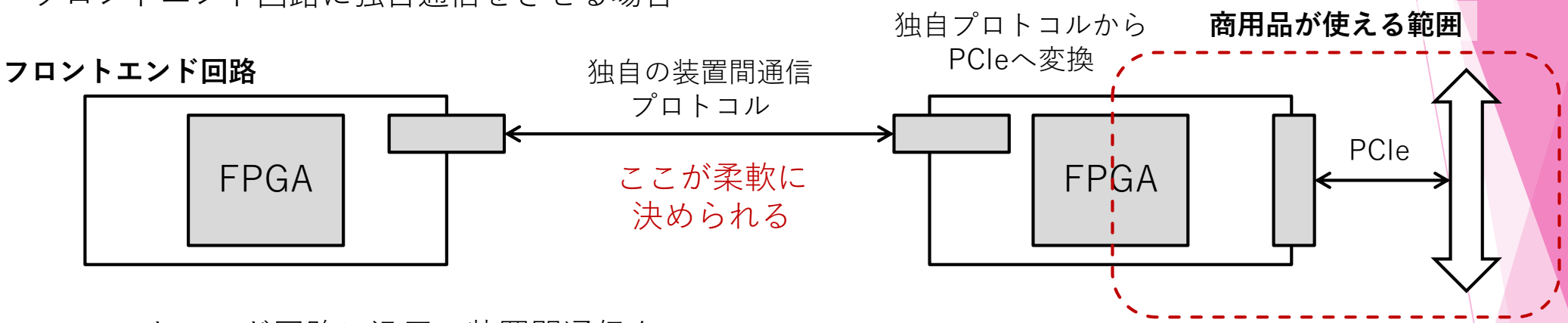
フレーム送信の開始を
認識させるための特殊データ

IPアドレスではない！
IPは次の層 (第3層)

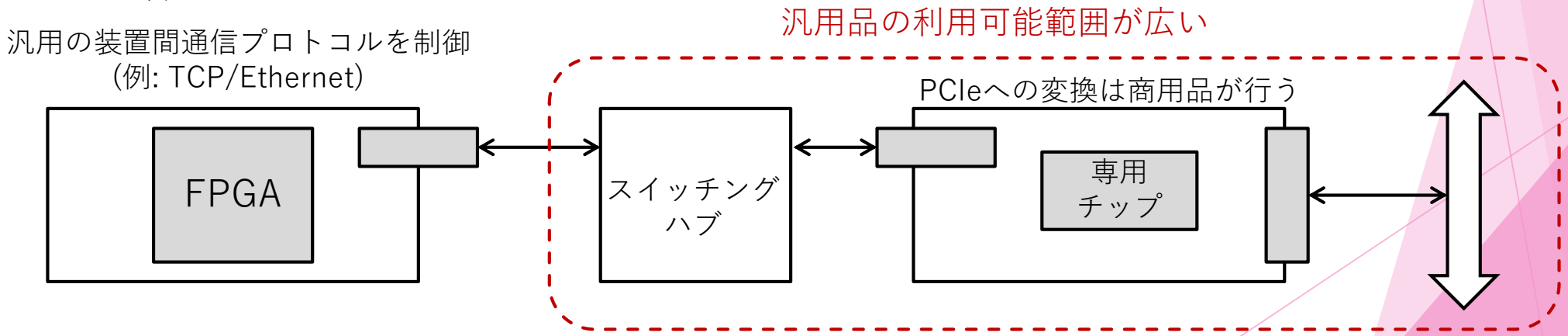
送受信でデータが化けてないか
チェックするのに使う

装置内・装置外通信とフロントエンド回路

- フロントエンド回路に独自通信をさせる場合



- フロントエンド回路に汎用の装置間通信をさせる場合



参考文献

ネットワークに関する書籍

- W. Stevens, TCP/IP Illustrated, Vol. 1: The Protocols, Addison-Wesley Professional.
- 榊正憲、イーサネット＆TCP/IP入門、インプレスジャパン
- 瀬戸他、ギガビットEthernet教科書、アスキー

履歴

資料原案

- 2018/7/27 第3.0版 内田智久 (Esys, KEK)

- 2020/7/29 v1.0 本多良太郎 (東北大)

