

四則演算

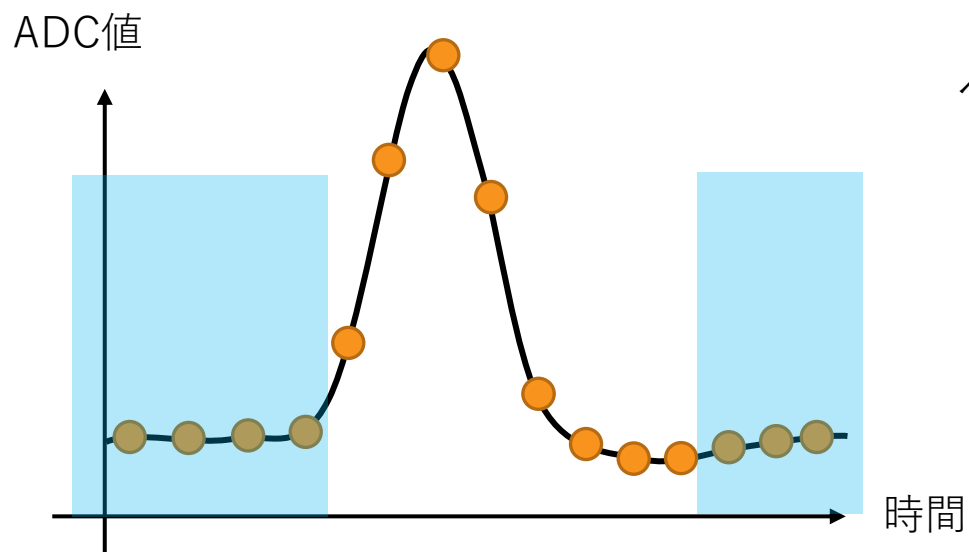
KEK IPNS E-sys
本多良太郎

四則演算が必要な例：ベースライン補正

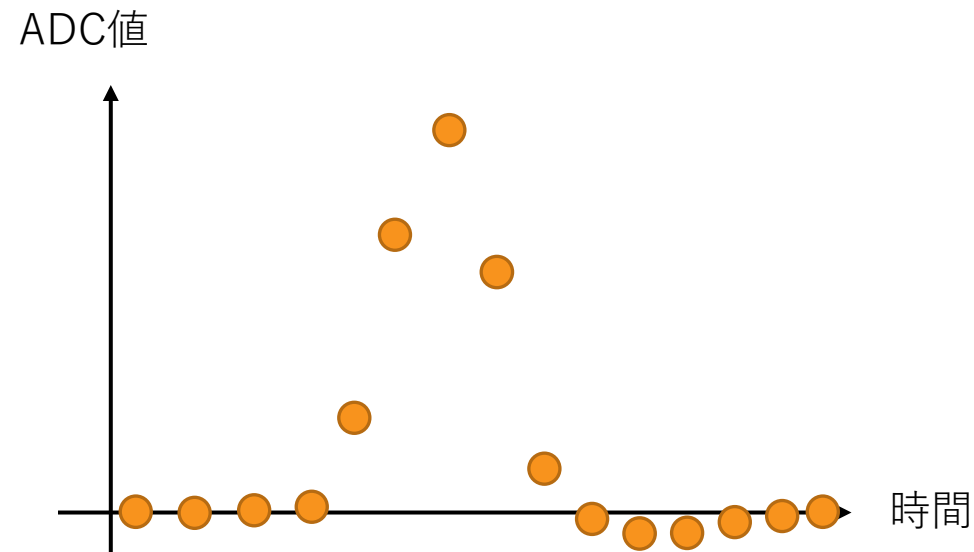
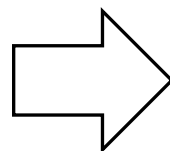
ADCで取得した波形からベースラインを求め
動的なベースライン補正を施す

必要な演算

- 加算（累算）
- 除算（平均導出）
- 減算



ベースライン
の引き算



FPGA内で四則演算が出来るとデータ処理の幅が広がる

二進数による数値表現

N, Mは正の整数

$$X = \sum_{n=0}^{N-1} x_n 2^n \quad (x_n: 0 \text{ or } 1)$$

符号なしN-bit整数

ではこのビット列を
10進数表記すると？

0b11011

13.5でした！

$$X = -1x_{N-1}2^{N-1} + \sum_{n=0}^{N-2} x_n 2^n$$

符号付きN-bit整数

$$X = -1x_{N-1}2^{N-1} + \sum_{n=-M}^{N-2} x_n 2^n$$

符号付きN-bit整数+M-bit小数
(固定小数点表記)

データ形式を定義しないと
10進の数字に直らない
(符号なし4-bit整数+1-bit小数)

デジタル回路での四則演算を考えると…

加算減算: 2の補数を使う事で加算器のみで実現可能

乗算除算: 最も単純な乗算器では符号 (2の補数) を取り扱えない

浮動小数点は今のところ取り扱う予定は無いです

2の補数のおさらい

補数: 元の数と補数を足した場合に**桁上がりが発生する最小の数**

減基数の補数: 元の数と補数を足した場合に**桁上がりが発生しない最大の数**

2進数の場合

- 補数: **2の補数**, 負の数表現するために使う
- 減基数の補数: **1の補数**, TCPやUDPでチェックサムを計算するときに使う (余談)

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

計算方法

1の補数: 元のビット列の否定 (not)をとる

2の補数: 1の補数に1を足す

左の表の型 (N=5)

$$X = -1x_N 2^{N-1} + \sum_{n=0}^{N-2} x_n 2^n \quad \text{符号付き5-bit整数}$$

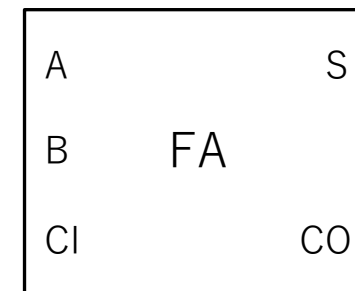
全加算器のおさらい

A, BおよびCarryInの3入力加算器（CarryInがないと半加算器）

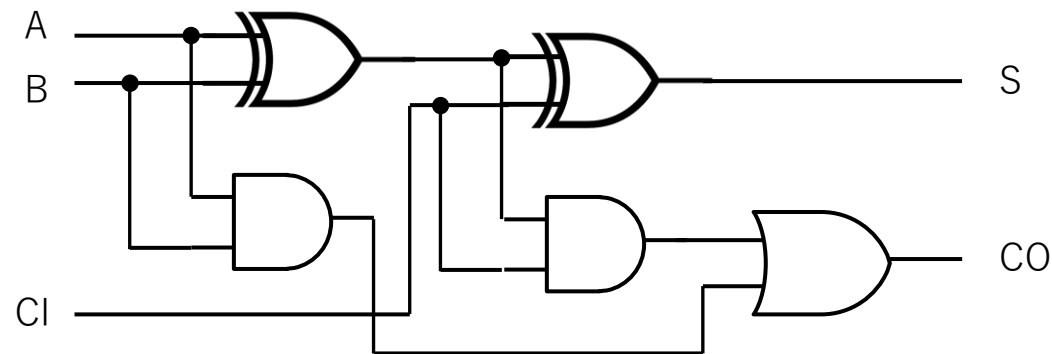
- **Carry bit:** 桁上りを通知するためのビット

真理値表を使って入出力関係が一意に決まる:

