Computer Science A
Hardware Design Excise Text 5
Fast Fourier Transform

May 9th.,2019

CSAHW Computer Science A, Meiji University

CSA_B3_Text_5.PPTX 27 Slides

Renji Mikami

講義(補足)編5について

FFTについて.

FFT算法の解説は、数式だけの解説ではわかりづらいので。
ここでは、パターンのイメージから解説します。データを並べかえることによって、次々に共通項が現れる面白さを体験してください。FFTの発見者も最初は、きっと図を書いて、共通パターンをさがしたのだと思います。

図で、イメージがわいたら、これはそのまま行列で解けます、あとはバタフライ演算に持込みます。細かくやればやるほどに計算量が減っていきますが、プログラム化するとサブルーチンが増えて複雑になっていきますので、適当なところで切り上げてください。

授業のサイトに典型的な64ポイントFFTのモジュールのソースコードを示してありますので、参考にしてください。

Text4 の復習

連続量の離散化とは、連続量を一定の区間で区切り、この区間で信号が周期的に繰り返すとして処理を行う。周期的信号を扱うフーリエ級数展開に近い。

連続量の離散化には、サンプリングを行うがサンプリング周波数の1/2をナイキスト周波数といい、この周波数以下のデータは正しく再現されない。サンプリングと複素フーリエ級数展開は双対となる。

離散フーリエ変換は、サンプリング値と正弦関数、余弦関数の離散値との相関を計算することであり、正弦、余弦をひとつにまとめた回転子(TF)を用いて、行列演算を行うことで、効率的に計算を行うことができる。

回転子は、周期性と対象性の性質があり、これによって計算をさらに効率化することができる。

高速フーリエ変換(FFT)

離散フーリエ変換を高速で計算するアルゴリズム

比較的新しい(1965年 James.W.Cooley, John.W.Tukey,)

基数2のFFTが基本で、時間間引きと周波数間引き法があり、時間間引き法が一般的だが、両者は基本的に同じものと考えてよい。

高速化は、TFの周期性と対称性を利用した、データの2分割の繰り返しとバタフライ演算による計算量の削減によって行われる。

現在でも基数の異なるもの、ハードウェア解などさまざまな研究が進んでいる。DFT部分もさらに発展して、離散コサイン変換(DCT)やウェーブレット変換なども応用として使われているが、演算部分でFFTのような高速の演算アルゴリズムを使えることがこれらの応用を飛躍的に高めています。

DFTの計算量とFFT

DFTは、データ数nとTFのn行n列の行列演算を行います。データ数が2の場合は、乗算2回と加算2回の演算になります。データ数が4の場合は、乗算16回加算16回、データ数が8の場合は、同64回ずつになります。つまり2^nの演算回数が必要となります。

FFTの算法は、項を奇数と偶数に分割して計算します。そのため、それぞれのデータ数が半分になり、べき乗で増加する計算量を削減できます。たとえば8データのDFTを4データのDFT2つに分解できれば、64回の乗算が32回に削減でします。同様に4データ化したDFTをさらに2データのDFTに分解すると、乗算の回数は、16回で済みます。

また分解の過程で、同じ形の項が生成されますから、これらは一度計算すればよいことになります。

FFTでは分割と式の変形による統合を繰り返し行い、2^nの計算量を$n\log_2 n$に減らすことができ、プログラム化が容易です。

以降の解説では、奇数項と偶数項をまとめ、分割し、式を変形して、バタフライ演算の形にもっていく点に着目してください。

FFTの応用例：OFDM

OFDMは、無線通信分野で使われている、デジタル変調方式で、直交周波数上の複数のサブキャリアを使用して、デジタル変調したデータを送受信する方式です。

サブキャリアの直交性を利用して、送信側で逆フーリエ変換したデータを受信側でフーリエ変換して、元のデータに復調します。このときデータは、直交するサブキャリアに乗っていますから、元のデータの取り出しが容易になります。(テキスト2の直交性を思い出してください。)

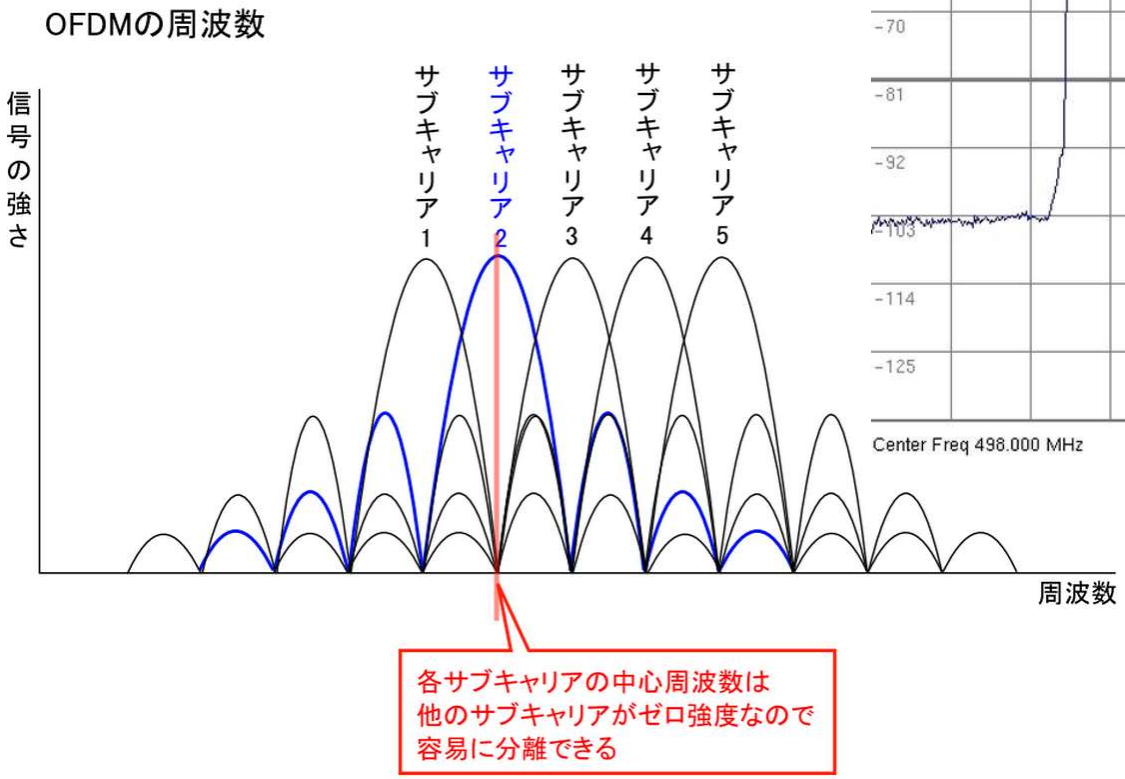
DFTによるフーリエ変換では、計算量が多くなりますから、これをFFTで計算します。演習では、64ポイントFFTを取り上げます。ソースはHPに掲載してあります。

