# Arithmetic operations

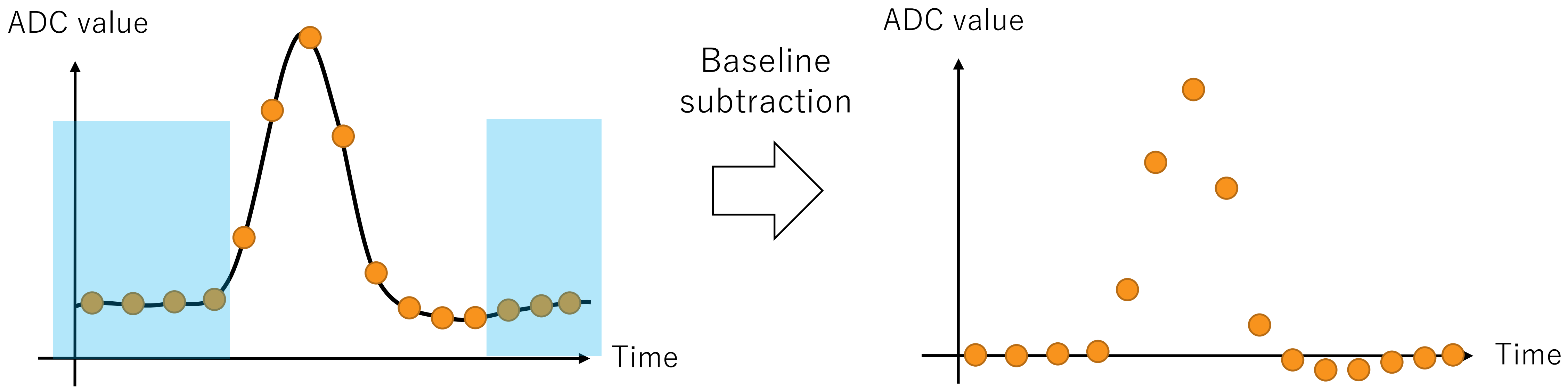
KEK IPNS E-sys
Ryotaro Honda, Yun-Tsung Lai

## An example which requires arithmetic operations: Baseline correction

Calculate the baseline from the waveform acquired by the ADC and perform dynamic baseline correction

**Necessary operations**

- Addition (accumulation)
- Division (calculating mean)
- subtraction



**If arithmetic operations can be performed in FPGA, the range of data processing will expand**

## Numeric representation in binary

**N, M: positive integer**

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

**unsigned N-bit integer**

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

**signed N-bit integer**

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

**Signed N-bit integer + M-bit decimal**
**(fixed-point notation)**

If this bit string is expressed in decimal notation?
**0b11011**

It's **13.5**!

If don't define the data format, you can't fix it as a decimal number
(unsigned 4-bit integer + 1-bit decimal)

Considering the arithmetic operations in a digital circuit...
Addition and Subtraction: Can be realized with only an adder by using 2's complement
Multiplication and Division: The simplest multipliers can't handle signs (Two's Complement)
Currently no plans to support floating point

## Review on 2's complement

Complement: **the smallest number causing a carry** when the original number and the complement are added
Diminished radix complement: **the largest number that does not cause carry** when added to the original number and complement

In case of binary

- complement: **2's complement**, used to represent negative numbers
- Diminished radix complement: **1's complement**, used when calculating checksums with TCP or UDP (digression)

| decimal | binary | 2's complement |
|---|---|---|
| 0 | 0b00000 | |
| 1 | 0b00001 | 0b11111 |
| 2 | 0b00010 | 0b11110 |
| 3 | 0b00011 | 0b11101 |
| 4 | 0b00100 | 0b11100 |
| 5 | 0b00101 | 0b11011 |
| 6 | 0b00110 | 0b11010 |
| 7 | 0b00111 | 0b11001 |

**How to calculate**
1's complement: take the negation (not) of the original bit string
2's complement: add 1 to 1's complement

**For the left table (N=5)**

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

**signed 5-bit integer**

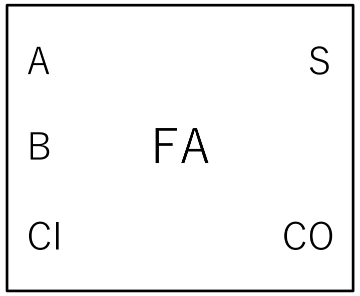
## Review on full adder

A 3-input adder with A, B and CarryIn (a half adder without CarryIn)

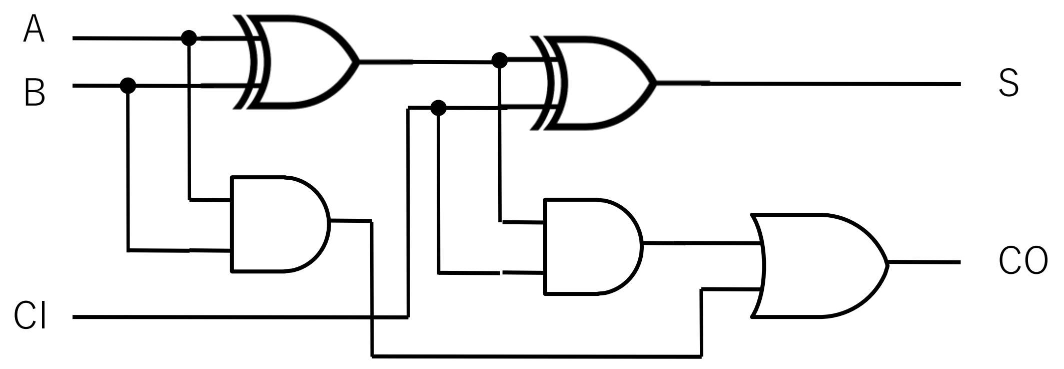
- **Carry bit**: A bit to inform the carry

A truth table is used to uniquely determine the I/O relation:

- **It can be implemented as a combinational circuit**
  **⇔ Consumes LUT or CARRY4 of CLB slice**



| 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** |



## The case of addition and subtraction

If the data have the same format, nothing to worry.

**EX1**
0b10110 + 0b00011 = 0b11001

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

Each interpretation of this bit string is
- Unsigned 5-bit integer
- Signed 5-bit integer
- Signed 3-bit integer + 2-bit decimal

Let's convert to decimal number for these cases and check the result or calculation.

**N, M: positive integet**

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

**Unsigned N-bit integer**

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

**Signed N-bit integer**

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

**Signed N-bit integer + M-bit decimal (fixed-point notation)**

## Adder’s bit width

```
      10110
+     01011
     100001
```

**carry bit output**
A bit to indicate carry

**Output of adder**
Align with the input bit width even if using HDL

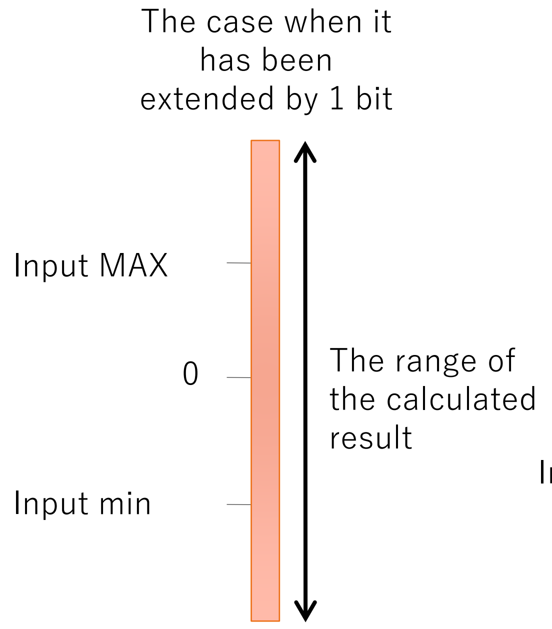
**EX2**
0b10110 + 0b01011 = ?

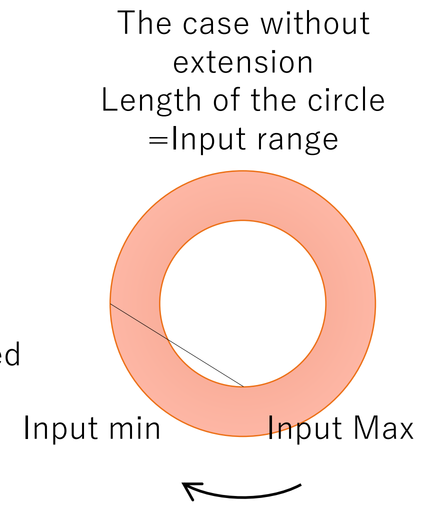
What happens if the bit width of the calculated result of the adder carries the same as the input?

- Carry bit is provided to indicate carry
- Full adder has carry input from lower order
- When using the + operator in HDL, usually no need to keep the carry in mind.

Select the bit width based on what you’d like to express

- Extend the input width to prevent the calculated result from exceeding the input range
- Intentionally overflow and circulate


The case when it has been extended by 1 bit
Input MAX
0
Input min
The range of the calculated result



Overflow: go beyond the RAM address value, etc.
It is used when there is periodicity.