- **組み合わせ回路で実装できる** ⇔ CLBスライスのLUTかCARRY4を消費する



A	B	CI	S	CO
0	0	0	0	0
1	0	0	1	0
0	1	0	1	0
0	0	1	1	0
1	1	0	0	1
1	0	1	0	1
0	1	1	0	1
1	1	1	1	1



加算・減算の場合

同じ形式のデータ同士であれば気にせず加算してよい

例題1

0b10110 + 0b00011 = 0b11001

$$\begin{array}{r} 10110 \\ + 00011 \\ \hline 11001 \end{array}$$

このビット列の解釈をそれぞれ

- 符号なし5-bit整数
- 符号付き5-bit整数
- 符号付き3-bit整数+2-bit小数

の場合に対して10進数に変換し計算結果を確認しよう

N, Mは正の整数

$$X = \sum_{n=0}^{N-1} x_n 2^n \quad (x_n: 0 \text{ or } 1)$$

符号なしN-bit整数

$$X = -1x_{N-1}2^{N-1} + \sum_{n=0}^{N-2} x_n 2^n$$

符号付きN-bit整数

$$X = -1x_{N-1}2^{N-1} + \sum_{n=-M}^{N-2} x_n 2^n$$

符号付きN-bit整数
+M-bit小数
(固定小数点表記)

加算器のビット幅

例題2

$$0b10110 + 0b01011 = ?$$

加算器の計算結果のビット幅は入力と同じ
桁上がりした場合はどうなる？

- 桁上りを示すためにキャリービットが備わっている
- 全加算器には下位からのキャリー入力が備わっている
- HDLで書く際に+オペレータで済みます場合キャリーの事は通常頭に入れなくてよい

何を表現したいかに合わせビット幅を選択する

- 入力幅を拡張して計算結果が入力レンジを超えられないようにする
- 意図的にオーバーフローさせ周回するようにする

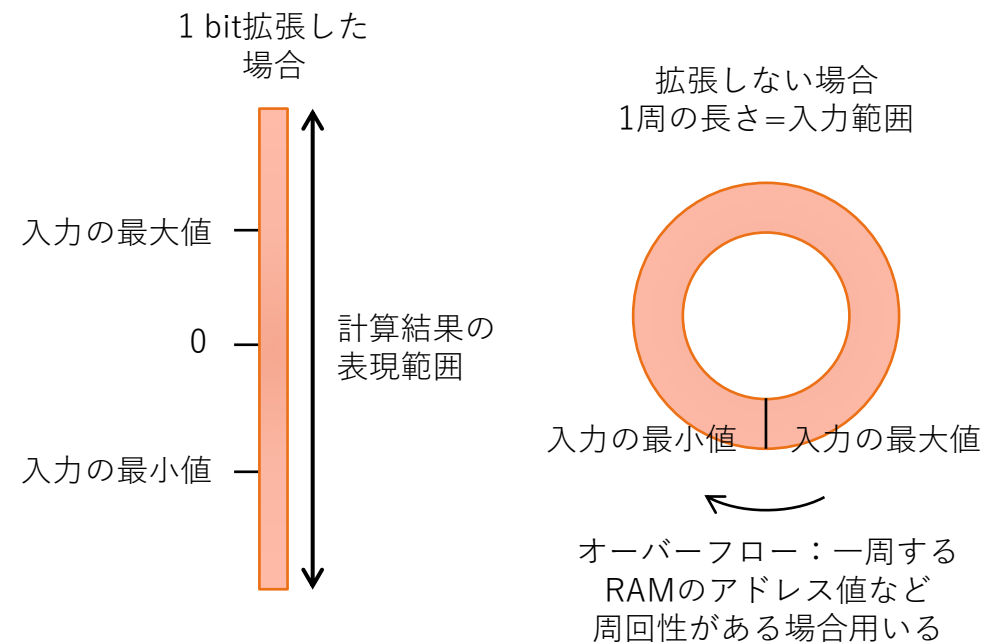
$$\begin{array}{r} 10110 \\ + 01011 \\ \hline 100001 \end{array}$$

キャリービット出力

桁上りを通知するビット

加算器の出力

HDLで書く場合も入力の
ビット幅と揃える



2進数の乗算

符号なしの場合

- 部分積を計算しその総和を求める

$$X \cdot Y = (\sum_{n=0}^{N-1} x_n 2^n) (\sum_{m=0}^{M-1} y_m 2^m) = \sum_n (\underbrace{\sum_m x_n y_m 2^{n+m}}_{\text{部分積}})$$

符号付きの場合

- 符号項があり符号なし乗算器の単純な拡張にはならない

$$X \cdot Y = (-1x_{N-1} + \sum_{n=0}^{N-2} x_n 2^n) (-1y_{M-1} + \sum_{m=0}^{M-2} y_m 2^m)$$

ここでは符号なしの場合の最も単純な乗算器について説明する
符号も取り扱いたいなら絶対値を抽出し符号は別処理、もしくはDSPを使う。

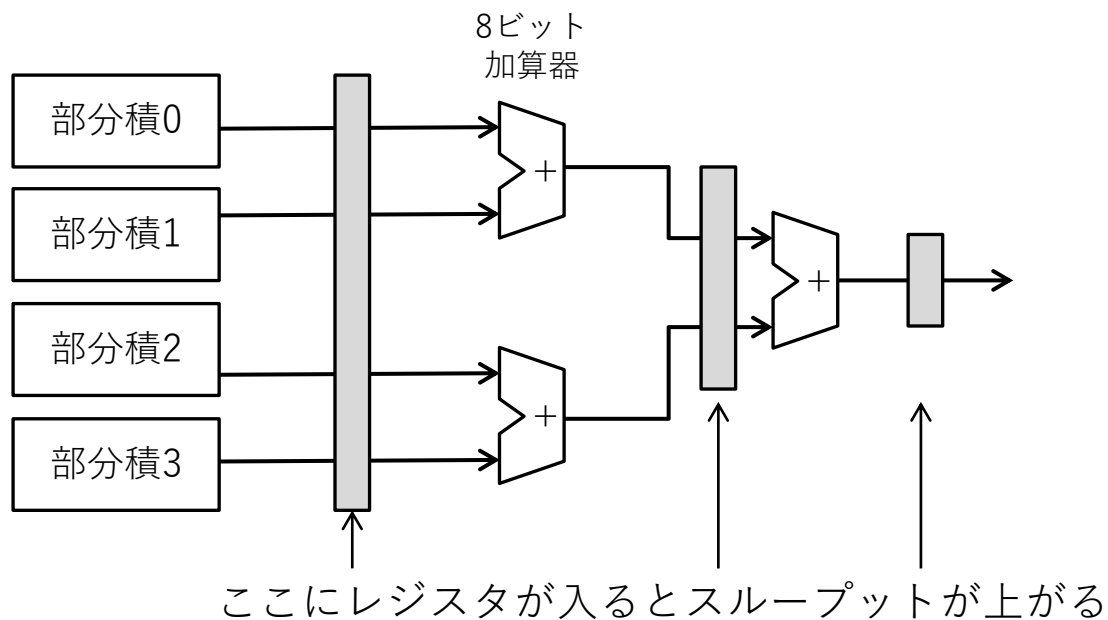
2進数の乗算

符号なし整数の場合

- 部分積を計算しその総和を求める

$$X \cdot Y = (\sum_{n=0}^{N-1} x_n 2^n) (\sum_{m=0}^{M-1} y_m 2^m) = \sum_n (\sum_m x_n y_m 2^{n+m})$$

部分積



1011	
x 0101	

1011	y0の部分積
0000	y1の部分積
1011	y2の部分積
0000	y3の部分積

00110111	結果のビット幅は N+M (この場合2N)

入力のビット数だけ部分積の計算が必要

より演算量を減らすには
ブースのアルゴリズム
(この講義では取り扱わない)