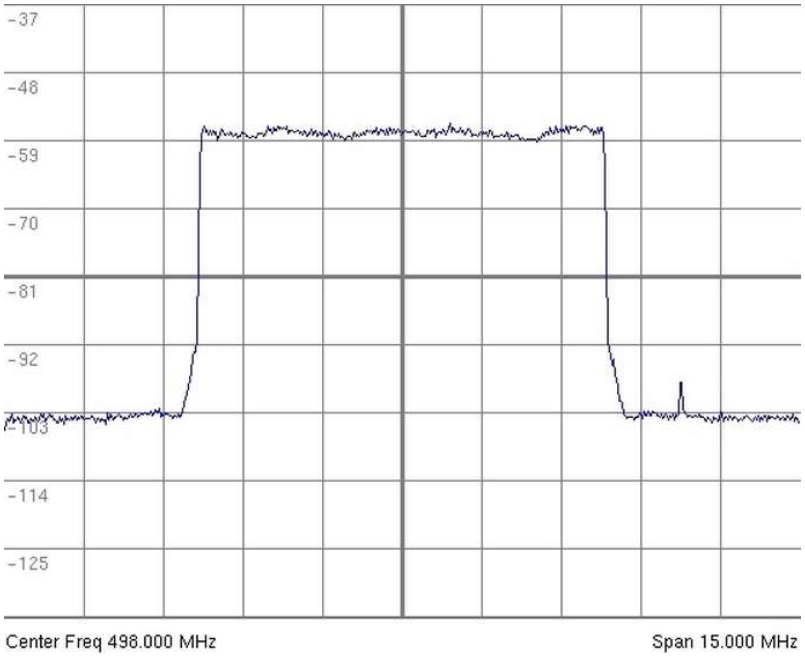
FFTの場合、基数が重要になります。64ポイントFFTを基数4と2で分解した場合の例を示します。パイプラインは、乱れませんので、段数を増やしても1段あたりの計算速度は落ちません。

しかし基数を2とすると、一段あたりの計算回路が小型になり、結果として高速の演算が可能になります。

OFDM(直交周波数分割多重), 無線LAN IEEE802.11a,g,n

マルチキャリア送信

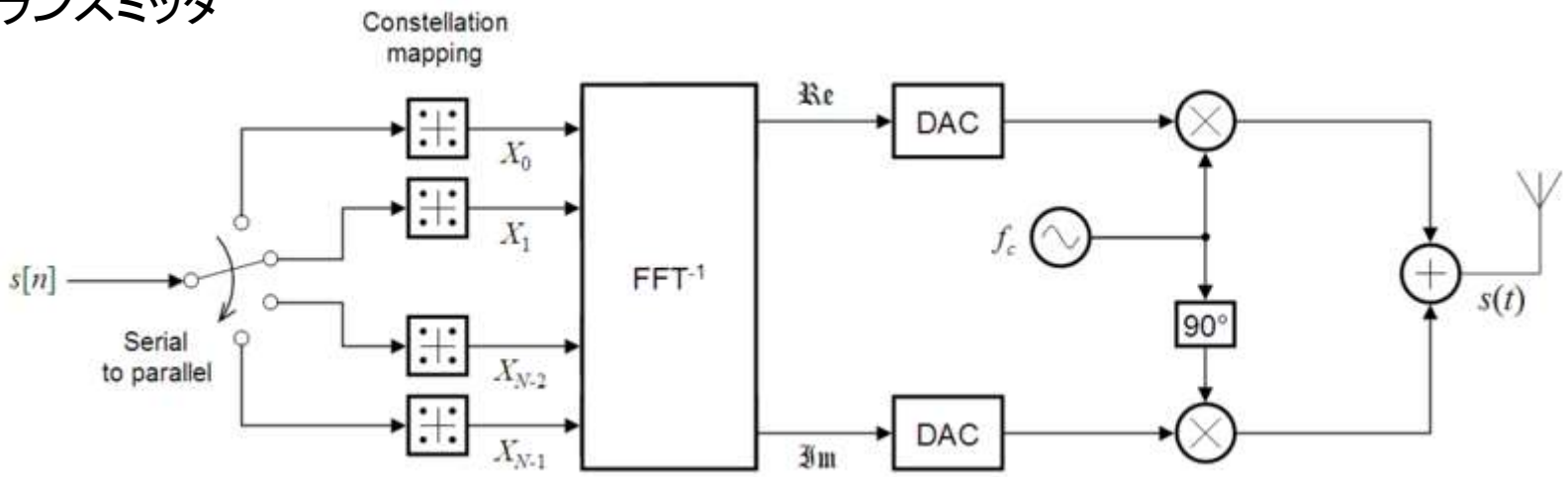




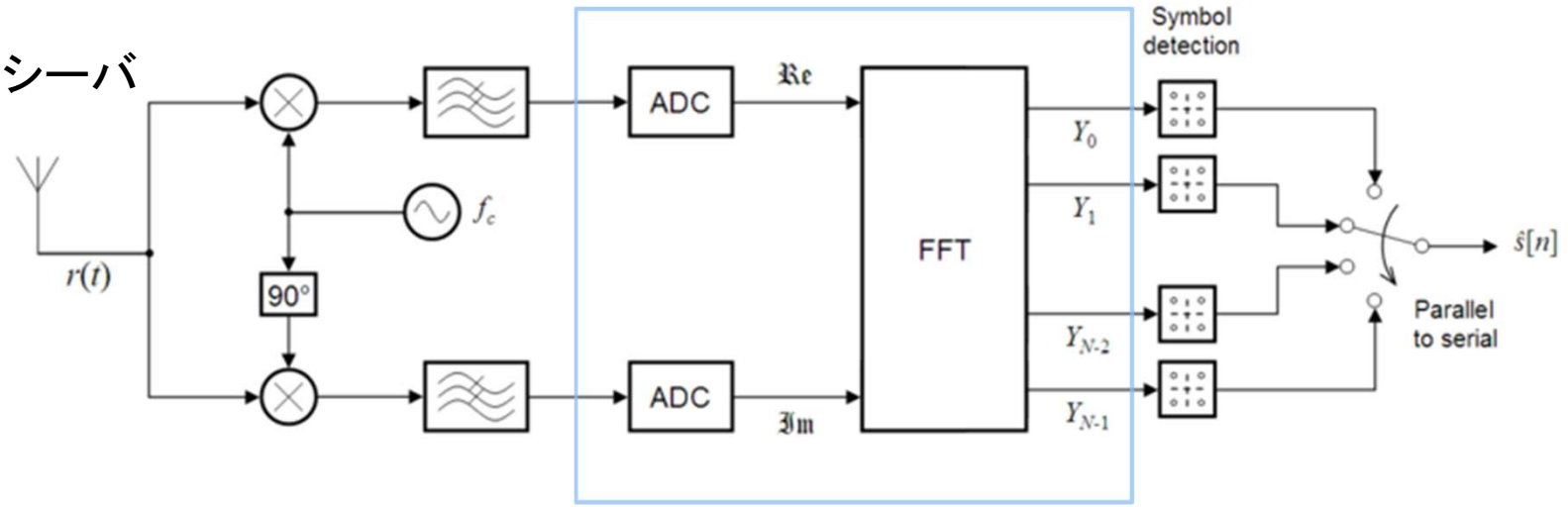
引用 : https://ja.wikipedia.org/wiki直交周波数分割多重方式