## Multiplication for binary

The case of unsigned

- Calculate the partial products and get the sum

$$X \cdot Y = \left(\sum_{n=0}^{N-1} x_n 2^n\right)\left(\sum_{m=0}^{M-1} y_m 2^m\right) = \sum_n \underbrace{\left(\sum_m x_n y_m 2^{n+m}\right)}_{\textbf{Partial sum}}$$

The case of signed

- It has a signed term and it is not just a simple extension of unsigned multipliers.

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

Here we will describe the simplest multiplier for the case of unsigned.
If you want to handle the sign as well, extract the absolute value and use a separate process or DSP for the sign.

## Multiplication for binary

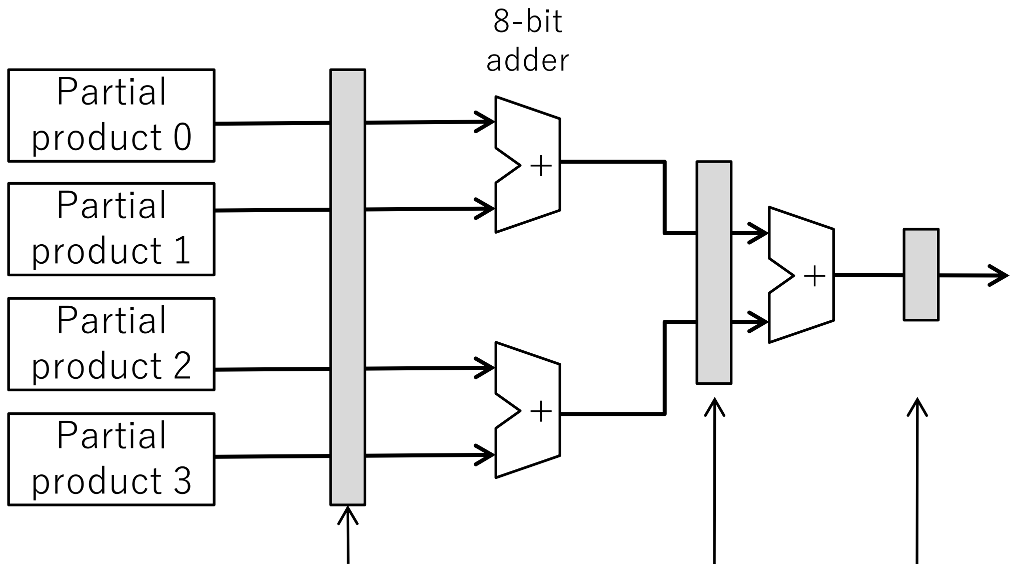
The case of unsigned

- Calculate the partial products and get the sum

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

```
     1011
  x  0101
  -------
     1011      y0's partial product
    0000       y1's partial product
   1011        y2's partial product
  0000         y3's partial product
  -------
  00110111
```

The bit width of the output is N+M (2N in this case)



Putting a register here will increase the throughput

Partial products are needed for the number of bits in the input

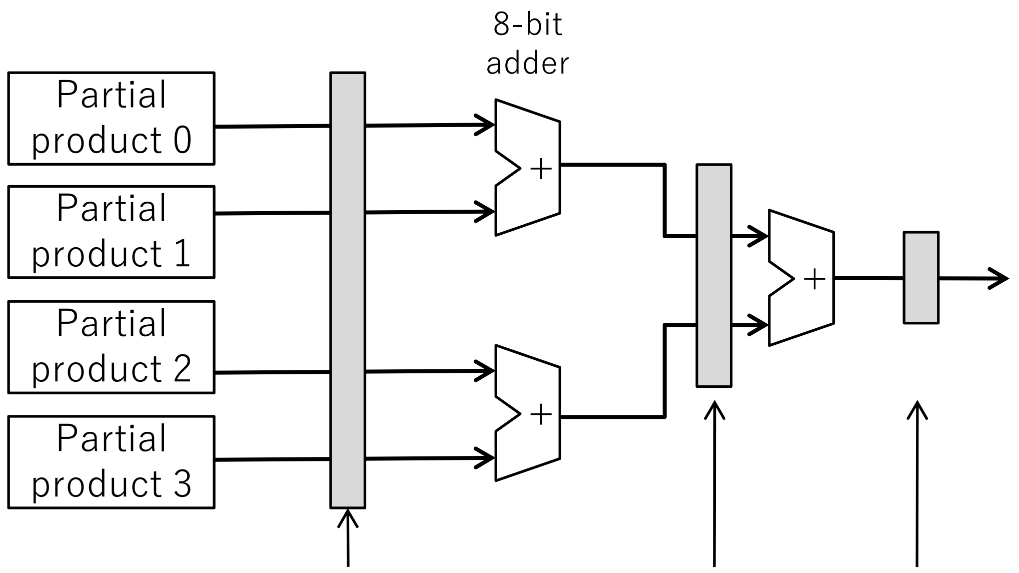
Use Booth's algorithm to reduce computation (not in this lecture)

## Multiplication for binary

The case of unsigned fixed point

- Calculate the partial products and get the sum

$$X \cdot Y = \left(\sum_{n=-A}^{N-1} x_n 2^n\right)\left(\sum_{m=-B}^{M-1} y_m 2^m\right) = \underbrace{\sum_n\left(\sum_m x_n y_m 2^{n+m}\right)}_{\textbf{Partial sum}}$$



Putting a register here will increase the throughput

If there is a decimal part, where is the decimal point?

```
      1011      Decimal part
    x 0101
    ------
      1011      y0's partial product
     0000       y1's partial product
    1011        y2's partial product
   0000         y3's partial product
   --------
   00110111
Integer part   Decimal part
```

The bit width of the output:
The decimal part Is the sum of the number of bits in the decimal part
The integer part is the sum of the number of bits in the integer part.

# If we can multiply, we can also divide.

**When the range covered by the divisor is known in advance**

- Represent the divisor as a bit table in fixed-point notation up to the desired precision
- Multiply the dividend by decimal divisor

**EX**: unsigned 4-bit integer's range (1-15, excluding 0 for division)

**Selection of decimal bit width**

- For divisors that are not divisible (such as 3) 4 bits don't have enough precision.
- Even with 8 bits, the second decimal place is somehow correct.

**Implementing the table in FPGA**

- Few divisor patterns：Implement with LUT(Combinational circuit)
- Many divisor patterns：Use ROM with memory blocks
    - Consider the RAM address as a divisor and the value as a decimal divisor

For the non-integer divisors, tables can be created, but difficult.

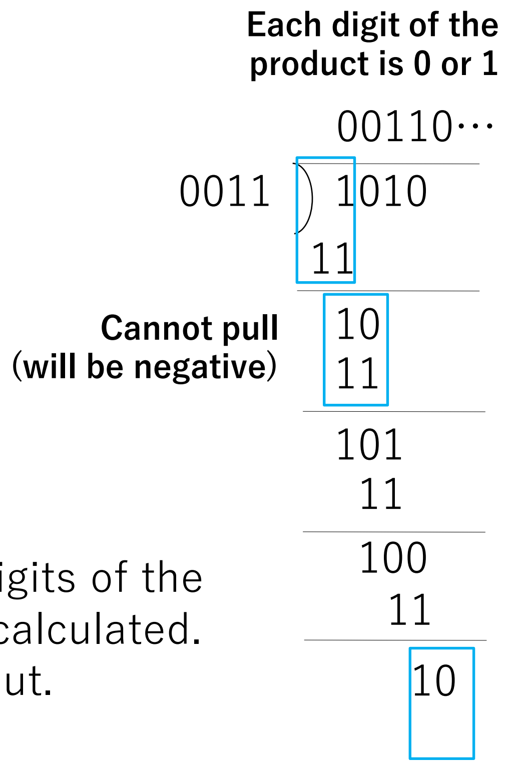
The divisor table for unsigned 4-bit integer, 4-bit fixed decimal

| Divisor | Decimal notation |
|---|---|
| 1 | 0b0001'0000 |
| 2 | 0b0000'1000 |
| 3 | 0b0000'0101 |
| 4 | 0b0000'0100 |
| … | |
| 15 | 0b0000'0001 |