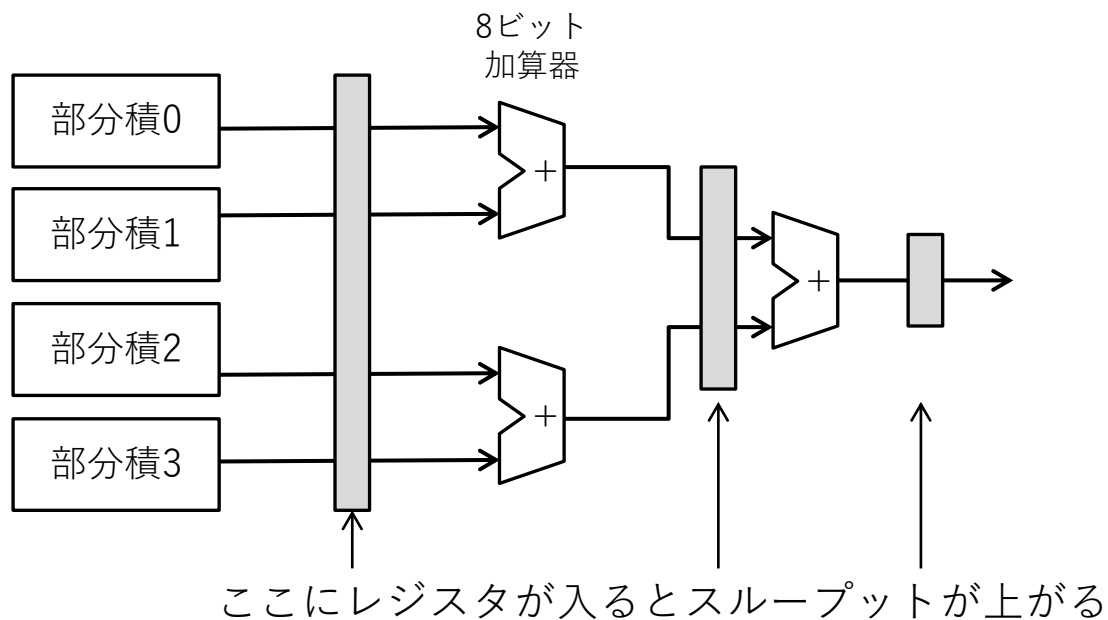
2進数の乗算

符号なし固定小数の場合

- 部分積を計算しその総和を求める

$$X \cdot Y = (\sum_{n=-A}^{N-1} x_n 2^n) (\sum_{m=-B}^{M-1} y_m 2^m) = \sum_n (\sum_m x_n y_m 2^{n+m})$$

部分積



小数部がある場合
小数点はどこに来るか？

$$\begin{array}{r} 1011 \\ \times 0101 \\ \hline 1011 \quad y0 \text{の部分積} \\ 0000 \quad y1 \text{の部分積} \\ 1011 \quad y2 \text{の部分積} \\ 0000 \quad y3 \text{の部分積} \\ \hline 00110111 \end{array}$$

整数部 小数部

計算結果のビット幅は
小数部は小数部の
整数部は整数部のビット数の和となる

乗算ができれば除算もできる

除数を取る範囲があらかじめ分かっている場合

- 必要な精度までの固定小数点表記のビットテーブルで除数を表現
- 小数表記の除数を被除数に乗算

例: 符号なし4ビット整数範囲 (1-15, 0除算は除外)

小数ビット幅の選択

- 3など割り切れない除数の場合4ビットでは全然精度が足りない
- 8ビットでも10進の小数点第2位が何とか正しい程度

FPGAへのテーブルの実装

- 除数のパターンが少ない: LUT (組み合わせ回路) で実装
- 除数のパターンが多い: メモリブロックを使用しROMを形成
 - RAMのアドレスを除数, 値を小数表記の除数と考える

整数以外の除数に対してもテーブルは作れるが大変

符号なし4-bit整数, 4-bit固定小数
の除数テーブル

除数	小数表記
1	0b0001'0000
2	0b0000'1000
3	0b0000'0101
4	0b0000'0100
...	
15	0b0000'0001

乗算器を使わない除算

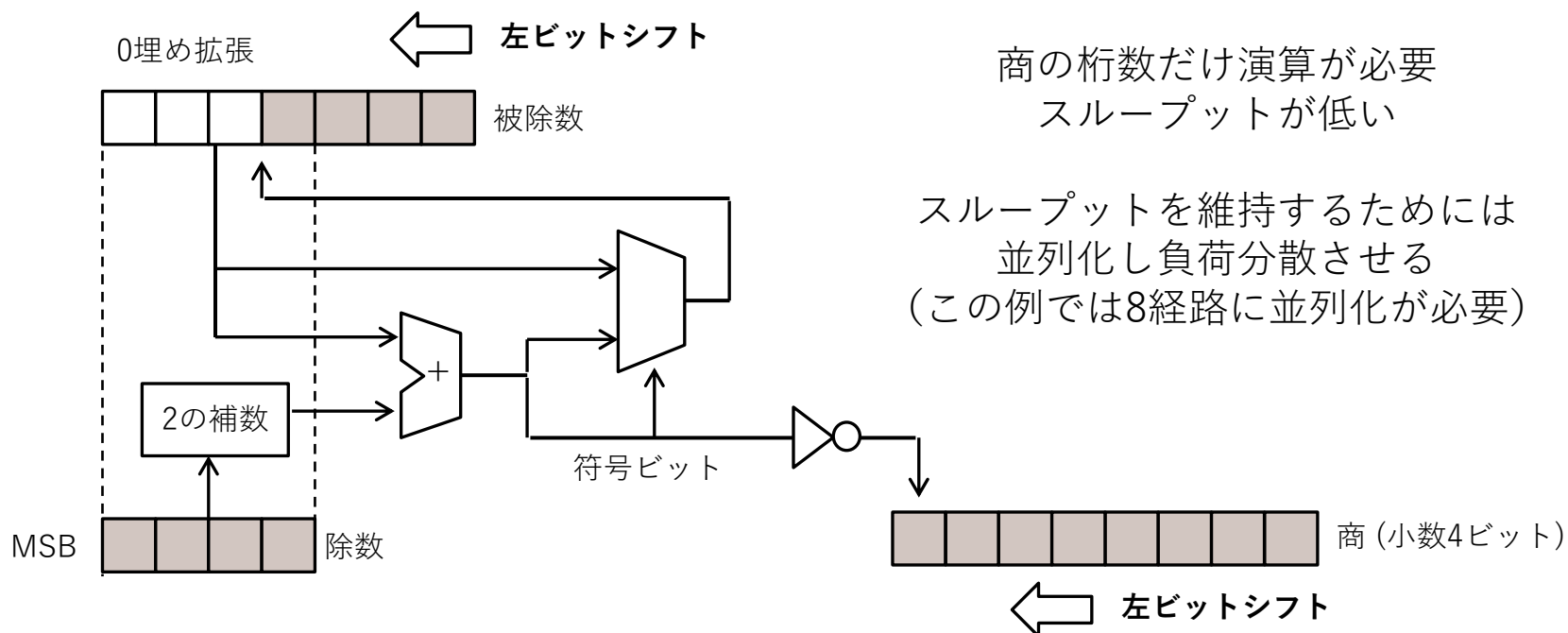
2進数の割り算をひっ算で考えてみる

- 被除数の上位ビットから減算を試み結果が
 - 負の数であれば何もせず左ビットシフトし次の桁へ
 - 正の数であれば減算結果を頭につけ左ビットシフトし次の桁へ
 - 符号が商のN桁目の0/1を決める
- 必要な精度までこの演算を繰り返す
この手法を回復法による除算と呼ぶ

商の各桁は0か1

```

      00110...
0011 ) 1010
      11
      --
       10
       11
       --
        101
         11
         --
          100
           11
           --
            10
  
```



汎用から専用へ

今まで説明した四則演算回路はすべてFPGAのCLBスライスを消費して合成される

大規模な演算には
専用スライスの**digital signal processor (DSP)** を活用しよう
Xilinx 7-seriesであればDSP48E1プリミティブを利用する