64ポイント FFT実装-
OFDM(直交周波数分割多重), 無線LAN IEEE802.11a,g,n

トランスミッタ



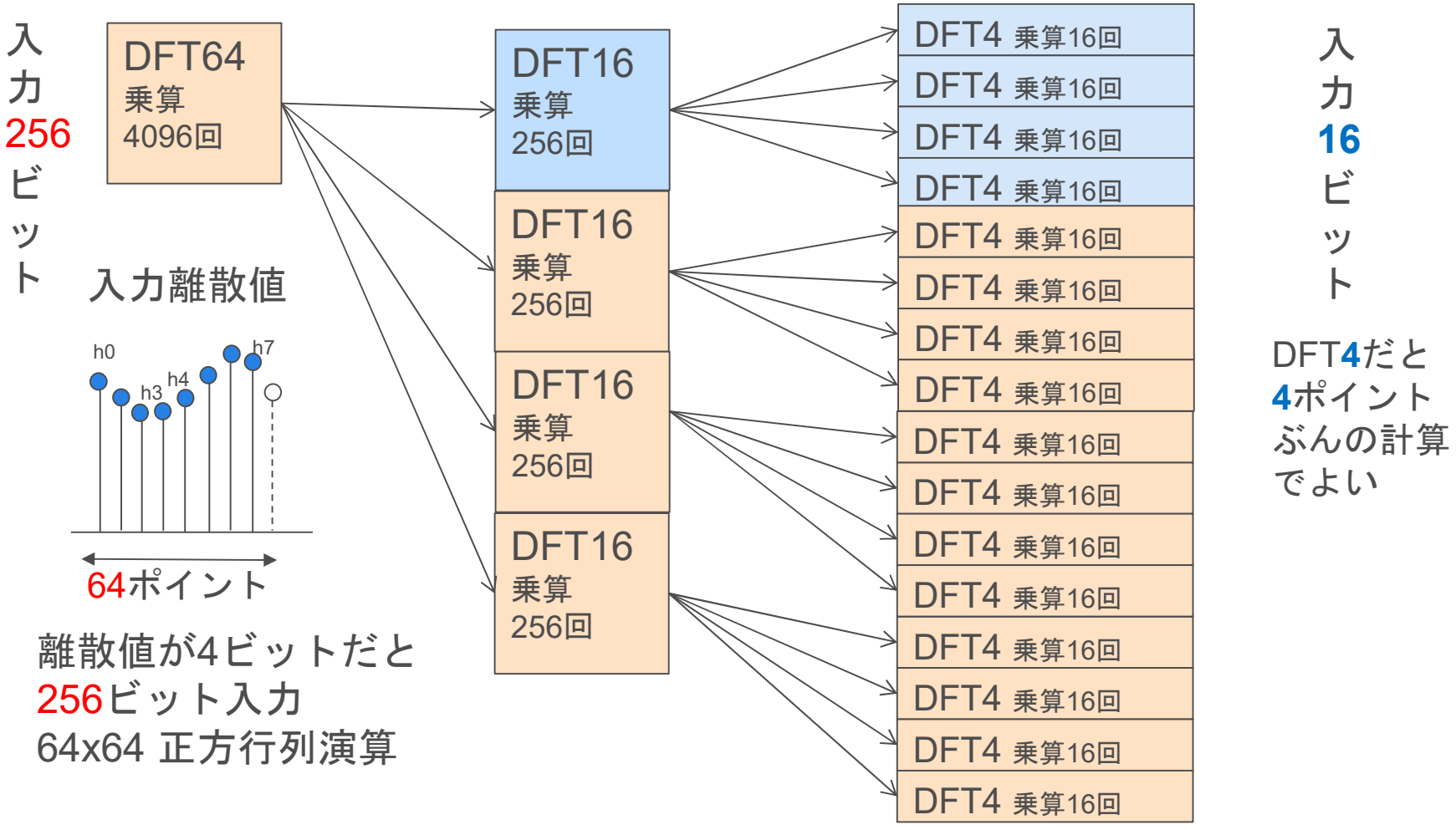
レシーバ



引用：https://ja.wikipedia.org/wiki直交周波数分割多重方式

64ポントFFT分解 1 (離散値4ビット)

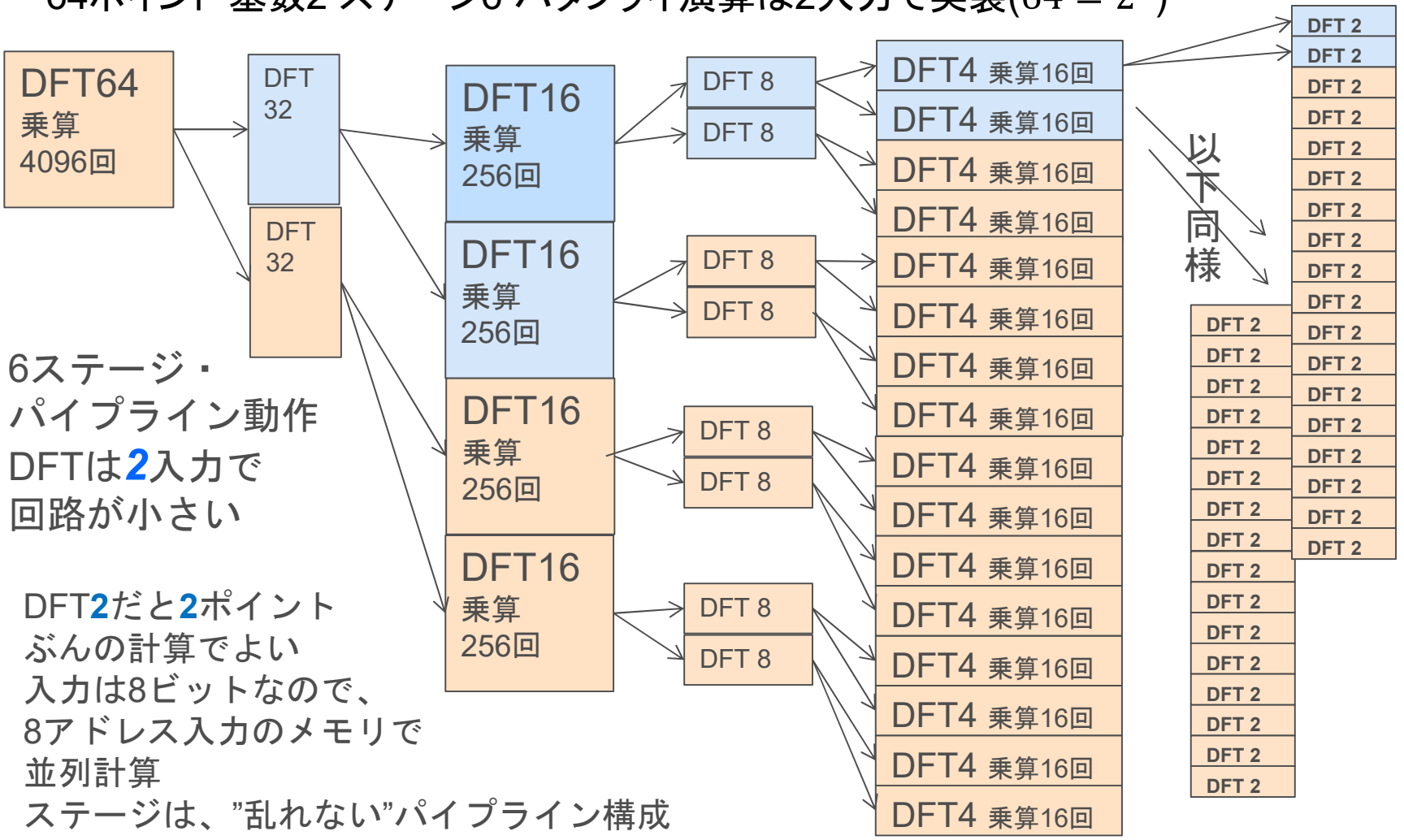
64ポイント 基数4 ステージ3 バタフライ演算は4入力で実装 $(64 = 4^3)$