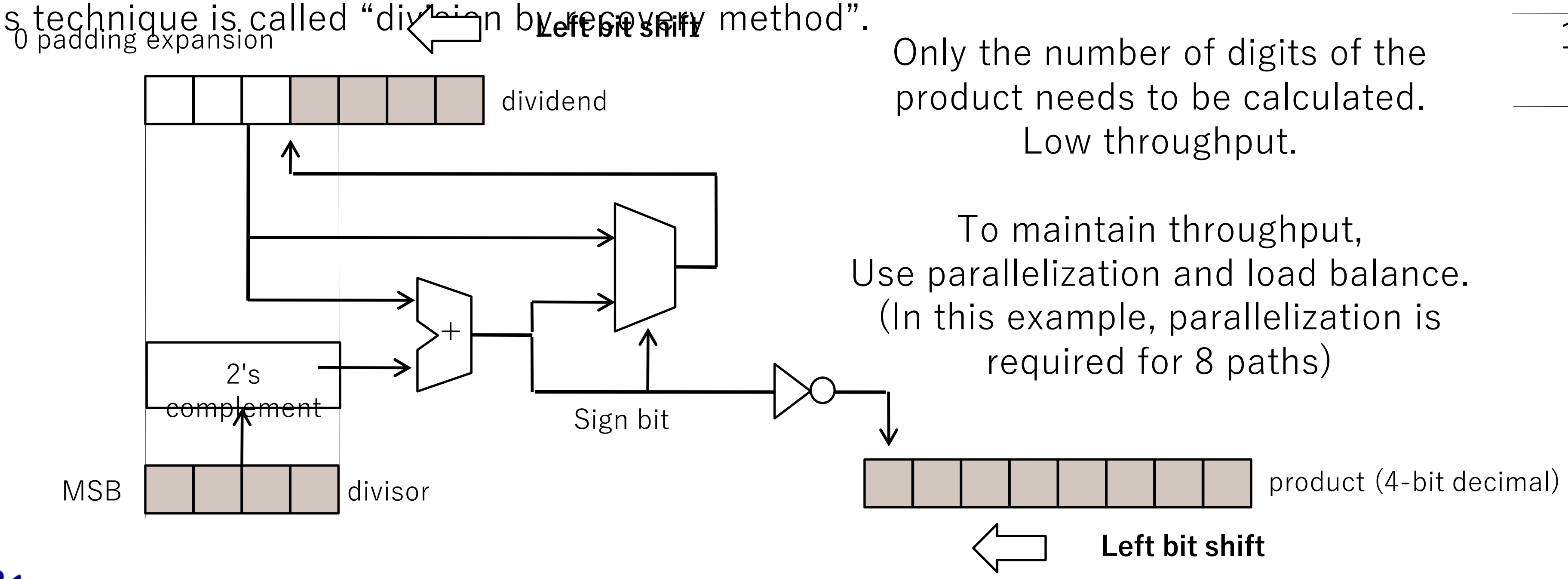
# Division without using multiplier

Try to consider the binary division with subtraction
- Try to subtract from the high-order bits of the dividend and the result is:
  - Negative: do nothing and shift the left bit to the next digit
  - Positive: the result of the subtraction is prepended and the left bit is shifted to the next digit.
  - The sign determines the Nth digit of the product as 0/1
- Repeat this operation until reaching the required precision

This technique is called “division by recovery method”.



Each digit of the product is 0 or 1
00110…
0011
1010
11
Cannot pull (will be negative)
10
11
101
11
100
11
10



Only the number of digits of the product needs to be calculated.
Low throughput.

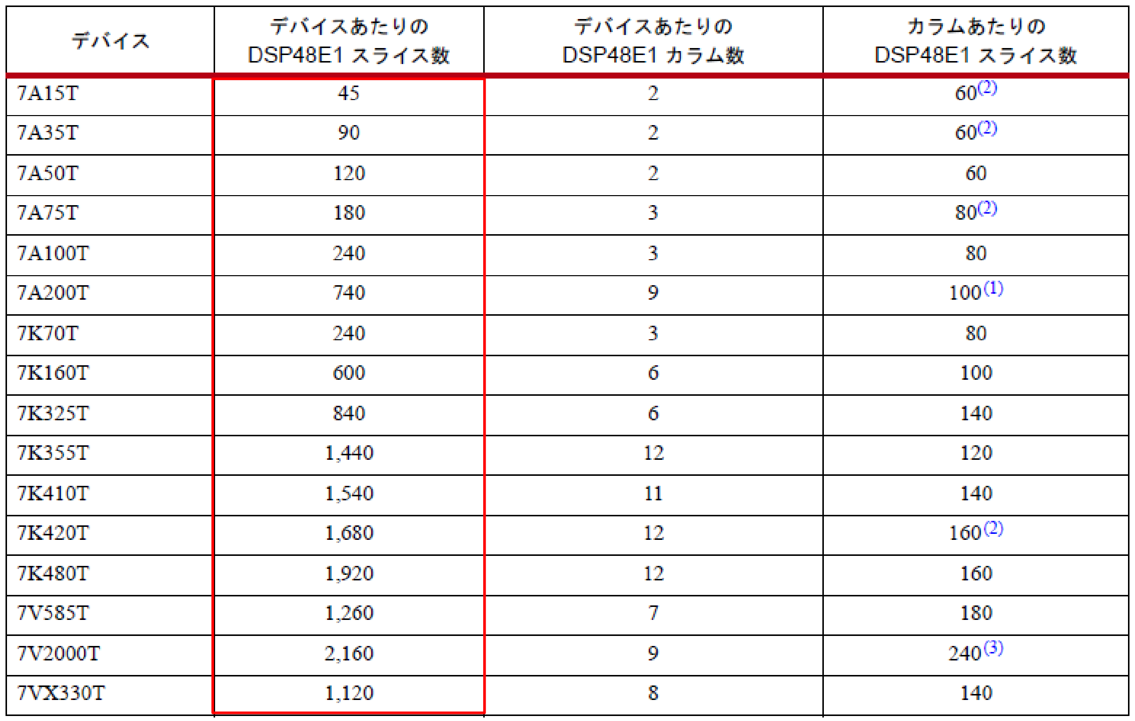
To maintain throughput,
Use parallelization and load balance.
(In this example, parallelization is required for 8 paths)

## From general-purpose to dedicated-purpose

For all the arithmetic circuits so far, they are synthesized by consuming CLB slices of FPGA.

For large-scale operations,
use the dedicated slice of **digital signal processor (DSP).**
Use the DSP48E1 primitives in Xilinx 7-series:.

表 2-1 : 7 シリーズの各デバイスの DSP48E1 スライス数

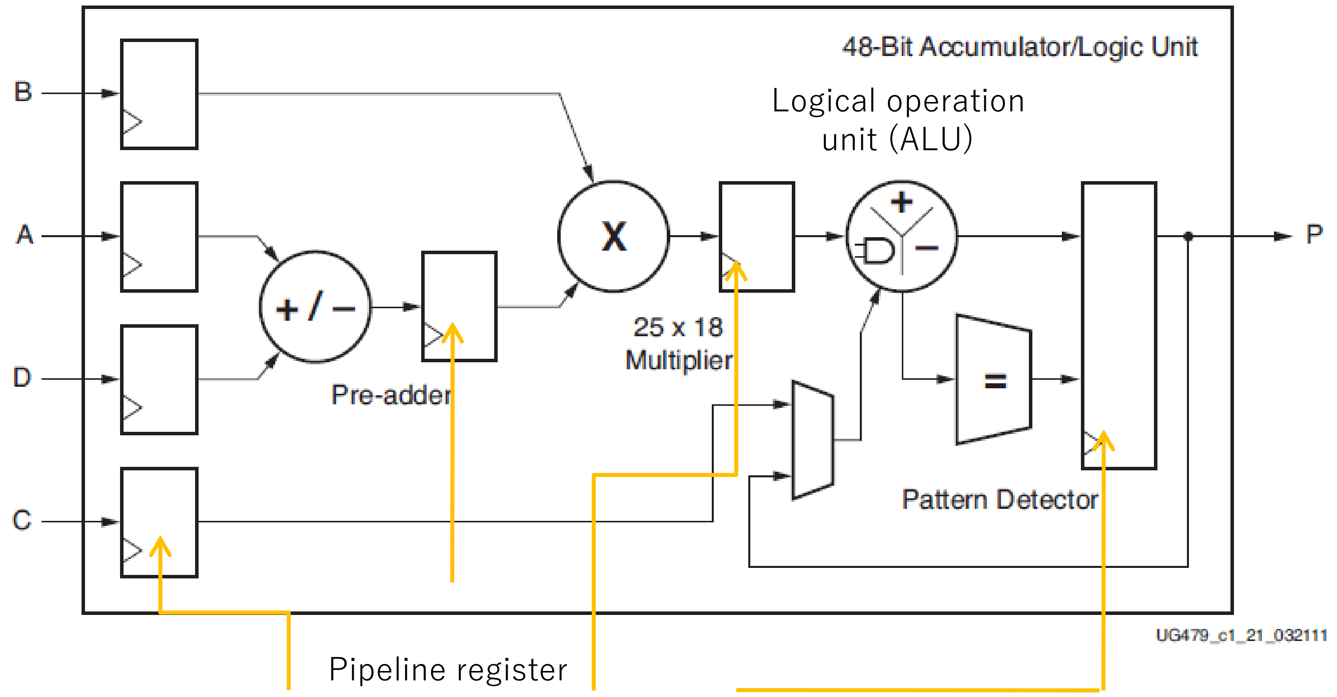


| デバイス | デバイスあたりの DSP48E1 スライス数 | デバイスあたりの DSP48E1 カラム数 | カラムあたりの DSP48E1 スライス数 |
|---|---|---|---|
| 7A15T | 45 | 2 | 60[2] |
| 7A35T | 90 | 2 | 60[2] |
| 7A50T | 120 | 2 | 60 |
| 7A75T | 180 | 3 | 80[2] |
| 7A100T | 240 | 3 | 80 |
| 7A200T | 740 | 9 | 100[1] |
| 7K70T | 240 | 3 | 80 |
| 7K160T | 600 | 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 |

## Outline of DSP48E1

**Can do addition, accumulation, multiplication, etc, with 4 inputs**
Below is a simplified diagram. The realistic case is more complicated.