表 2-1 : 7 シリーズの各デバイスの DSP48E1 スライス数

デバイス	デバイスあたりの DSP48E1 スライス数	デバイスあたりの DSP48E1 カラム数	カラムあたりの DSP48E1 スライス数
7A15T	45	2	60 ⁽²⁾
7A35T	90	2	60 ⁽²⁾
7A50T	120	2	60
7A75T	180	3	80 ⁽²⁾
7A100T	240	3	80
7A200T	740	9	100 ⁽¹⁾
7K70T	240	3	80
7K160T	600	6	100
7K325T	840	6	140
7K355T	1,440	12	120
7K410T	1,540	11	140
7K420T	1,680	12	160 ⁽²⁾
7K480T	1,920	12	160
7V585T	1,260	7	180
7V2000T	2,160	9	240 ⁽³⁾
7VX330T	1,120	8	140

DSP48E1の概略

4つの入力に対し加算, 累算, 乗算などが演算可能

以下は簡略化した図であり実際にはもっと複雑

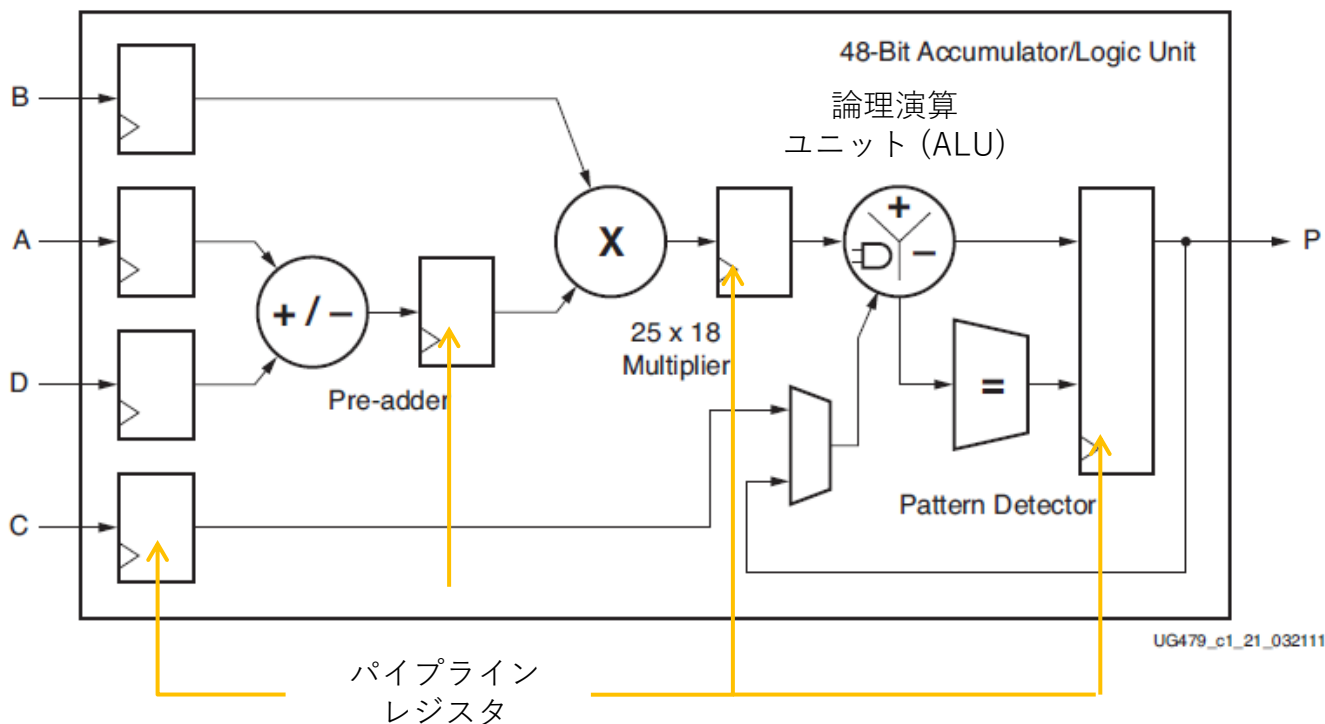
- 完全に使いこなすにはUG479を読み込む必要あり

実装できる演算の例

- $(A+D)*B+C$
- $A+D+C$
- $A+P$ (累算)

Pattern detector

- オーバフロー検出
- カウンタ自動リセット (周期パルス)



DSPの入力オペランドは符号付き (2の補数)!
HDL側で符号拡張していることを推奨

どうやって使うか

1. IP catalogからIPを生成し実装する

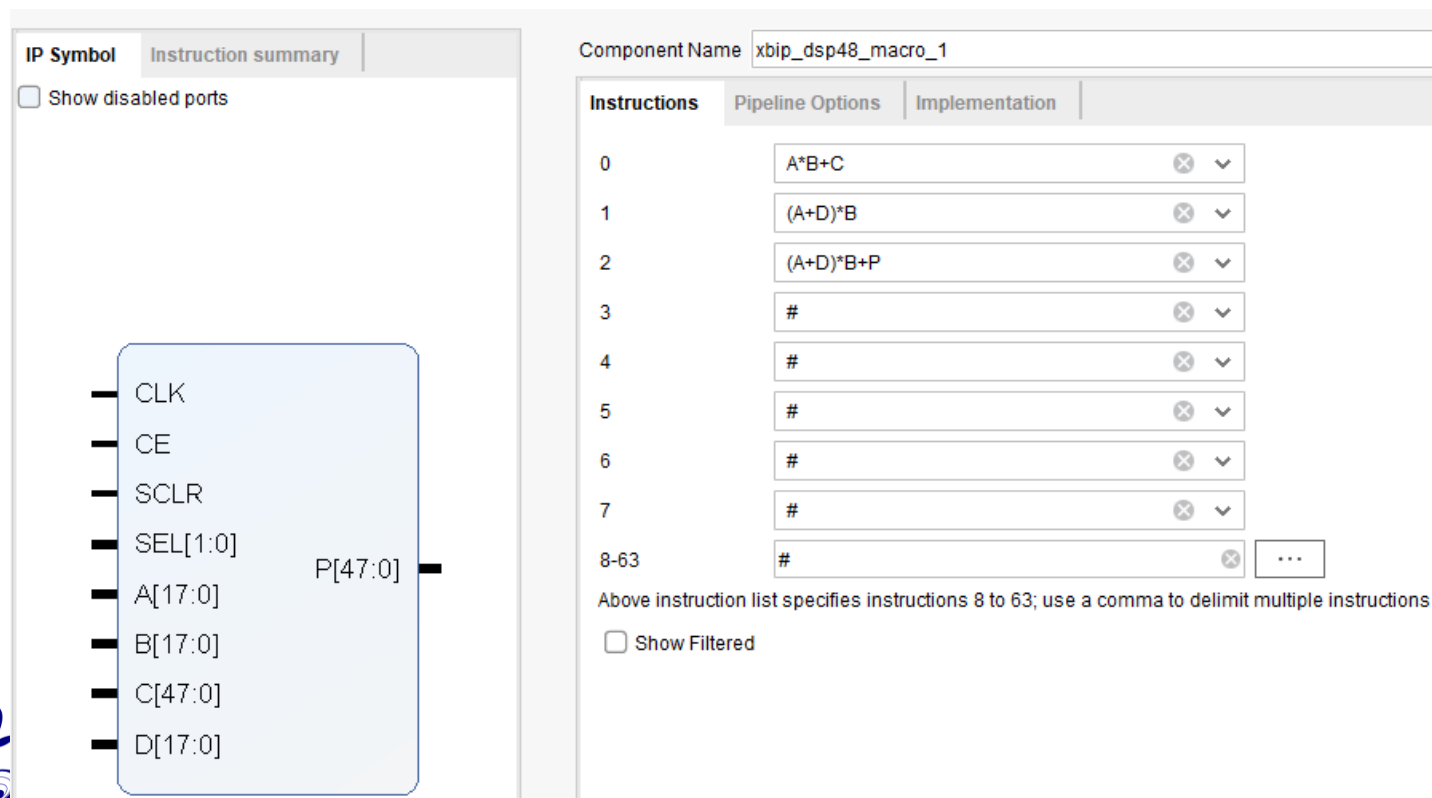
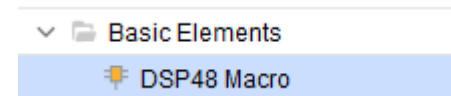
- こちらをこの演習では推奨