3ステージ・パイプライン動作DFTは4入力で回路が大きい

64ポントFFT分解 2 (基数2で分解)

64ポイント 基数2 ステージ6 バタフライ演算は2入力で実装($64 = 2^6$)



DFTの計算演習課題(テキスト4)の再考

TFのW^0からW^3 、までの値を回転子の向きを表す矢印↓に書き換えて、そのパターンを考えてみます。

同じ番号の行と列のデータの並びが同じになります。

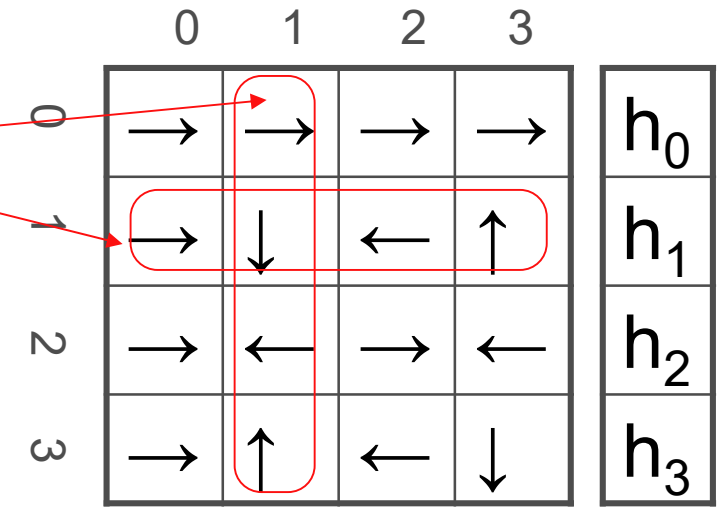
$$\begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^0 & W_4^2 \\ W_4^0 & W_4^3 & W_4^2 & W_4^1 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

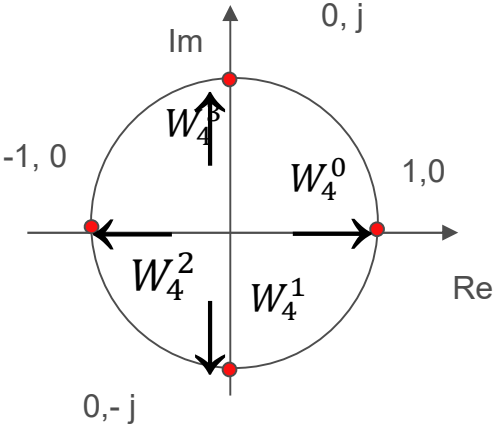
0番の列：→ → → →
0番の行：→ → → →

1番の列：→↓←↑
1番の行：→↓←↑

2番の列：→← →←
2番の行：→← →←

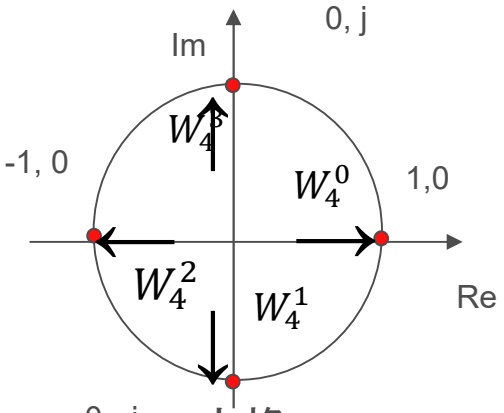
3番の列：→ ↑ ←↓
3番の行：→ ↑ ←↓

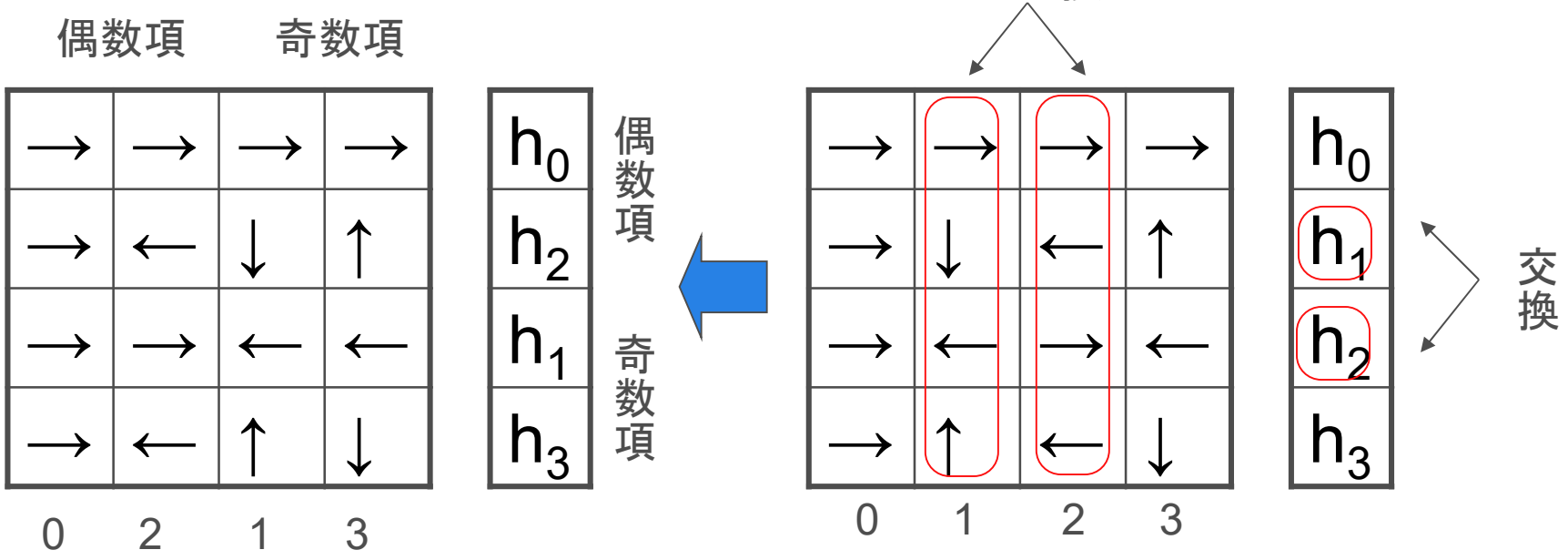




項の並べ替え

h1,h2の行を交換し、TFの2列3列を交換しても行列演算の結果は変わりません。偶数項、奇数項同士がまとまり、隣り合うようになります。(ビットリバーサル-後述)





TFの対称性と周期性

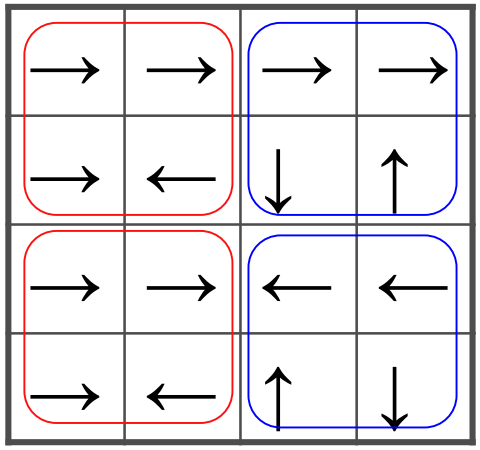
行の交換後、全体を4つのブロックに分割して考えると上のふたつのブロックのパターンは同じです。また下のふたつのパターンは、お互いが対称になっています。