- To fully utilize it, you have to read UG479.



**Examples of possible implementation:**

- (A+D)*B+C
- A+D+C
- A+P (accumulation)

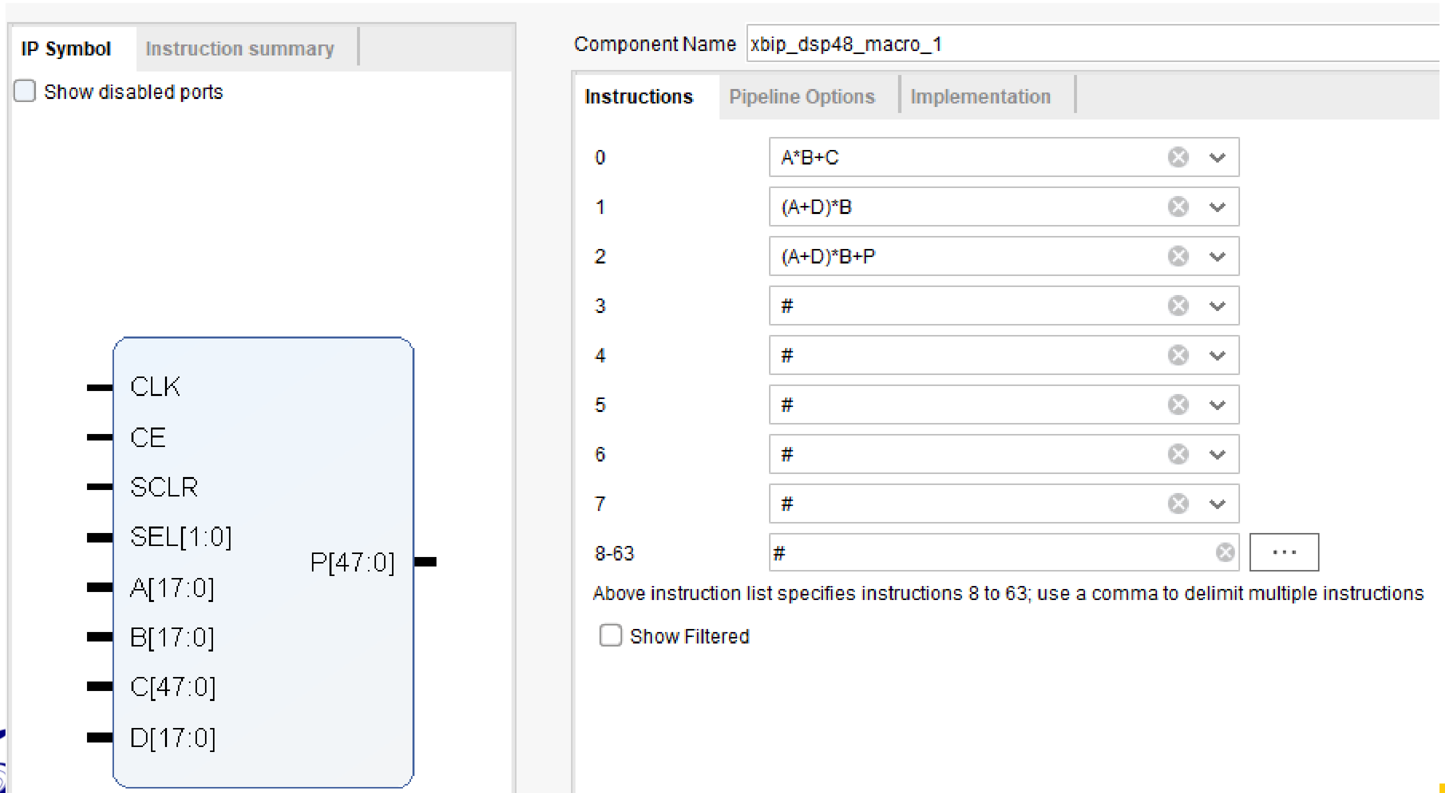
**Pattern detector**

- Overflow detection
- Auto-reset for counter (periodical pulse)

**DSP's input operand is signed(2's compliment)!**
**Recommended to extend the side at the HDL side**

## How to use it

1. **Generate and implement IP from the IP catalog**
    - **Recommended for this exercise**
2. Refer to the language template, and manually instantiate it on HDL.
    - Primitives of Xilinx FPGA is not limited to DSP, and it can be implemented directly from HDL
    - An advanced user must have a thorough understanding of how slicing works.
3. Infer DSP from HDL code
    - For people who don't want to introduce vendor dependencies to their codes.



Component Name xbip_dsp48_macro_1

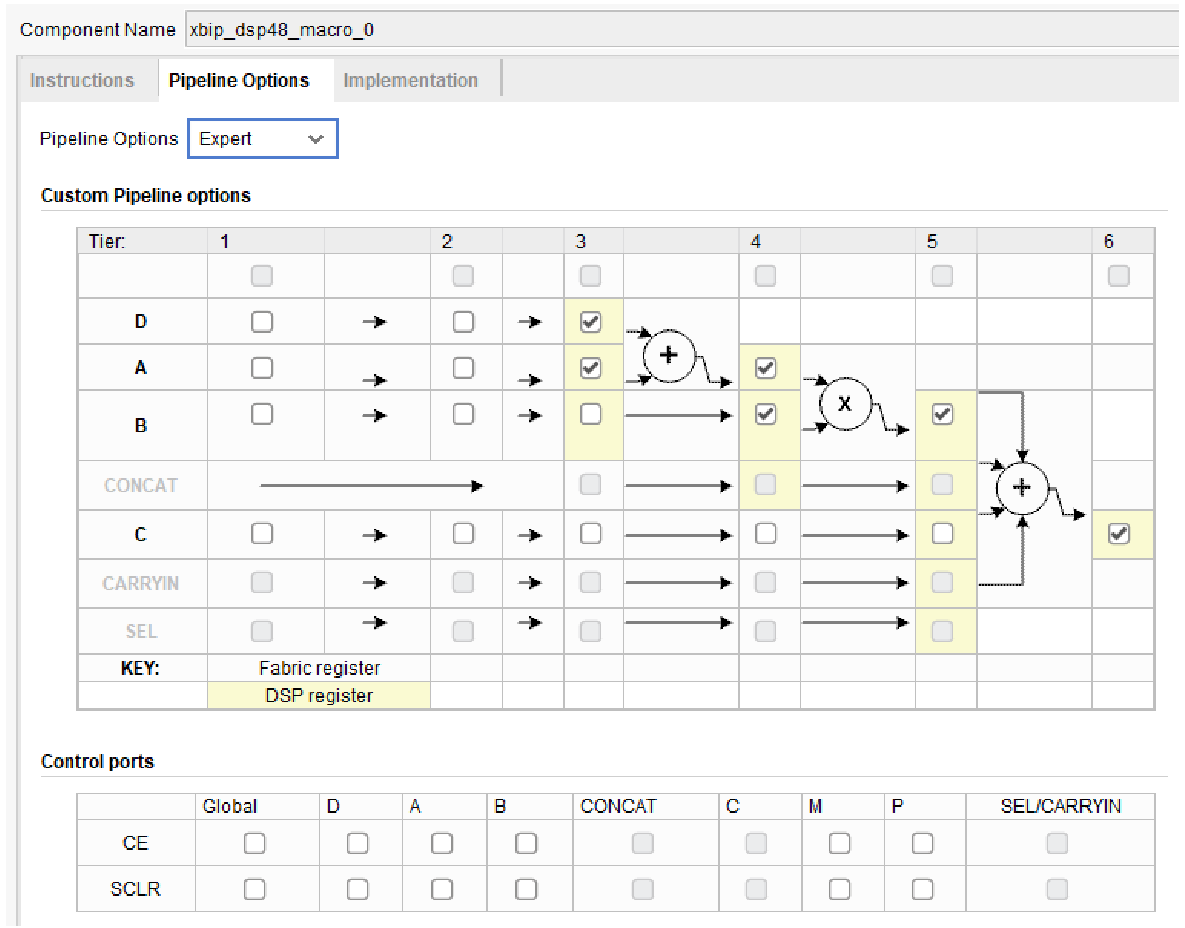
| | |
|---|---|
| 0 | A*B+C |
| 1 | (A+D)*B |
| 2 | (A+D)*B+P |
| 3 | # |
| 4 | # |
| 5 | # |
| 6 | # |
| 7 | # |
| 8-63 | # |

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

**SEL: Instruction switching**
- In this example, we defined 3 instructions, so the valid range of SEL is 0-2.

## How to use it

1. **Generate and implement IP from the IP catalog**
   - **Recommended for this exercise**



Component Name: xbip_dsp48_macro_0

Pipeline Options: Expert

**Custom Pipeline options**

| Tier: | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
|  | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |
| D | ☐ | ☐ | ☑ |  |  |  |
| A | ☐ | ☐ | ☑ | ☑ |  |  |
| B | ☐ | ☐ | ☐ | ☑ | ☑ |  |
| CONCAT |  |  | ☐ | ☐ | ☐ |  |
| C | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ |
| CARRYIN | ☐ | ☐ | ☐ | ☐ | ☐ |  |
| SEL | ☐ | ☐ | ☐ | ☐ | ☐ |  |

KEY: Fabric register / DSP register

**Control ports**

|  | Global | D | A | B | CONCAT | C | M | P | SEL/CARRYIN |
|---|---|---|---|---|---|---|---|---|---|
| CE | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |
| SCLR | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |

**Turn ON in Pipeline options**
**Possible ports:**

- CE: clock enable
- SCLR: Sync. reset
  - The accumulation result cannot be dynamically set to 0 without it.