2. Language templateを参考にしHDL上で手動でinstantiateする

- DSPに限らずXilinx FPGAのプリミティブは大体HDLから直に実装できる
- 上級者向け, スライスの機能を完全に理解してないといけない

3. HDLコードからDSPを推論させる

- コードにベンダ依存性を持ち込みたくない人向け



The screenshot shows the Xilinx IP configuration tool for the 'xbip_dsp48_macro_1' component. On the left, the 'IP Symbol' tab displays a block diagram with inputs: CLK, CE, SCLR, SEL[1:0], A[17:0], B[17:0], C[47:0], and D[17:0], and an output: P[47:0]. On the right, the 'Instruction summary' tab is active, showing a table of instructions. The 'Component Name' is 'xbip_dsp48_macro_1'. The table has columns for 'Instructions', 'Pipeline Options', and 'Implementation'. The instructions are: 0: A*B+C, 1: (A+D)*B, 2: (A+D)*B+P, 3: #, 4: #, 5: #, 6: #, 7: #, and 8-63: #. Below the table, a note states: 'Above instruction list specifies instructions 8 to 63; use a comma to delimit multiple instructions'. There is also a 'Show Filtered' checkbox.

Instructions	Pipeline Options	Implementation
0	A*B+C	⊗ ▼
1	(A+D)*B	⊗ ▼
2	(A+D)*B+P	⊗ ▼
3	#	⊗ ▼
4	#	⊗ ▼
5	#	⊗ ▼
6	#	⊗ ▼
7	#	⊗ ▼
8-63	#	⊗ ...

Above instruction list specifies instructions 8 to 63; use a comma to delimit multiple instructions

☐ Show Filtered

SEL: Instruction切り替え

- この例では3つのinstructionを定義したのでSELの有効範囲は0-2

1. IP catalogからIPを生成し実装する

- こちらをこの演習では推奨

Component Name

Instructions Pipeline Options Implementation

Pipeline Options

Custom Pipeline options

Tier:	1	2	3	4	5	6
D	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
A	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
B	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
CONCAT	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
C	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
CARRYIN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SEL	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
KEY:	Fabric register					
	DSP register					

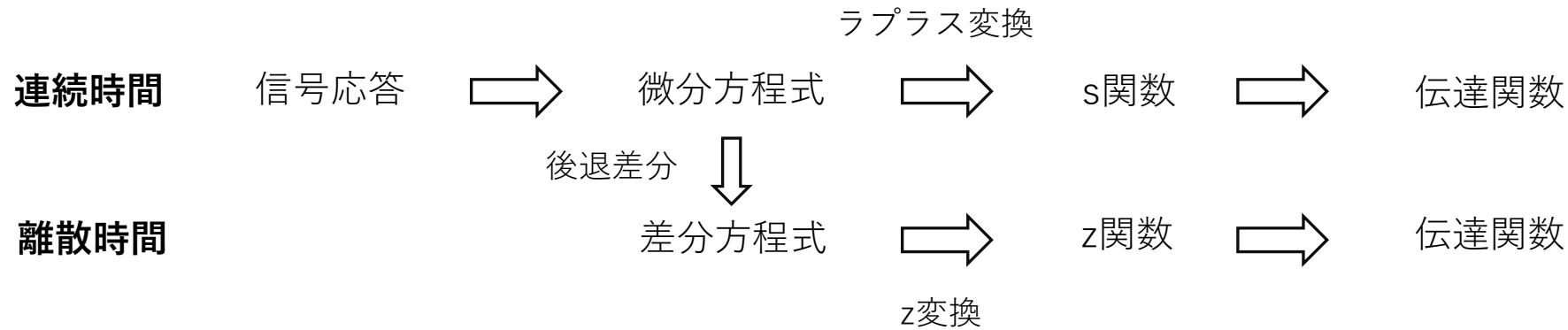
Control ports

	Global	D	A	B	CONCAT	C	M	P	SEL/CARRYIN
CE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SCLR	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Pipeline optionsでONに出来るポート

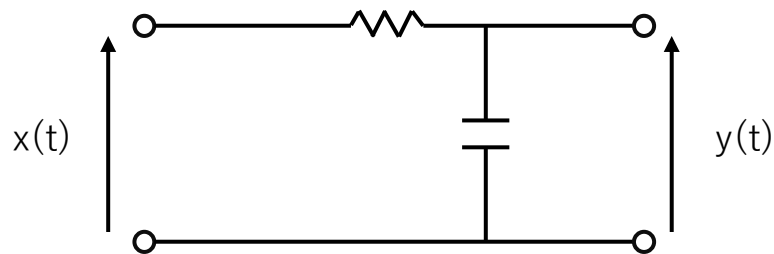
- CE: clock enable
- SCLR: 同期リセット
 - これが無いと累算の結果を動的に0に出来ない

DSPによる デジタルフィルタの実装

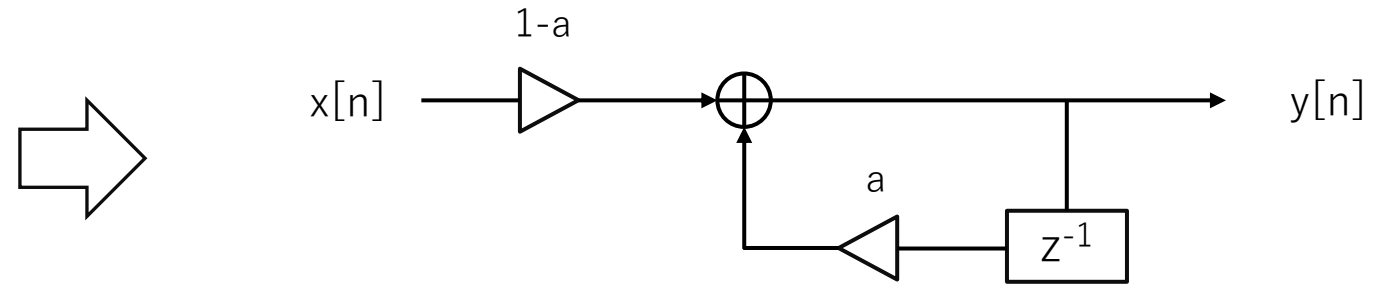


物理学科ではやったとしてもs関数まで
それ以上は自習が必要

アナログフィルタ



デジタルフィルタ



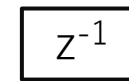
加算(器)



乗算(器)