４．デジタル技術の例

2020年7月29日
本多良太郎（東北大学）

内容

ここまで様々な事を学んできた

- デジタル回路の構成要素
- 構成要素同士の通信方法

これらを組み合わせて出来るものの具体例を見ていく

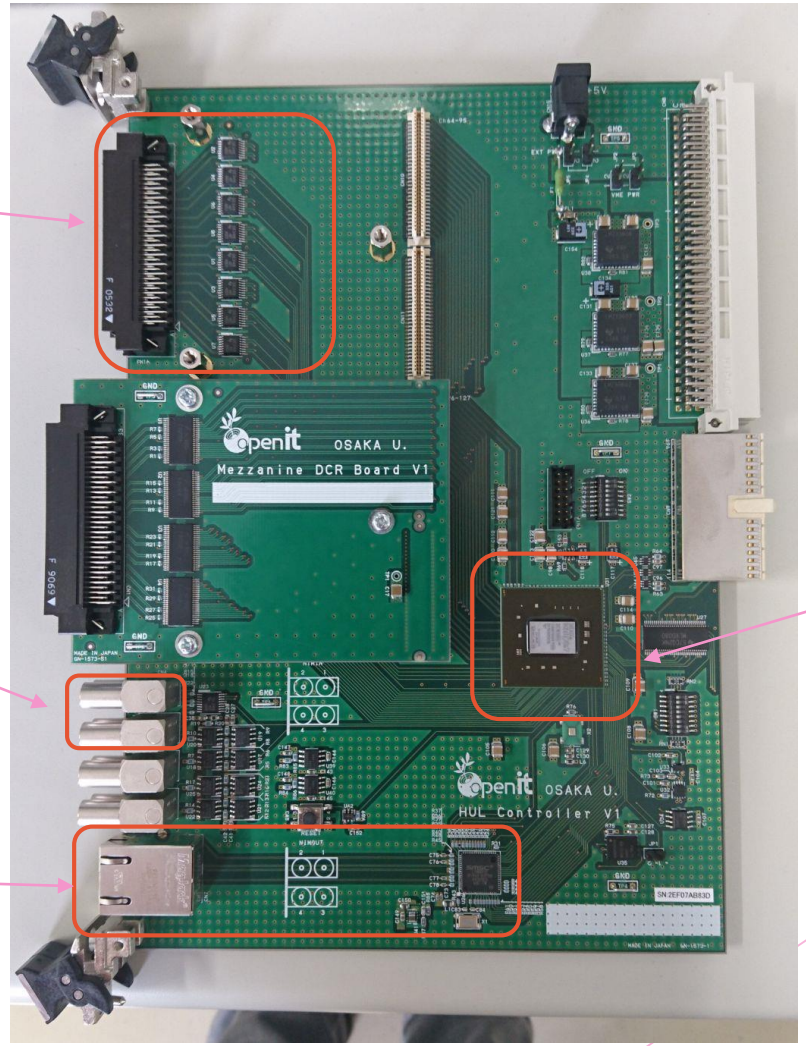
Common-start-type single-hit TDCを実装する

Stop
- 検出器の信号（コンパレータ出力）
  - LVDS入力
  - バッファICでLVCMOSへ変換
  - FPGAへ入力

⇕ 時間差を測る

Start
- DAQ trigger
  - NIM信号
  - LVCMOS変換しFPGAに入力