Electronics System Group

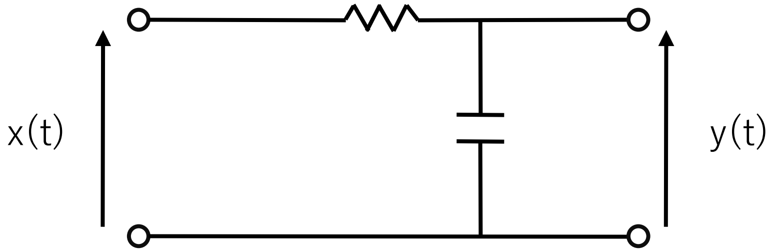
## Digital filter implementation by using DSP

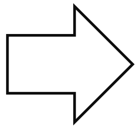
# Digression: Digital Filter

$$\begin{tikzcd}
\textbf{continuous time} & \text{Signal response} \arrow[r] & \text{differential equation} \arrow[r, "\text{Laplace transform}"] \arrow[d, "\text{backward difference}"'] & \text{s-function} \arrow[r] & \text{transfer function} \\
\textbf{discrete time} & & \text{difference equation} \arrow[r, "\text{z transform}"'] & \text{z-function} \arrow[r] & \text{transfer function}
\end{tikzcd}$$

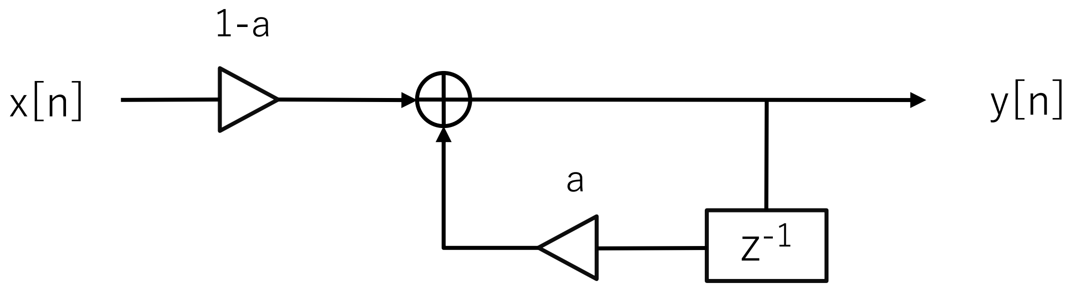
For a physics-major student, self-learning is needed for s-function.

## Analog filter





## Digital filter



**Adder** $\oplus$

**Multiplier** $\triangleleft$

Write the multiplier at above or below

**Delay** $\boxed{z^{-1}}$

Go back by the sampling time of the number on the shoulder.
This is 1-sample back.

# Digression: Digital Filter

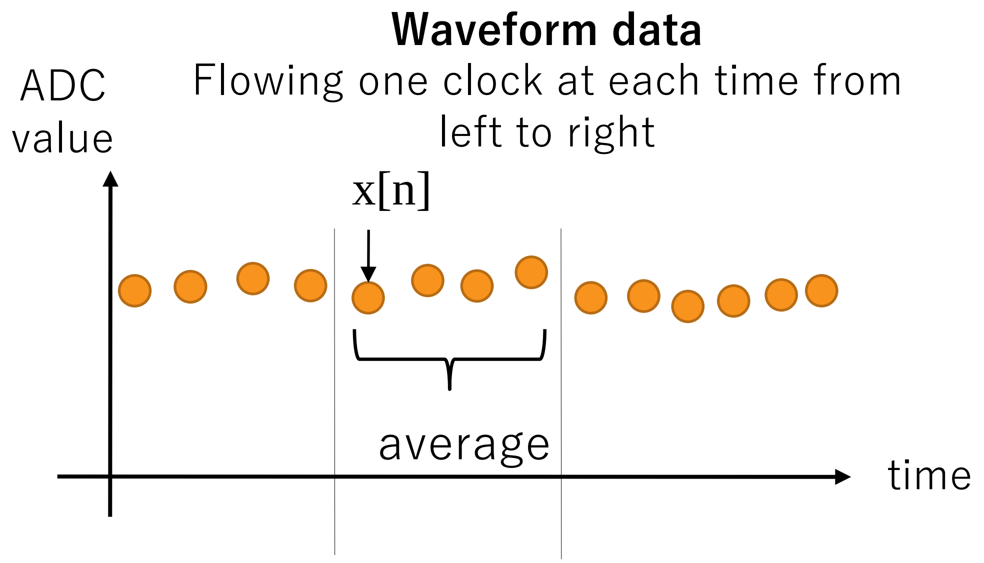
- はじめて学ぶディジタル・フィルタと高速フーリエ変換 (三上直樹 著, CQ出版)

  Digital Filters and Fast Fourier Transform for Beginners, by Naoki Mikami.
- ディジタル・フィルタ 理論&設計入門 (三谷政昭 著, CQ出版)

  Introduction to digital filter theory and design, by Masaaki Mitani.
- ディ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)

  Digital Filter Principle and Design Method (Design Technology Series), by Takehito Suyama.

# Moving average filter



n-th discrete time signal: x[n]

The difference equation for a moving average filter with N points is

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

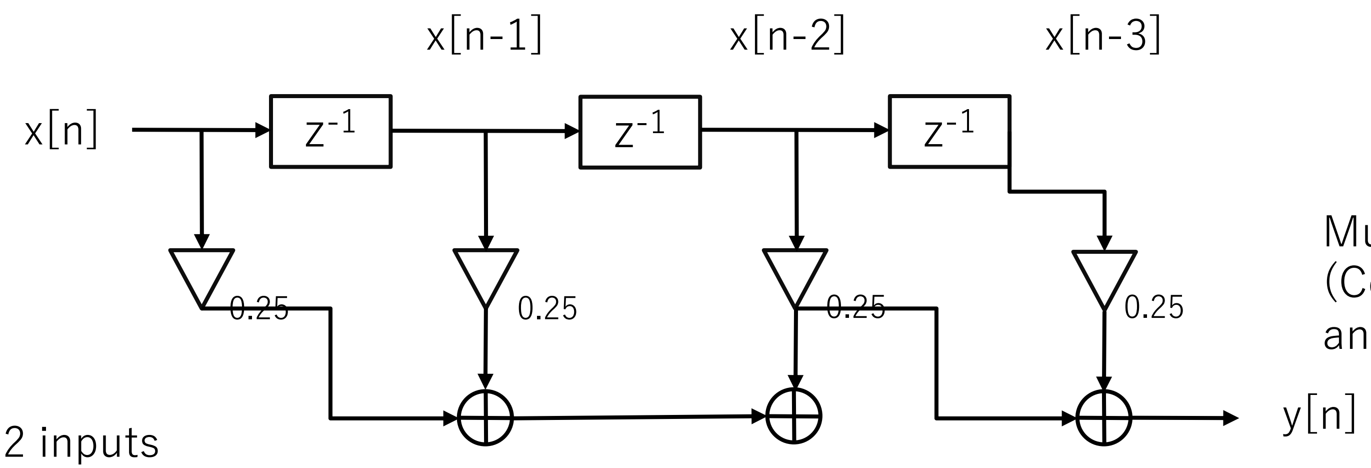
Taking the average of successive discrete signals is the same as making a moving average filter.

The moving average filter is easy to understand intuitively, since the sample average formula matchs the difference equation.

Try to implement a moving average filter using DSP

# Moving average filter

Consider a filter taking moving average of 4 points.
Because we only need to go back 3 points from the n-th sampling ...