遅延

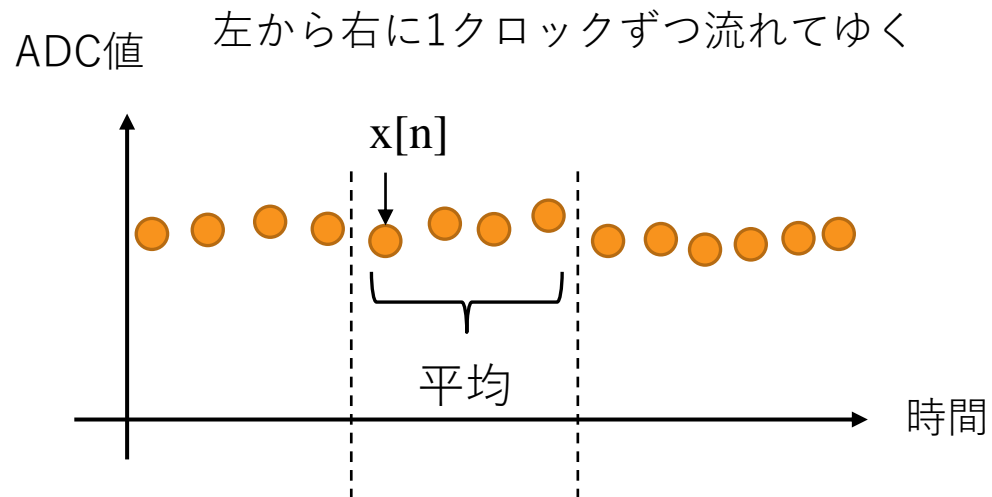


乗数は上か下に
書くとする

肩の数字のサンプリング時間分戻る
これは1サンプリング戻る

- はじめて学ぶデジタル・フィルタと高速フーリエ変換 (三上直樹 著, CQ出版)
- デジタル・フィルタ 理論&設計入門 (三谷政昭 著, CQ出版)
- デジタルフィルタ原理と設計法 (設計技術シリーズ) (陶山 健仁 著, 科学情報出版)

波形データ



n番目の離散時間信号: $x[n]$

N点を使った移動平均フィルタの差分方程式は

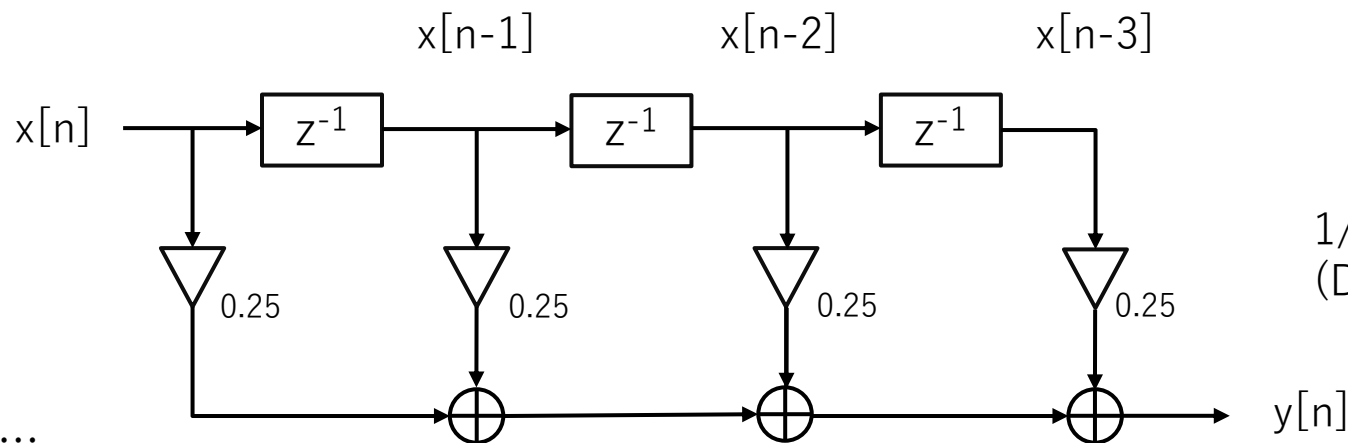
$$y[n] = \sum_{i=0}^{N-1} ax[n-i] \quad a = \frac{1}{N}$$

次々と流れていく離散信号の平均を取るのは
移動平均フィルタを実装するのと同じである

移動平均フィルタは標本平均の式と差分方程式
が一致するので直感的にわかりやすい

DSPを使って移動平均フィルタを
実装してみる

4点の移動平均を取るフィルタを考える。
n番目のサンプリングから3点戻ればよいので…



1/4を掛けよう
(DSPとの相性を考え毎度かける)