上のパターンは、そのままくくりだしができ、下のパターンは反転してくくりだせます。この性質を使うと4項のDFT計算を2項のDFT計算に置き換えることができます。

W^0	W^0	W^0	W^0	W^0	W^0	W^0	W^0	h0
W^0	W^1	W^2	W^3	W^4	W^5	W^6	W^7	h1
W^0	W^2	W^4	W^6	W^0	W^2	W^4	W^6	h2
W^0	W^3	W^6	W^1	W^4	W^7	W^2	W^5	h3
W^0	W^4	W^0	W^4	W^0	W^4	W^0	W^4	h4
W^0	W^5	W^2	W^7	W^4	W^1	W^6	W^3	h5
W^0	W^6	W^4	W^2	W^0	W^6	W^4	W^2	h6
W^0	W^7	W^6	W^5	W^4	W^3	W^2	W^1	h7

偶数項

奇数項



h_0
h_2
h_1
h_3

この性質は、2^nの項数で成り立ち、上の8項のDFTは、ビットリバーサルで並べ変えたあとは、4項のDFTに分解することができます。また分解された4項のDFTは、さらに左のように2項のDFTに分解することができます。

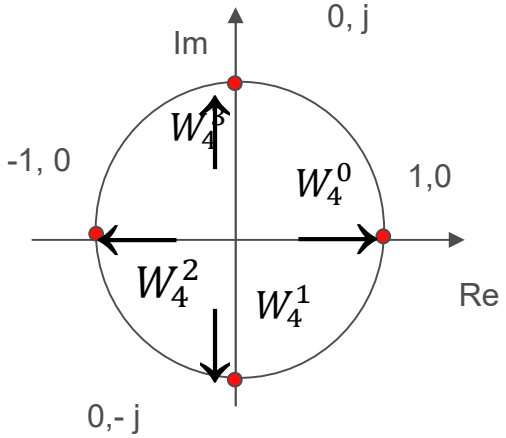
これが高速フーリエ変換で、項数nのDFTの乗算の回数2^nをFFTでは、$n\log_2 n$に減らすことができます。

W^nの対称性

TFの回転子では、W^0とW^2 、W^1とW^3 は、お互いに原点に対して対称の位置にあります。

このように原点対称の位置にある回転子は、符号を反転させると相互に変換することができます。また、W^0は1です。

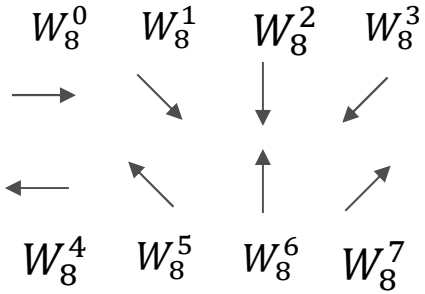
$$W_n^0 = 1$$
$$W_4^1 = -W_4^3$$
$$W_4^2 = -W_4^0$$
$$W_4^3 = -W_4^1$$

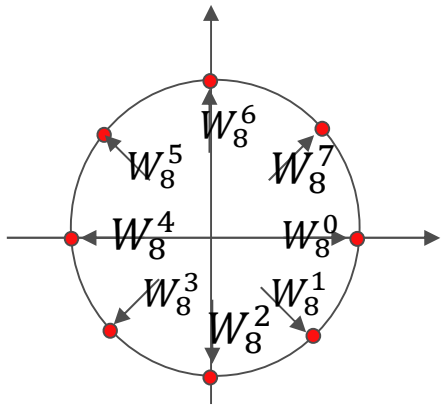


W_8^nの場合は

右のようになります。

対称性を頭に入れておいてください





部分行列によるDFTの分解と統合1

*DFT*式

$$\begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^0 & W_4^2 \\ W_4^0 & W_4^3 & W_4^2 & W_4^1 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix} =$$

項を入れ替えたもの

$$\left[\begin{array}{cc:cc} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^2 & W_4^1 & W_4^3 \\ \hdashline W_4^0 & W_4^0 & W_4^2 & W_4^2 \\ W_4^0 & W_4^2 & W_4^3 & W_4^1 \end{array}\right] \begin{bmatrix} h_0 \\ h_2 \\ h_1 \\ h_3 \end{bmatrix}$$

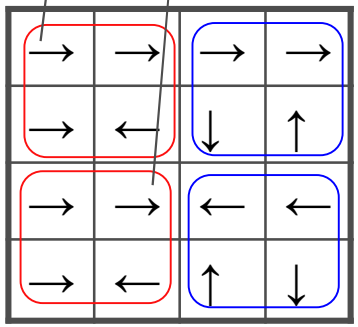
を与式として

与式を部分行列化

$$\text{与式} = \begin{bmatrix} \begin{bmatrix} W_4^0 & W_4^0 \\ W_4^0 & W_4^2 \end{bmatrix} & \begin{bmatrix} W_4^0 & W_4^0 \\ W_4^1 & W_4^3 \end{bmatrix} \\ \begin{bmatrix} W_4^0 & W_4^0 \\ W_4^0 & W_4^2 \end{bmatrix} & \begin{bmatrix} W_4^2 & W_4^2 \\ W_4^3 & W_4^1 \end{bmatrix} \end{bmatrix} \begin{bmatrix} \begin{bmatrix} h_0 \\ h_2 \end{bmatrix} \\ \begin{bmatrix} h_1 \\ h_3 \end{bmatrix} \end{bmatrix}$$

部分行列単位で計算

$$= \begin{pmatrix} \begin{bmatrix} W_4^0 & W_4^0 \\ W_4^0 & W_4^2 \end{bmatrix} \begin{bmatrix} h_0 \\ h_2 \end{bmatrix} + \begin{bmatrix} W_4^0 & W_4^0 \\ W_4^1 & W_4^3 \end{bmatrix} \begin{bmatrix} h_1 \\ h_3 \end{bmatrix} \\ \begin{bmatrix} W_4^0 & W_4^0 \\ W_4^0 & W_4^2 \end{bmatrix} \begin{bmatrix} h_0 \\ h_2 \end{bmatrix} + \begin{bmatrix} W_4^2 & W_4^2 \\ W_4^3 & W_4^1 \end{bmatrix} \begin{bmatrix} h_1 \\ h_3 \end{bmatrix} \end{pmatrix}$$

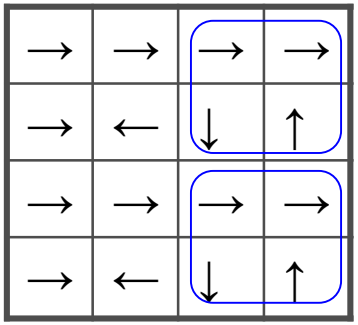


左側の枠(赤)は上下同じパターンです
右側の枠(青)は上下が反転しています、これは-1を掛けるのと同じになります。-1は、W_4^2です。
この部分行列化は、n=4のDFTがn=2のDFTに分解できること、同じ項の再使用で計算を削減できることを示しています。