Mutiply by 1/4
(Consider compatibility with DSP and apply it every time)

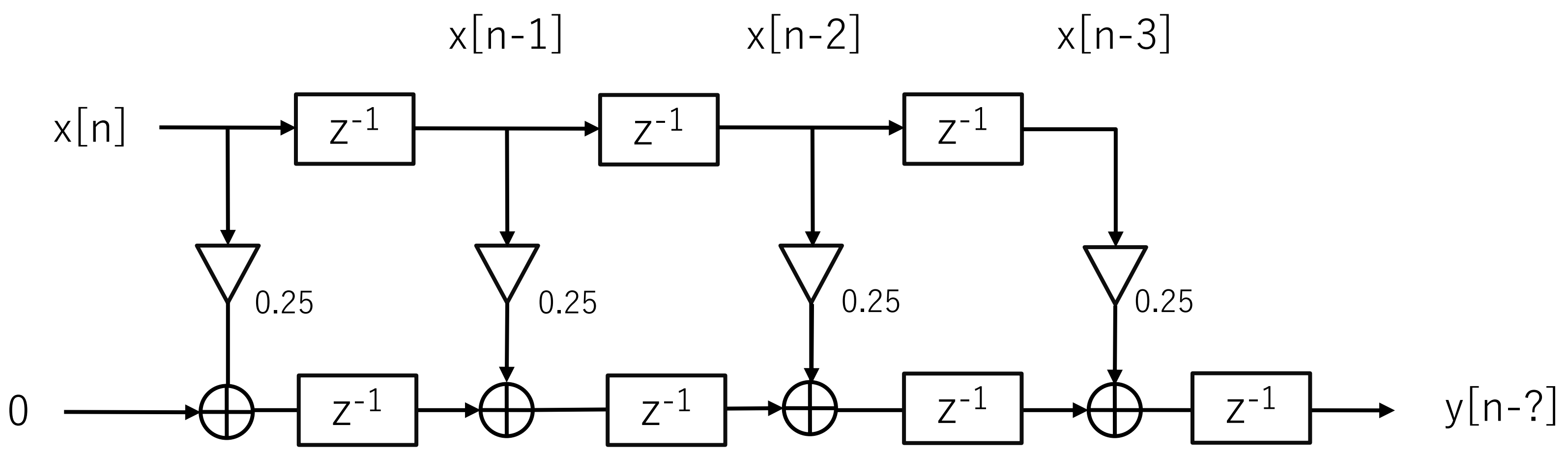
Since the adder has 2 inputs
… OK for now.
Where to put the next adder?

It is common to connect sideways

Complete
This is the form in ordinary textbooks
But...
Adder's propagation delay would integrate.
Could we implement it on an FPGA?

## Moving average filter

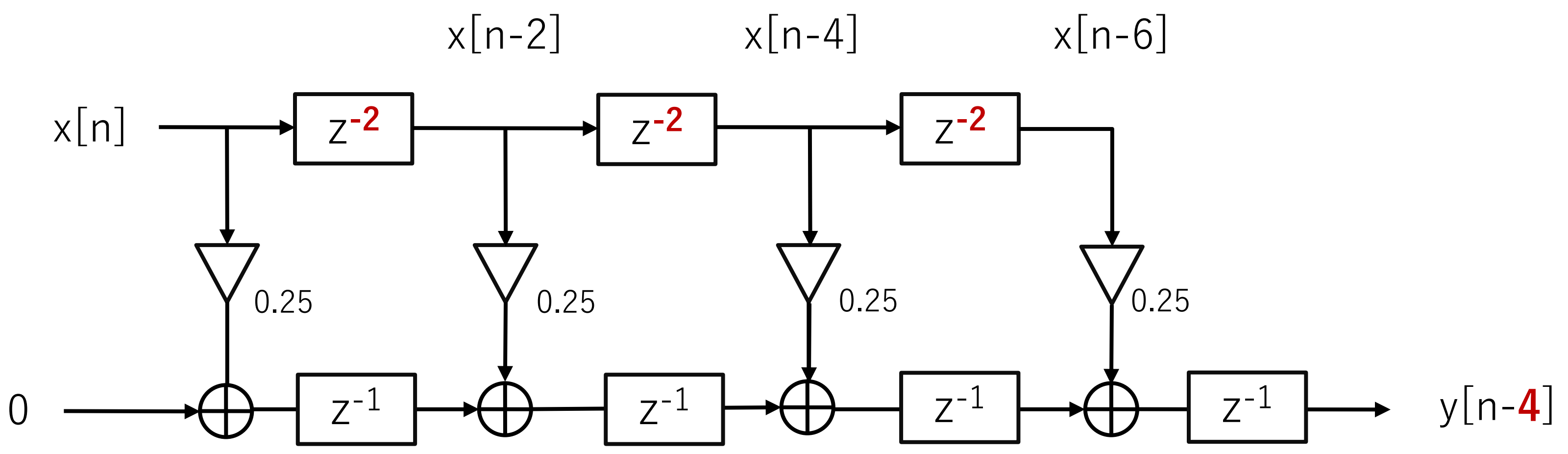
Let's put a register at the output of each adder.
No exception, so don't forget that the leftmost adder is also hidden.



When the register enters, the amount of delay for the number to be added is not enough...

# Moving average filter

The input must also be delayed by the number of the pipeline register which is put into the output of the adder.

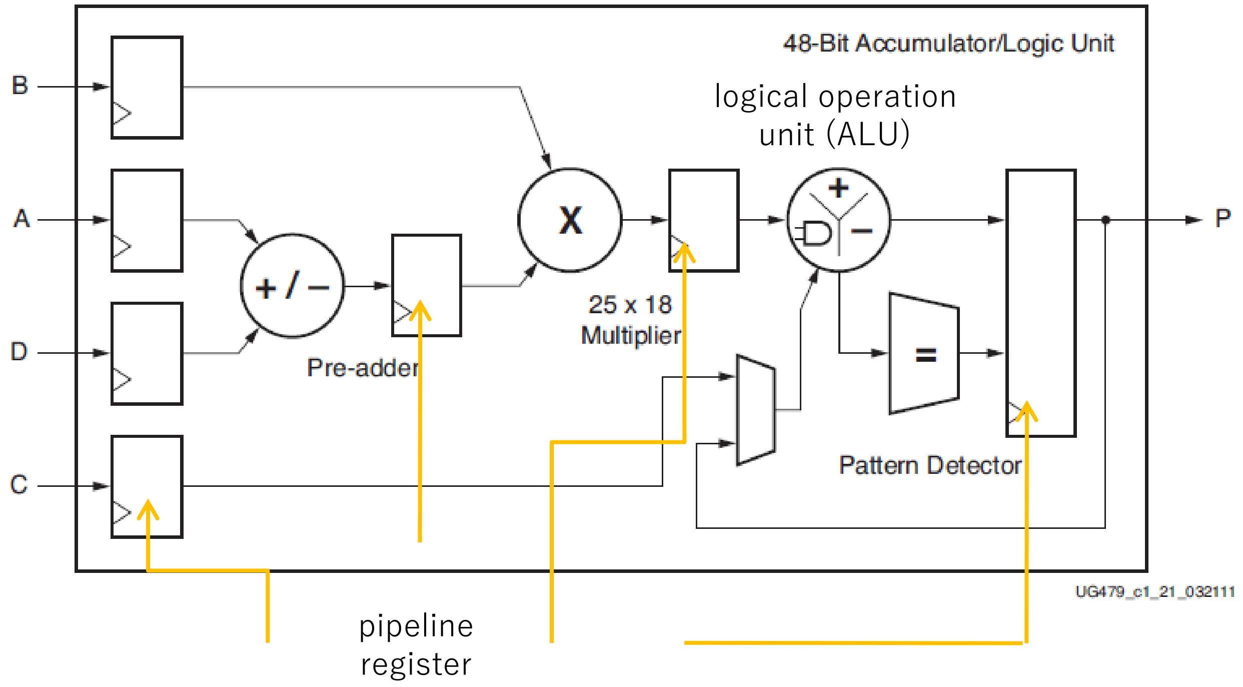


Complete?
The amount of output delay is determined by the number of delay elements in the shortest path.

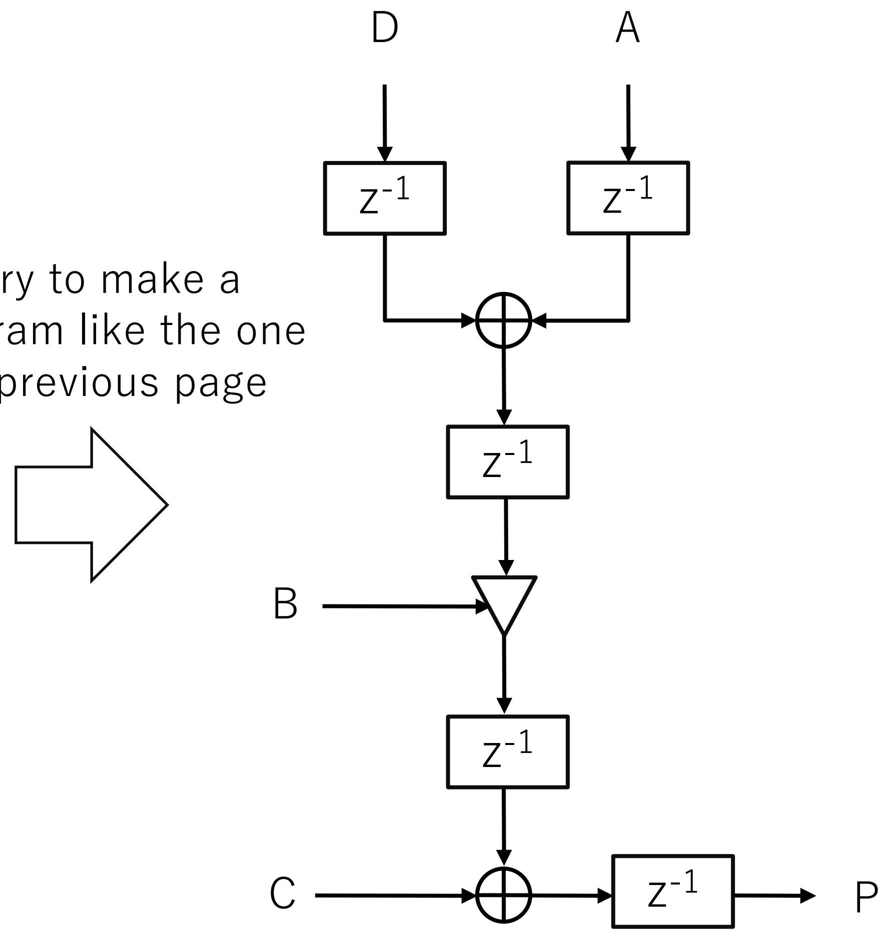
By the way, does the multiplier need registers?