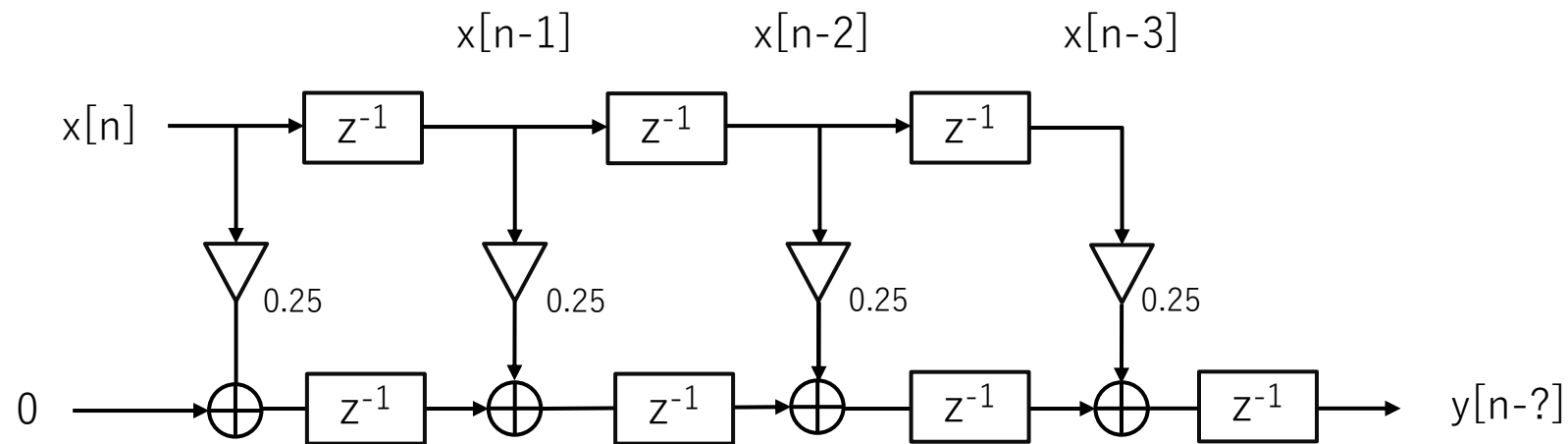
加算器は2入力なので…
ここまではよい。
次の加算器はどこに置くか？

横につなげるのが一般性が
あってよい

完成
これが普通教書に出てくる形
しかし…
加算器の伝搬遅延が積分される
FPGAに実装できるだろうか？

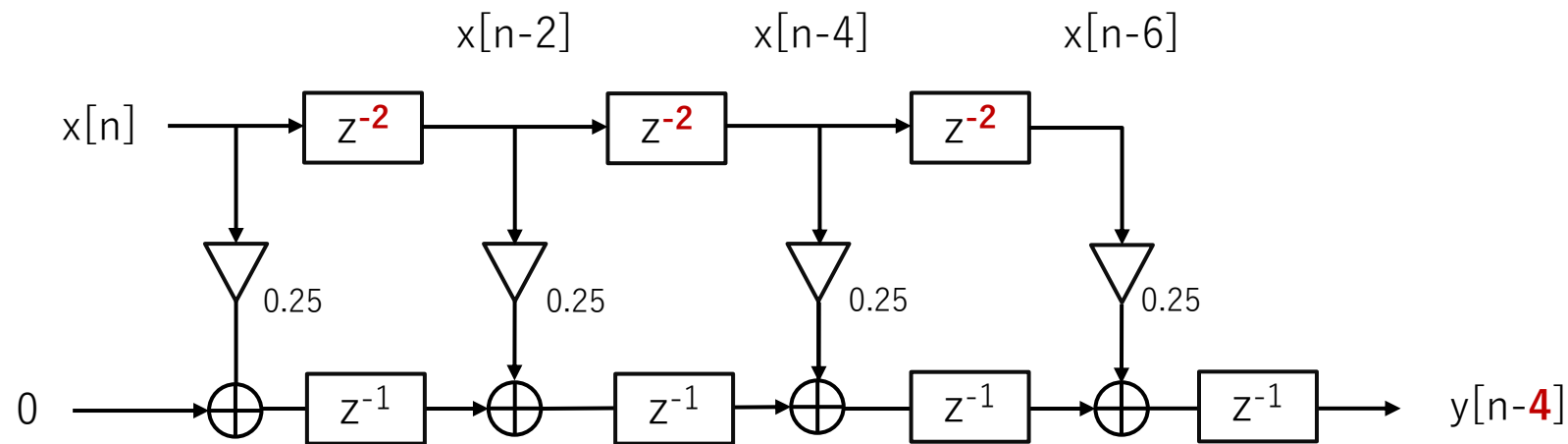
各加算器の出力にレジスタを置こう。

1項目は例外ではないので、最も左にも加算器が隠れている事を忘れないように。



レジスタが入ると足される数
に対する遅延量が足らなく
なった…

加算器の出力に入れたパイプラインレジスタ分
入力も遅らせないとイケない



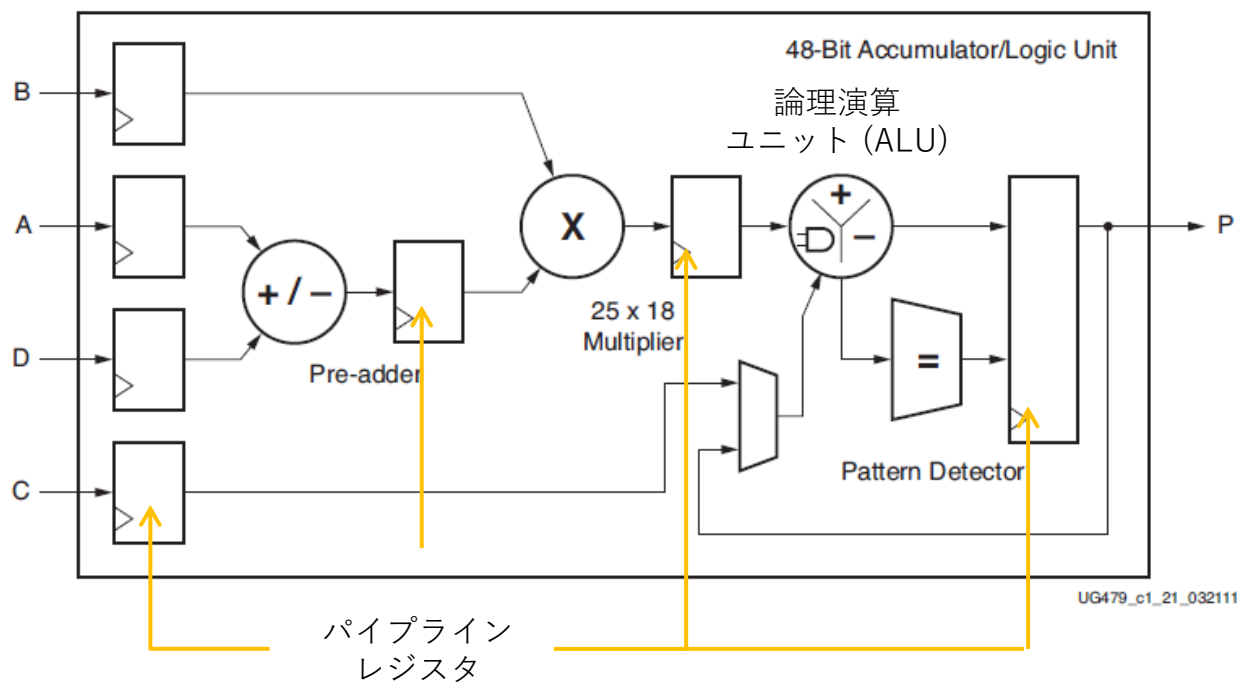
完成？

出力の遅延量は最短経路に入っている遅延素子の数で決まる

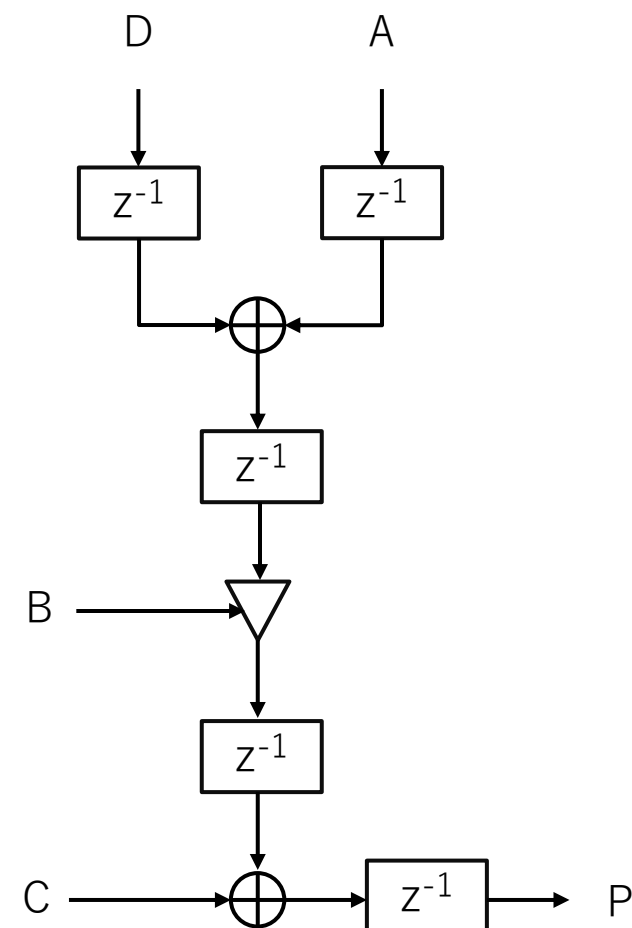
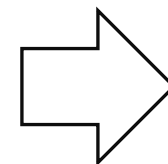
ところで乗算器にはレジスタは
要らない？