データ通信
- 装置間通信
  - (TCP/Ethernet)でPCへ転送



TDCの本体
- 時刻測定
- イベントビルド
- 通信プロトコル制御

回路を作り始める前に

仕様を決める

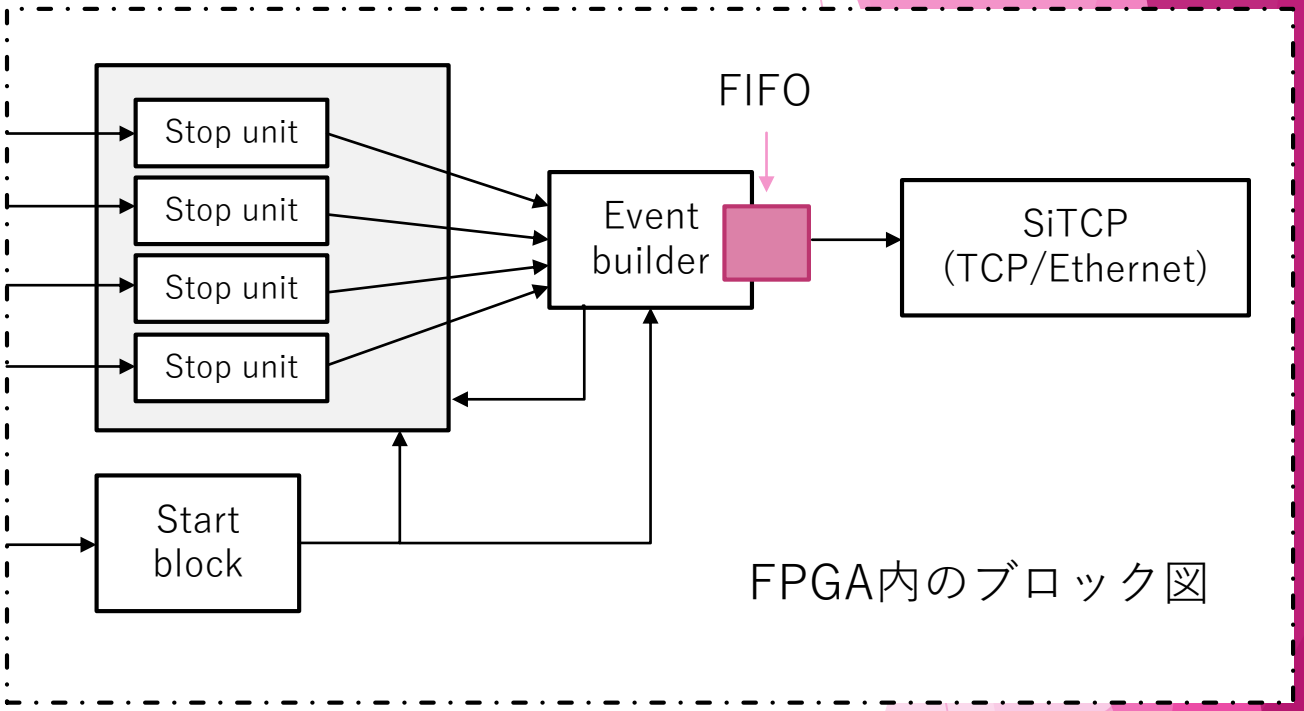
- 何チャンネル入力？　　検出器構成に依存
  - 4にしよう
- 時間精度は？　　測りたい物理量に依存
  - 10 nsで十分 (100 MHz)
- 測れる時間範囲は？　　実験構成に依存
  - Start入力から1 us必要
- Busyの長さは？　　トリガーレートや間隔に依存
  - 1 us程度に抑えたい