# Moving average filter

Again, the structure of Xilinx FPGA's DSP
- There are 2 adders, 1 multiplier and 4 registers.



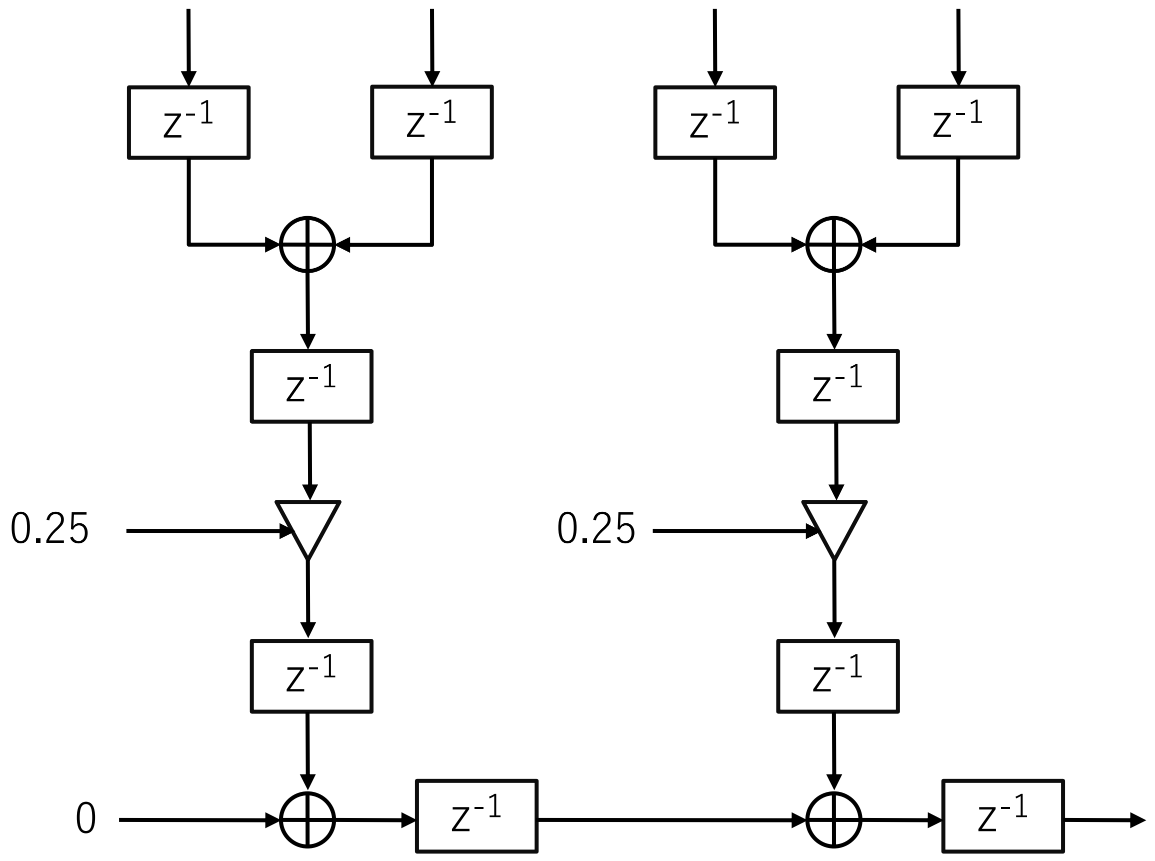
Try to make a diagram like the one in previous page