再びXilinx FPGAのDSPの構造

- 2つの加算器、1つの乗算器、4つのレジスタが存在する

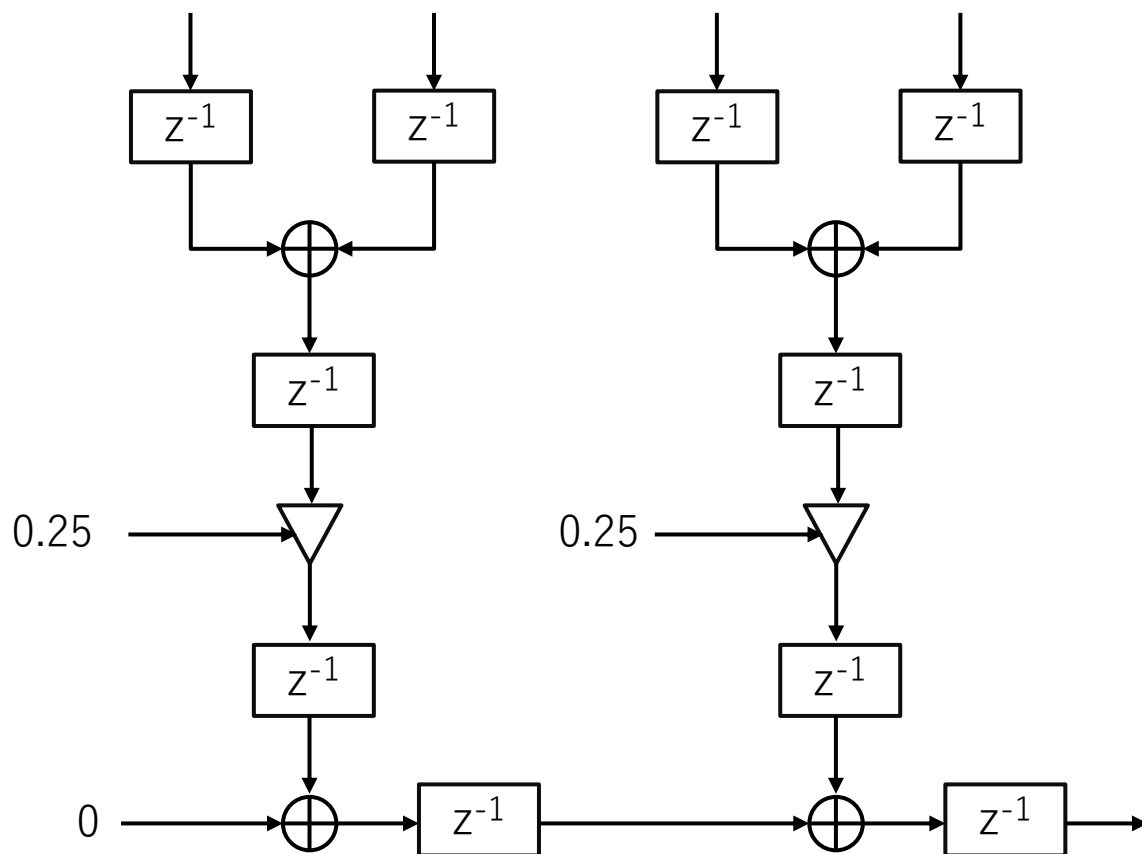


前頁のような図にしてみる

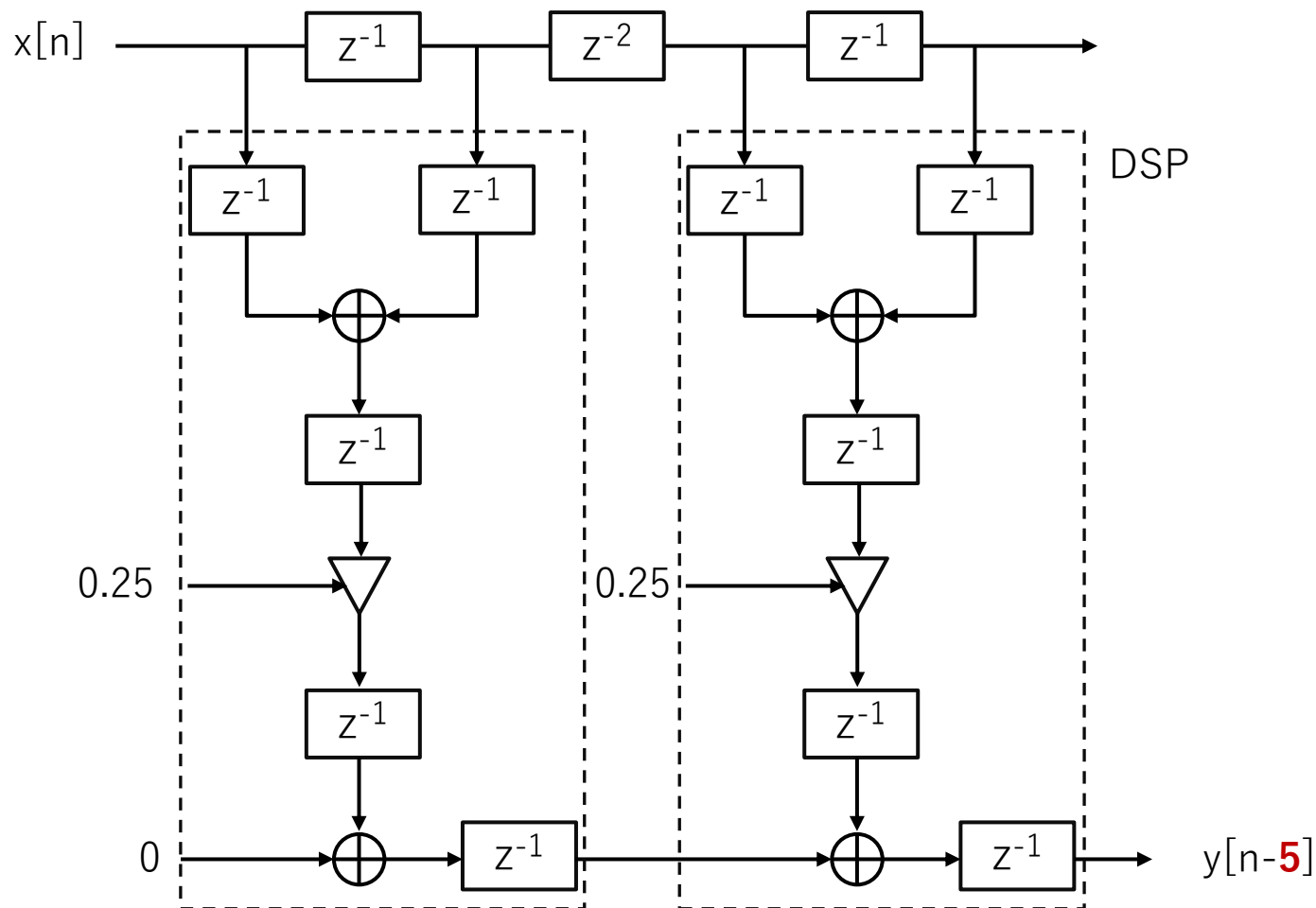


加算が入れ子になることがわかる

必要なパイプレジスタと入力の遅延を決める



必要なパイプラインレジスタと入力の遅延を決める



完成

これを実装してみましょう

条件

符号付き固定小数点
整数部: 8-bit
小数部: 2-bit

詳しくは例題手順書
EX1を参照

DSPをIPカタログから生成する

IP Symbol | **Instruction summary**

☐ Show disabled ports

CLK

A[9:0]

B[9:0]

C[47:0]

D[9:0]

P[47:0]

Component Name `xbip_dsp48_macro_0`

Instructions | Pipeline Options | Implementation

0	<input type="text" value="(A+D)*B+C"/>	<input type="button" value="✕"/> <input type="button" value="▼"/>
1	<input type="text" value="#"/>	<input type="button" value="✕"/> <input type="button" value="▼"/>
2	<input type="text" value="#"/>	<input type="button" value="✕"/> <input type="button" value="▼"/>
3	<input type="text" value="#"/>	<input type="button" value="✕"/> <input type="button" value="▼"/>
4	<input type="text" value="#"/>	<input type="button" value="✕"/> <input type="button" value="▼"/>
5	<input type="text" value="#"/>	<input type="button" value="✕"/> <input type="button" value="▼"/>
6	<input type="text" value="#"/>	<input type="button" value="✕"/> <input type="button" value="▼"/>
7	<input type="text" value="#"/>	<input type="button" value="✕"/> <input type="button" value="▼"/>
8-63	<input type="text" value="#"/>	<input type="button" value="✕"/> <input type="button" value="⋮"/>

Above instruction list specifies instructions 8 to 63; use a comma to delimit multiple instructions

☐ Show Filtered

DSPをIPカタログから生成する

Component Name

Instructions Pipeline Options Implementation

Pipeline Options

Custom Pipeline options

Tier:	1	2	3	4	5	6
	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
D	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input checked="" type="checkbox"/>	
A	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input checked="" type="checkbox"/>	
B	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input checked="" type="checkbox"/>	
CONCAT	→			<input type="checkbox"/>		
C	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input type="checkbox"/>	<input checked="" type="checkbox"/>
CARRYIN	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input type="checkbox"/>	
SEL	<input type="checkbox"/>	→	<input type="checkbox"/>	→	<input type="checkbox"/>	
KEY:	Fabric register					
	DSP register					

Control ports

	Global	D	A	B	CONCAT	C	M	P	SEL/CARRYIN
CE	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SCLR	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

先ほどの図の通り実装するので
C入力へのレジスタは省略

動きましたか？
移動平均が1ずつ増えていくはずですが
入力と出力間のレイテンシは5になったでしょうか

DSPは2の補数表記でオペランドを受け取るので移動平均の
結果が負の値になっているときがあると思います。
その動きも確認してみましょう。