部分行列によるDFTの分解と統合2

$W_4^2=-1$を使って右枠(青)の項を強引?に同じにします。$W_4^0=1$です

$$与式=\begin{pmatrix}\begin{bmatrix}W_4^0 & W_4^0\\ W_4^0 & W_4^2\end{bmatrix}\begin{bmatrix}h_0\\ h_2\end{bmatrix}+W_4^0\begin{bmatrix}W_4^0 & W_4^0\\ W_4^1 & W_4^3\end{bmatrix}\begin{bmatrix}h_1\\ h_3\end{bmatrix}\\ \begin{bmatrix}W_4^0 & W_4^0\\ W_4^0 & W_4^2\end{bmatrix}\begin{bmatrix}h_0\\ h_2\end{bmatrix}+W_4^2\begin{bmatrix}W_4^0 & W_4^0\\ W_4^1 & W_4^3\end{bmatrix}\begin{bmatrix}h_1\\ h_3\end{bmatrix}\end{pmatrix}$$



展開計算します

$$=\begin{pmatrix}(W_4^0h_0+W_4^0h_2)+W_4^0(W_4^0h_1+W_4^0h_3)\\ (W_4^0h_0+W_4^2h_2)+W_4^0(W_4^1h_1+W_4^3h_3)\\ (W_4^0h_0+W_4^0h_2)+W_4^2(W_4^0h_1+W_4^0h_3)\\ (W_4^0h_0+W_4^2h_2)+W_4^2(W_4^1h_1+W_4^3h_3)\end{pmatrix}$$

バタフライ演算用に変形します。

$$=\begin{pmatrix}(W_4^0h_0+W_4^0h_2)+W_4^0(h_1+W_4^0h_3)\\ (W_4^0h_0+W_4^2h_2)+W_4^1(h_1+W_4^2h_3)\\ (W_4^0h_0+W_4^0h_2)+W_4^2(h_1+W_4^0h_3)\\ (W_4^0h_0+W_4^2h_2)+W_4^3(h_1+W_4^2h_3)\end{pmatrix}$$

バタフライ演算用にW_4^0,W_4^2の項に変換しました。

展開計算します

2行2項目は、$W_4^0(W_4^1h_1+W_4^3h_3)$

$=(W_4^1h_1+W_4^3h_3)...W_4^0=1$だから

$=W_4^1(h_1+\frac{W_4^3}{W_4^1}h_3)...\frac{W_4^3}{W_4^1}=\frac{W_4^3}{-W_4^3}=-1=W_4^2$から

$=W_4^1(h_1+W_4^2h_3)$

4行2項目は、$W_4^2(W_4^1h_1+W_4^3h_3)$...展開して

$=(W_4^2W_4^1h_1+W_4^2W_4^3h_3)...W_4^3$でくくると

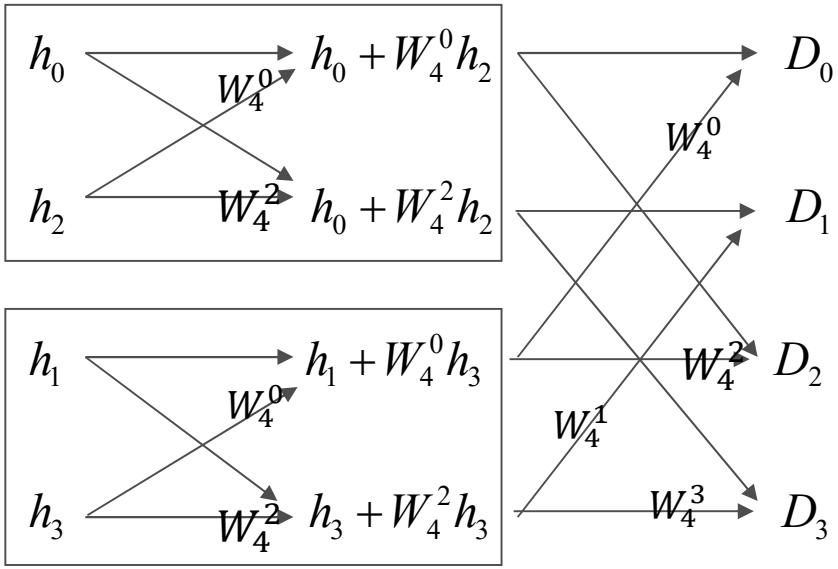
$=W_4^3(\frac{W_4^2W_4^1}{W_4^3}h_1+\frac{W_4^2W_4^3}{W_4^3}h_3)...W_4^3=-W_4^1,W_4^2=-1$から

$=W_4^3(h_1+W_4^2h_3)$

バタフライ演算

前ページより

$$\begin{pmatrix} (W_4^0 h_0 + W_4^0 h_2) + W_4^0 (h_1 + W_4^0 h_3) \\ (W_4^0 h_0 + W_4^2 h_2) + W_4^1 (h_1 + W_4^2 h_3) \\ (W_4^0 h_0 + W_4^0 h_2) + W_4^2 (h_1 + W_4^0 h_3) \\ (W_4^0 h_0 + W_4^2 h_2) + W_4^3 (h_1 + W_4^2 h_3) \end{pmatrix} = \begin{pmatrix} h_0 + W_4^0 h_2 + W_4^0 (h_1 + W_4^0 h_3) \\ h_0 + W_4^2 h_2 + W_4^1 (h_1 + W_4^2 h_3) \\ h_0 + W_4^0 h_2 + W_4^2 (h_1 + W_4^0 h_3) \\ h_0 + W_4^2 h_2 + W_4^3 (h_1 + W_4^2 h_3) \end{pmatrix} = \begin{pmatrix} D_0 \\ D_1 \\ D_2 \\ D_3 \end{pmatrix}$$



バタフライ演算の記法

横矢印は、加算　h_0 → h_0+h_1

h_1 →

矢印上にWがある場合は、

Wをかけて加算　h_0 → W → Wh_0+h_1

h_1 →

矢印の形が蝶の羽のように見えるので、バタフライ演算といいます。重要なのは n=4のDFTがn=2のDFTに分解できることです。

DFTのデータ数と計算量

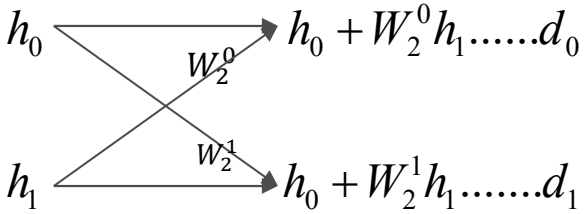
$$\begin{bmatrix} W_2^0 & W_2^0 \\ W_2^0 & W_2^1 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \end{bmatrix}$$

2ポイント(データ数2,n=2)DFTは、データ数nとTFのn行n列の行列演算を行うので、乗算4回と2項の加算2回の計算量になります。

$$d_0 = W_2^0 h_0 + W_2^0 h_1 = h_0 + W_2^0 h_1$$

$$d_1 = W_2^0 h_0 + W_2^1 h_1 = h_0$$

左式をバタフライ演算の形で表現



$$\begin{bmatrix} W_4^0 & W_4^0 & W_4^0 & W_4^0 \\ W_4^0 & W_4^1 & W_4^2 & W_4^3 \\ W_4^0 & W_4^2 & W_4^0 & W_4^2 \\ W_4^0 & W_4^3 & W_4^2 & W_4^1 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ h_2 \\ h_3 \end{bmatrix}$$