仕様を決めることは極めて大事です！
決めずになんとなく作り始めることだけは避けましょう！

仕様が決まったら

私の場合は…

- ブロック図を書く
- 各構成要素のタイミングチャートを書く
- (FPGAなら) HDLへ落とす

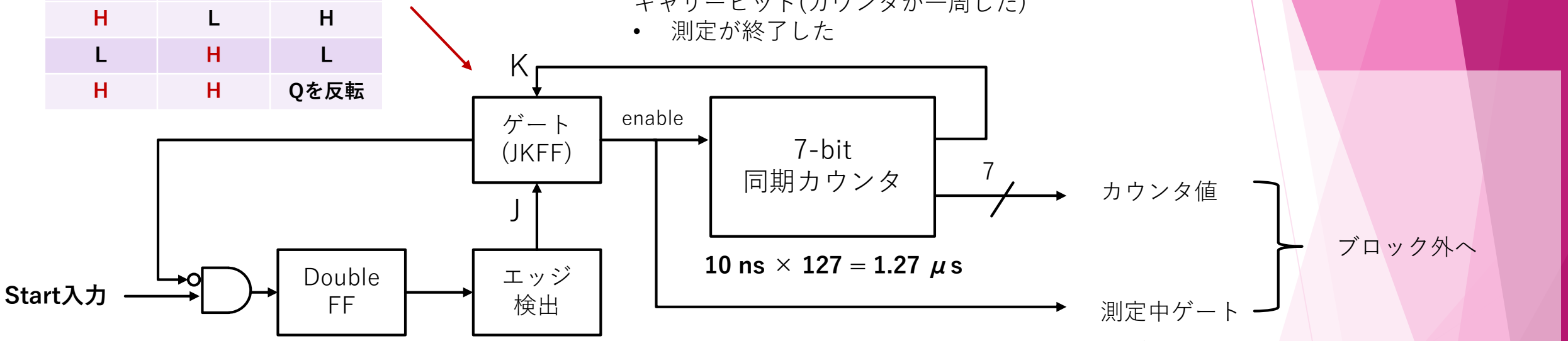


FPGA内のブロック図

今から見ていく構成は分かりやすくするために
極端にシンプルだったり、無駄があったりします
どんな時でも安全に動く構成ではないので気を付けてください

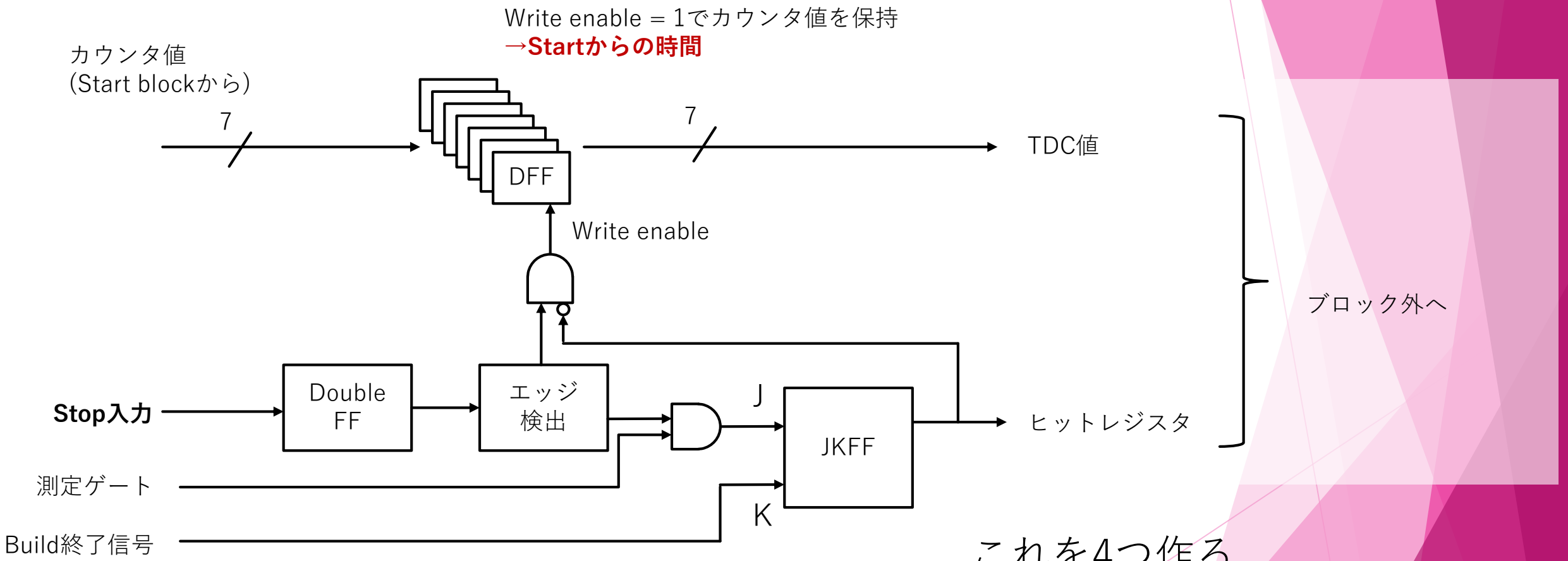
Start block

J	K	Q