**The addition is nested.**

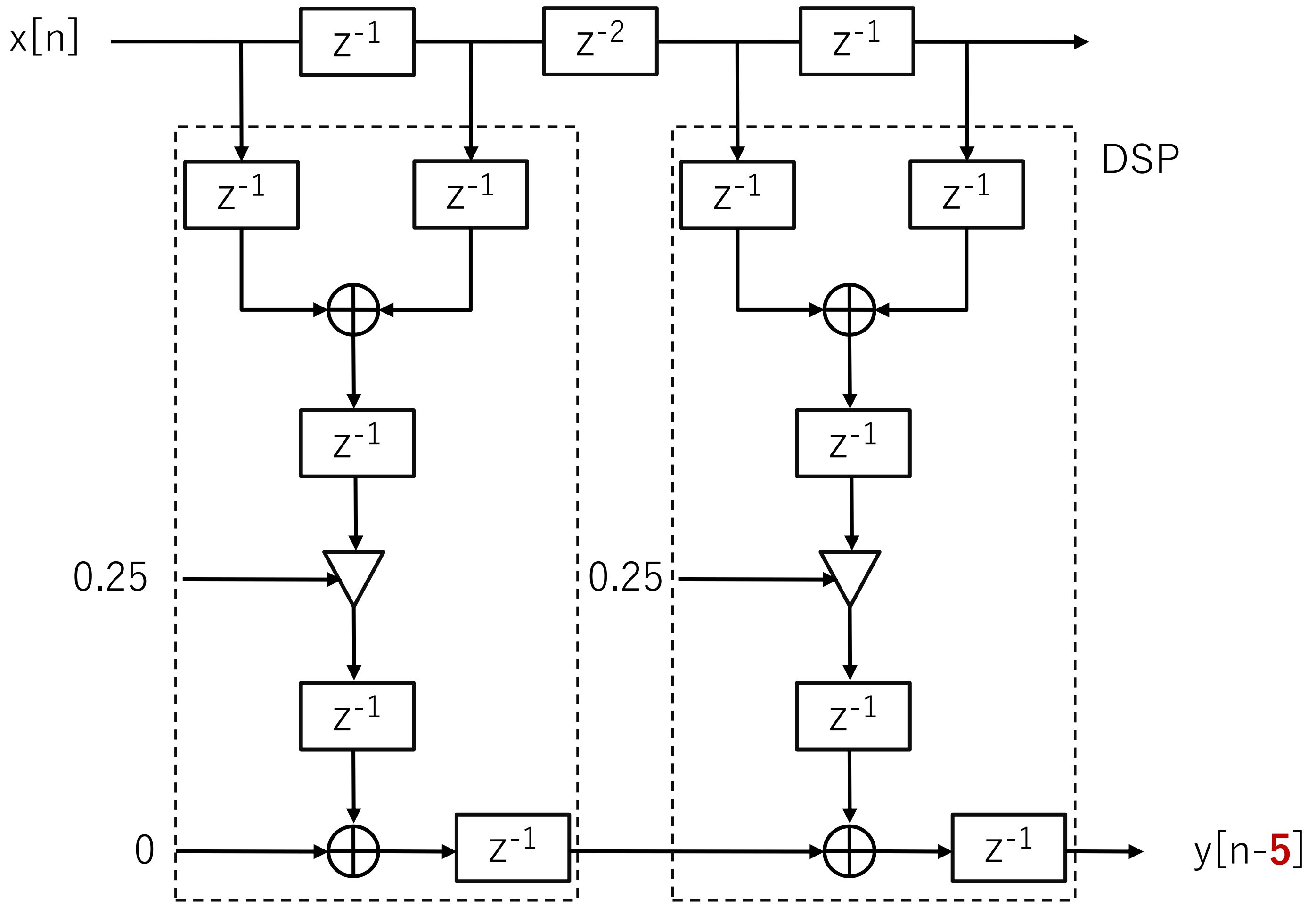
# Moving average filter

Determine the required pipeline registers and input delays



# Moving average filter

Determine the required pipeline registers and input delays



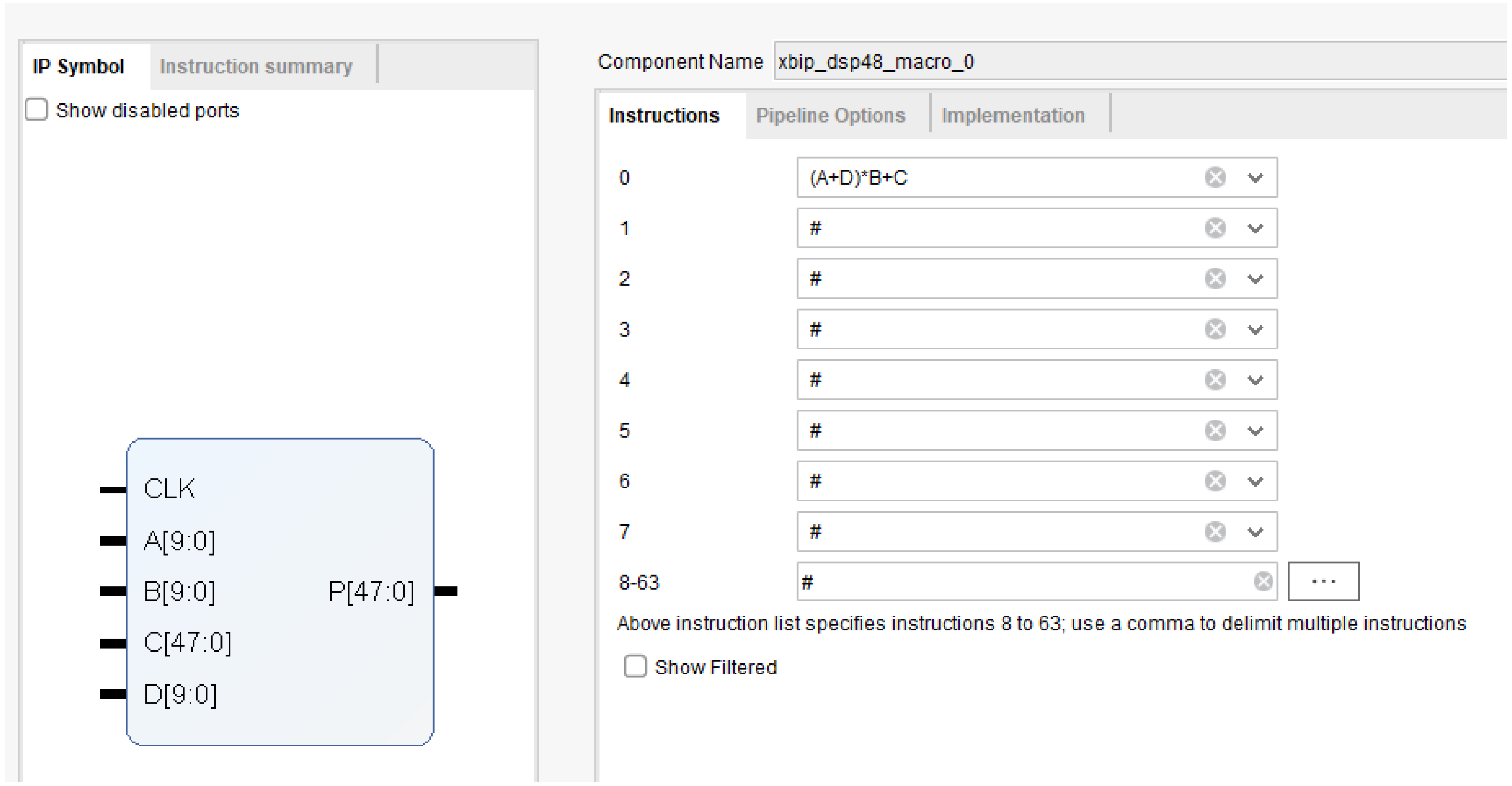
**Complete**
Let’s implement it.

**Condition**
signed fixed point
Integer part: 8-bit
Decimal part: 2-bit

Refer to the example manual
EX1 for details

## Moving average filter

Generate DSP from IP Catalog



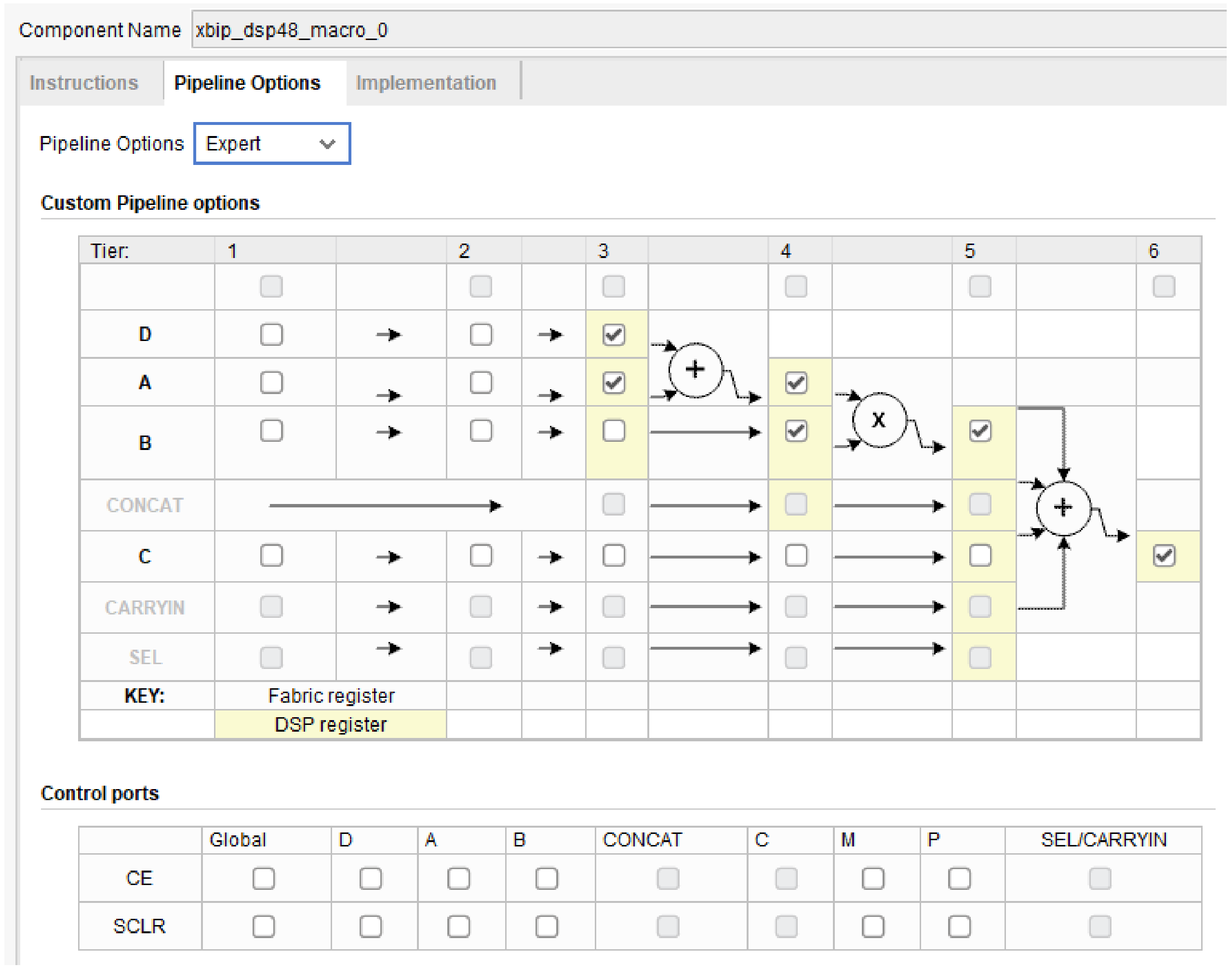
Component Name: xbip_dsp48_macro_0

| | |
|---|---|
| 0 | (A+D)*B+C |
| 1 | # |
| 2 | # |
| 3 | # |
| 4 | # |
| 5 | # |
| 6 | # |
| 7 | # |
| 8-63 | # |

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

# Moving average filter

## Generate DSP from IP Catalog



Component Name: xbip_dsp48_macro_0

Instructions | **Pipeline Options** | Implementation

Pipeline Options: Expert

**Custom Pipeline options**

| Tier: | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |
| D | ☐ | ☐ | ☑ | | | |
| A | ☐ | ☐ | ☑ | ☑ | | |
| B | ☐ | ☐ | ☐ | ☑ | ☑ | |
| CONCAT | | | ☐ | ☐ | ☐ | |
| C | ☐ | ☐ | ☐ | ☐ | ☐ | ☑ |
| CARRYIN | ☐ | ☐ | ☐ | ☐ | ☐ | |
| SEL | ☐ | ☐ | ☐ | ☐ | ☐ | |
| KEY: | Fabric register | | | | | |
| | DSP register | | | | | |

**Control ports**

| | Global | D | A | B | CONCAT | C | M | P | SEL/CARRYIN |
|---|---|---|---|---|---|---|---|---|---|
| CE | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |
| SCLR | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ | ☐ |

Since it is implemented just as shown in the previous figure,
register to C input is omitted.

Did it move?
The moving average should increase by 1.
Is the latency between input and output 5?

Since the DSP receives the operands in 2's complement notation, the result of the moving average may be negative sometimes.
Let's check the movement.