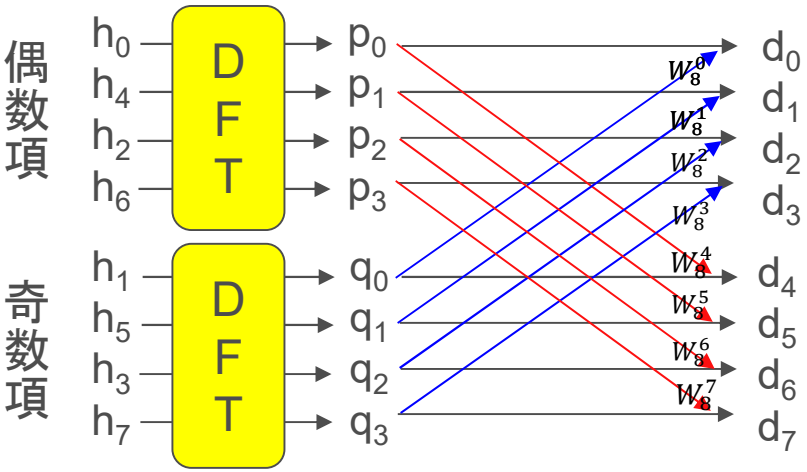
4ポイント(データ数4,n=4)DFTの行列演算では、乗算16回と4項の加算4回の計算量になります。

$$d_0 = W_4^0 h_0 + W_4^0 h_1 + W_4^0 h_2 + W_4^0 h_3$$
$$d_1 = W_4^0 h_0 + W_4^1 h_1 + W_4^2 h_2 + W_4^3 h_3$$
$$d_2 = W_4^0 h_0 + W_4^2 h_1 + W_4^0 h_2 + W_4^2 h_3$$
$$d_3 = W_4^0 h_0 + W_4^3 h_1 + W_4^2 h_2 + W_4^1 h_3$$

同様に8ポイントDFTでは、乗算64回8項加算8回となります

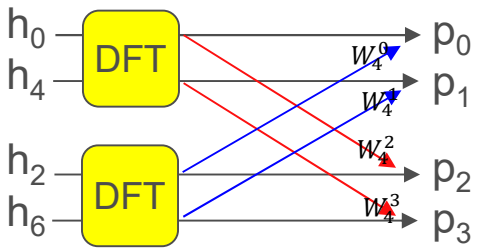
DFTの分解

nが偶数であればnポイントのDFTを偶数と奇数の項に分けてn/2ポイントのDFT2つに分割することができれば、計算量を2^nから、2^(n/2+1)に削減して解くことができます。



左図は、8ポイントのDFTを
偶数と奇数の項をまとめて
それぞれ4ポイントごとのDFT
に分割したものです。
乗算は、64回から40回に
削減されています。

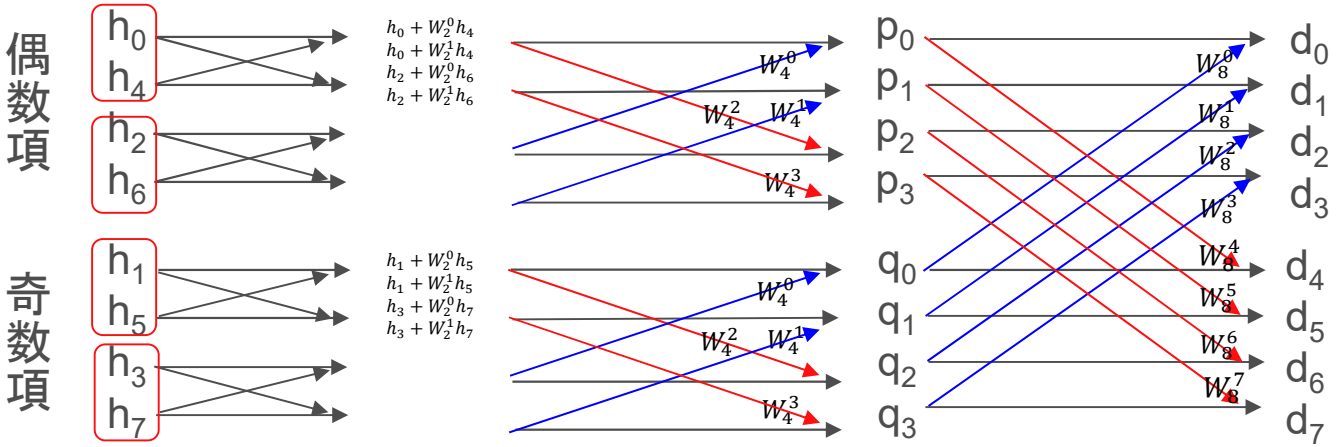
さらにnが2のべき乗の数であれば、上記の分割を繰り返し、2ポイントDFTにまで分解していくことができます。



左図は、偶数項部分の4ポイントのDFTを
2ポイントのDFTに分解した場合です。
奇数項部分も同様に分解できます。

DFTの分割とシグナルフロー

これまでの結果を使って、8ポイントのDFTは、以下のように2ポイントのDFTに分割され、バタフライ演算によって実装されました。乗算は24回となりました。
この図をシグナルフローといいます。

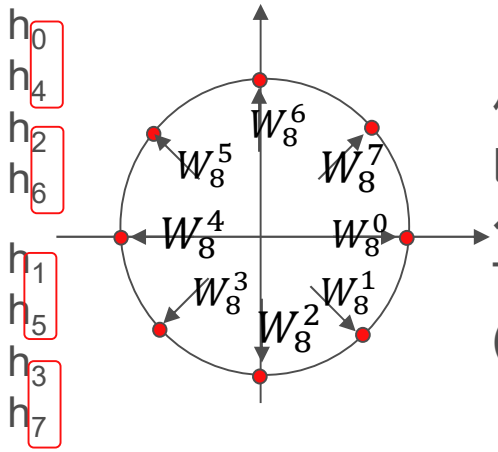


Note

$W_2^0 = W_4^0 = W_8^0$

$W_4^1 = W_8^2$
$W_2^1 = W_4^2 = W_8^4$

$W_4^3 = W_8^6$



偶数と奇数の項に分割したときの項番号の並べ方は、ビットリバーサルという方法で順番をきめます。
ペアになる項の数字は、Wの乗数で、W^nはお互いがTF上で原点に対して対称の位置にあります。
(符号が反転しています。)

バタフライ演算による計算量削減

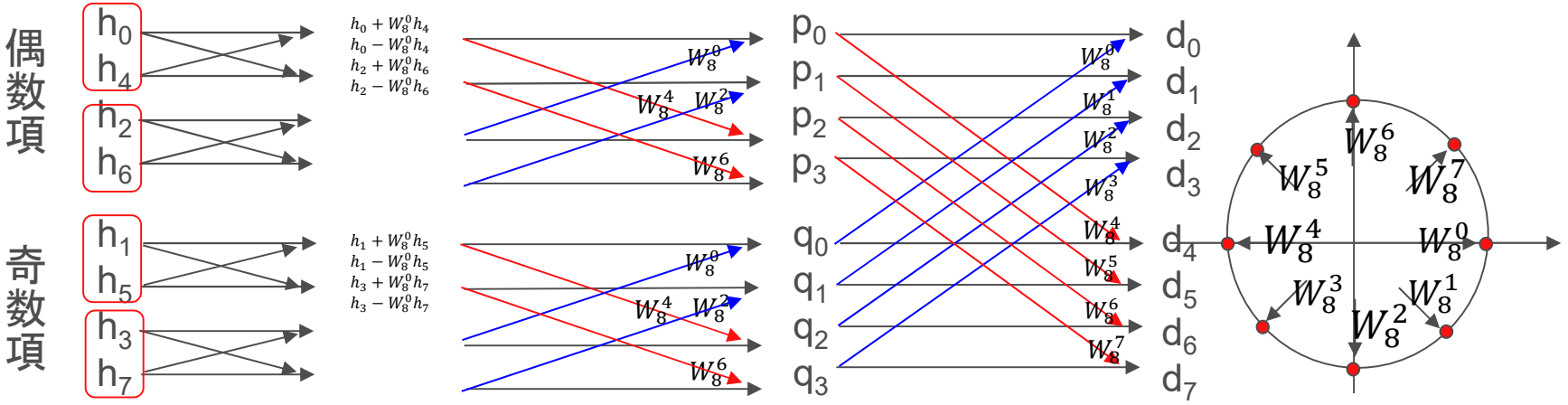
TFの周期性、対象性は

$$W_M^{(\frac{n}{2}+i)k} = e^{-j2\pi\frac{1}{n}(\frac{M}{2}+i)} = e^{-j\pi k}e^{-j2\ \frac{1}{n}(M2ik)} = -W_M^{ik}$$ ですから