L	L	Q_n
H	L	H
L	H	L
H	H	Qを反転

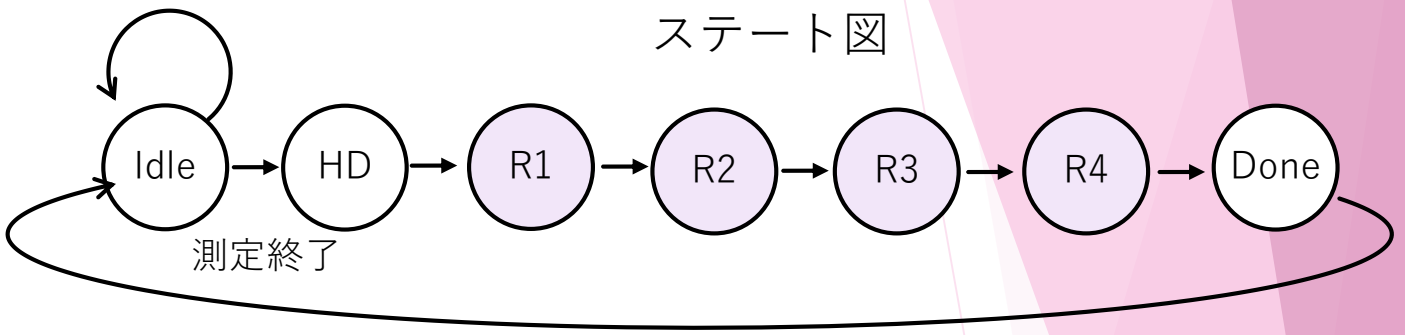


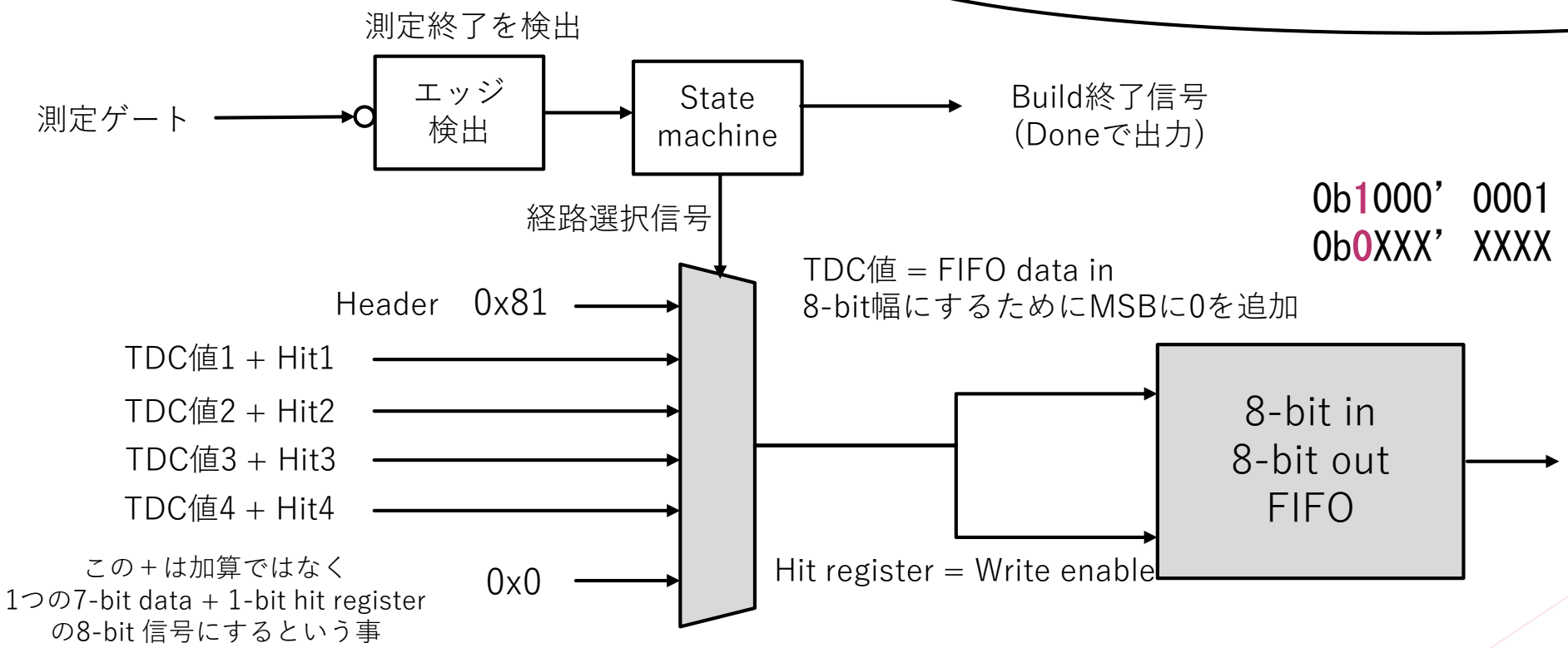
100 MHzのシステムクロックで同期

Stop Unit


Write enable = 1でカウンタ値を保持
→Startからの時間
カウンタ値
(Start blockから)
7
7
DFF
TDC値
Write enable
Stop入力
Double
FF
エッジ
検出
J
JKFF
ヒットレジスタ
ブロック外へ
測定ゲート
K
Build終了信号
これを4つ作る

Event builder



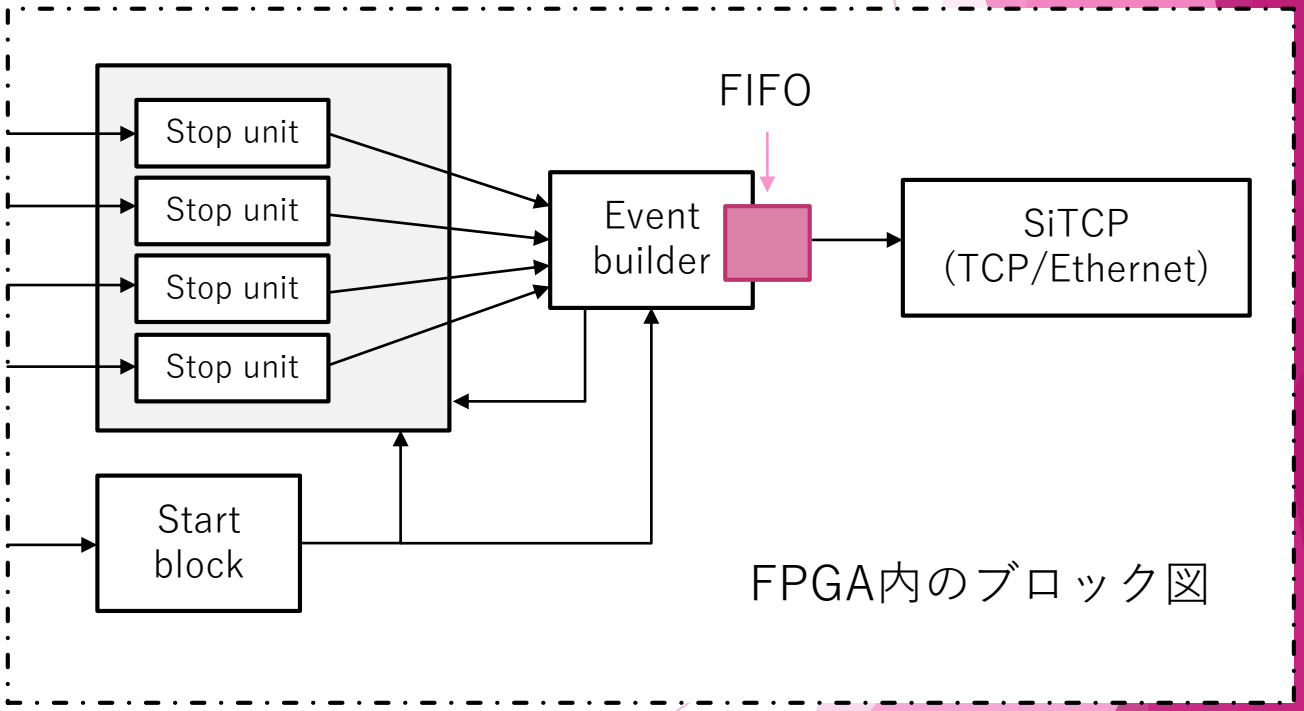


0b1000’ 0001 :Header
0b0XXX’ XXXX :TDC data

つなげる

右のようにつなげれば最低限要求を満たすものが出来る…はず

この設計では測定ゲートをBUSYの代わりとしている
しかし、それではちょっとBUSY解除が早くて危ない
どうして危ないのか？
→レポート問題



最後に

皆さん今後回路基板を設計する人、FPGA回路を開発する人、色々だと思います
私からのメッセージは

仕様を決めずに作り始めない！
仕様が決まっておらず色々機能を盛り込もうは失敗します

チップやFPGAのデータシートは隅々まで読み込みましょう！
最近の高級なICはうまく動く条件が事細かに決まっています

ソフトウェアとハードウェアは根本的に違う物であると認識しましょう！
タイミング、消費電力、電源プレーン…考えることは沢山あります

履歴

- 2020/7/29 v1.0 本多良太郎 (東北大)