$W_2^0 = W_4^0 = W_8^0$
$W_4^1 = W_8^2$
$W_2^1 = W_4^2 = W_8^4$
$W_4^3 = W_8^6$

W_2^α, W_4^βは左のようにW_8^γで表すことができ、またTFの原点対称性により、右のように置き換えることができます。前スライドのシグナルフローは、以下のように変更できます。

$$W_8^4 = -W_8^0$$
$$W_4^6 = -W_8^2$$



バタフライ演算のサブルーチンを増やせば計算量を削減できますが、計算時間とプログラムの複雑さとのトレードオフで決定します。

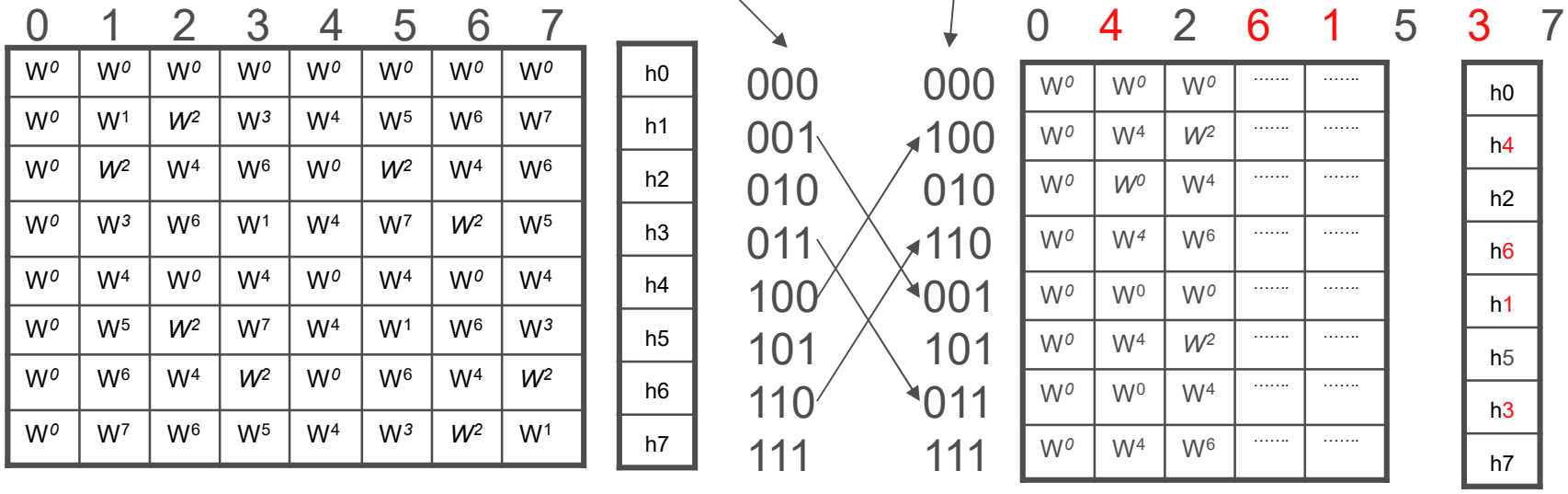
ビット・リバーサル-並べ替え

4項の場合h1とh2の項の入れ替えによって、共通パターンが現れました。この項の並べ替えは、2^nの項に対してビット・リバーサルという方法で行います。

まず、項の番号を通常のバイナリ表示(LSBが右)にします。

続いてLSBが左になるように並べ替えをします。

TF表もこれに従い並べ替えます。



ビット・リバーサル並び替えの詳細については、講義サイト資料で理解を深めてください

ビット・リバーサル-並べ替え演習

並べ変えの練習：

左の行列を右にビットリバーサル並べ替えしてください。

Wは省略して乗数だけでかまいません

000
001
010
011
100
101
110
111

0 1 2 3 4 5 6 7

000 001 010 011 100 101 110 111

0	1	2	3	4	5	6	7	
W^0	W^0	W^0	W^0	W^0	W^0	W^0	W^0	h0
W^0	W^1	W^2	W^3	W^4	W^5	W^6	W^7	h1
W^0	W^2	W^4	W^6	W^0	W^2	W^4	W^6	h2
W^0	W^3	W^6	W^1	W^4	W^7	W^2	W^5	h3
W^0	W^4	W^0	W^4	W^0	W^4	W^0	W^4	h4
W^0	W^5	W^2	W^7	W^4	W^1	W^6	W^3	h5
W^0	W^6	W^4	W^2	W^0	W^6	W^4	W^2	h6
W^0	W^7	W^6	W^5	W^4	W^3	W^2	W^1	h7

.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

ここまでのFFTのまとめ

TFは周期性と対称性をもち、この性質を利用する。

n=4のDFTがn=2のDFTに分解できたが、n=8のDFTは、n=4のDFTに分解できる。これを繰り返すことにより、2^nのDFTは、順次分解を繰り返すことにより計算量を大幅に低減することができる。

計算量は、データの数(項の数)をnとすると、DFTでは、行列演算で乗算がn回とその加算がn回になるため、計算量はn^2に比例して大きくなるが、FFTでは,$n\log_2 n$ですむことになる。

DFTの計算の場合は、データの数に制約はないが、FFTの場合は、

アルゴリズムが使えるのは、$n=2^m$(mは自然数)の場合になる。

効率は落ちるが、2のべき乗に満たないデータ量の場合は、不足のぶんに0を充足してFFT可能にする場合もある。

FFTによる多ビットDFTの分解と項のまとめによるバタフライ演算は、ネスティングとループによるプログラム化ができるので、コンピュータによる処理が容易である。

FFT実装演習

信号処理システムの実装時に、FFT部分をパッケージとして組み込んで使用されますが、その組み込みの過程で、FFTの基礎的な理解が必要となります。

これまでの解説で、FFTの基礎的な理解はできたと思います。続いて具体的なコーディング例をもとに、実装演習を行います。

FFTのアルゴリズムとコーディングは、多種多様なものがあり、今日も研究され、進化が続いています。FFTに限らず、その枠を超えて、各自で独創的なアルゴリズム、とプログラミングを試みてください。

それを具体的に研究成果として発表してください。

サンプル・コード等は、演習サイトに掲示します。

講義編：メモ

メモ

Memo

フォローアップURL (Revised)

http://mikami.a.la9.jp/meiji/MEIJI.htm

担当講師

三上廉司(みかみれんじ)

Renji_Mikami(at_mark)nifty.com

mikami(at_mark)meiji.ac.jp (Alternative)

http://mikami.a.la9.jp/_edu.htm