fft
高速フーリエ変換

1997年 10 月 ver.3 Data Sheet

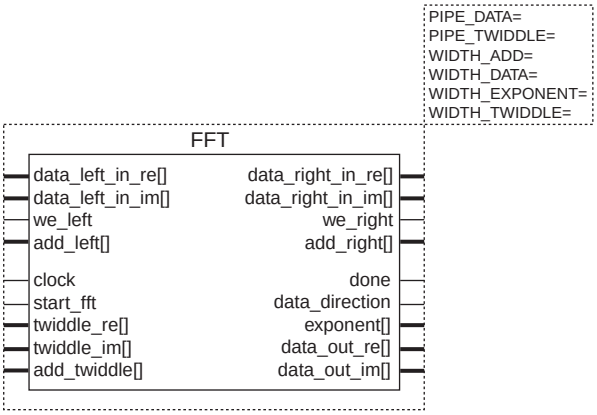
特長

- fftは高速フーリエ変換（FFT）機能を実現したMegaCoreファンクション
- アルテラのFLEX® 10Kアーキテクチャに最適化
- 最高の精度を提供するブロック・フローティング・ポイント方式
- データと回転因子（twiddle）のビット幅、ポイント数をパラメータ化
- デュアル・メモリ・アーキテクチャ
- 複素数フォーマットのデータ入力とデータ出力、周波数分割（Decimation in Frequency）タイプのFFT
- 内部または外部メモリとしても構成可能なデータおよび回転因子（twiddle）メモリとの柔軟性の高いインタフェース

概要

MegaCoreファンクションfftは、信号を構成する各周波数成分に分離するときに使用される高速フーリエ変換の機能を実現したものです。この機能はワイヤレス・コミュニケーション、音声認識、スペクトラム分析、ノイズ解析などを含む多様なディジタル信号処理（DSP）のアプリケーションに使用できます。図１はMegaCoreファンクションfftのシンボルを示したものです。

図１　fft のシンボル



スループットを最適化するため、このファンクションにはデュアル・メモリ・アーキテクチャが採用されており、データは一方のメモリから読み出され、もう一方のメモリに対して書き込みが行われます。このデュアル・メモリ・アーキテクチャはライト（右）、およびレフト（左）の２つのメモリによって構成されています。また、このファンクションは回転因子（twiddle）用のメモリとして３つ目のメモリを使用しています。この回転因子メモリはスループットを最大にするため、ライトおよびレフトのメモリ

A-DS-FFT-03/J
L01-05783-00/J

から分離されている必要があります。これら３種類のメモリはFLEX 10Kのエンベデッド・アレイ・ブロック（EAB）を使用して内部メモリとして構成することができ、また外部のRAMを使用して構成することもできます。

AHDLのFunction Prototype

このfftに対するアルテラ・ハードウェア記述言語（AHDL）のFunction Prototypeは下記の通りになります。

```
FUNCTION fft (clock, start_fft,
    data_left_in_re[WIDTH_DATA-1..0],
    data_left_in_im[WIDTH_DATA-1..0],
    data_right_in_re[WIDTH_DATA-1..0],
    data_right_in_im[WIDTH_DATA-1..0],
    twiddle_re[WIDTH_TWIDDLE-1..0],
    twiddle_im[WIDTH_TWIDDLE-1..0])
  WITH (WIDTH_DATA, WIDTH_TWIDDLE, PIPE_DATA,
    PIPE_TWIDDLE, WIDTH_EXPONENT, FFT_DIRECTION,
    WIDTH_ADD, EXPONENT_INITIAL_VALUE)
  RETURNS (done, data_direction, we_left,
    add_left[WIDTH_ADD-1..0], we_right,
    add_right[WIDTH_ADD-1..0],
    add_twiddle[WIDTH_ADD-2..0],
    data_out_re[WIDTH_DATA-1..0],
    data_out_im[WIDTH_DATA-1..0],
    exponent[WIDTH_EXPONENT-1..0]);
```

VHDLのComponent Declaration（コンポーネント宣言）

fftに対するVHDLのComponent Declarationは下記のようになります。

```
COMPONENT fft
  GENERIC(
    WIDTH_DATA : POSITIVE;
    WIDTH_TWIDDLE : POSITIVE;
    PIPE_DATA : INTEGER;
    PIPE_TWIDDLE : INTEGER;
    WIDTH_EXPONENT : POSITIVE;
    WIDTH_ADD : POSITIVE;
    EXPONENT_INITIAL_VALUE : INTEGER);
  PORT(
    clock : IN STD_LOGIC := '0';
    start_fft : IN STD_LOGIC;
    data_left_in_re, data_left_in_im,
    data_right_in_re, data_right_in_im : IN
    STD_LOGIC_VECTOR(WIDTH_DATA-1 DOWNTO 0);
    twiddle_re, twiddle_im : IN
    STD_LOGIC_VECTOR(WIDTH_TWIDDLE-1 DOWNTO 0);
```

```
        done, data_direction, we_left : OUT STD_LOGIC;
        add_left : OUT STD_LOGIC_VECTOR(WIDTH_ADD-1
        DOWNTO 0);
        we_right : OUT STD_LOGIC;
        add_right : OUT STD_LOGIC_VECTOR(WIDTH_ADD-1
        DOWNTO 0);
        add_twiddle : OUT STD_LOGIC_VECTOR(WIDTH_ADD-2
        DOWNTO 0);
        data_out_re, data_out_im : OUT
        STD_LOGIC_VECTOR(WIDTH_DATA-1 DOWNTO 0);
        exponent : OUT_STD_LOGIC_VECTOR(WIDTH_EXPONENT-1
        DOWNTO 0));
    END COMPONENT;
```

ポート

表１はfftに使用されている各ポート名と機能を示したものです。

表１　fftのポート

名　　称	タイプ	必要性	機　　能
clock	入力	Yes	クロック信号
start_fft	入力	Yes	データのロード後にfftを開始させる信号
data_left_in_re[]	入力	Yes	レフト・メモリからリードされたfftへの実数部データ入力
data_left_in_im[]	入力	Yes	レフト・メモリからリードされたfftへの虚数部データ入力
data_right_in_re[]	入力	Yes	ライト・メモリからリードされたfftへの実数部データ入力
data_right_in_im[]	入力	Yes	ライト・メモリからリードされたfftへの虚数部データ入力
twiddle_re[]	入力	Yes	回転因子メモリからリードされたfftへの回転因子の実数部入力
twiddle_im[]	入力	Yes	回転因子メモリからリードされたfftへの回転因子の虚数部入力
done	出力	Yes	fftが演算を完了した後にHighになる
data_direction	出力	Yes	この出力がHighのとき、fftはレフト・メモリからデータを読み出し、データをライト・メモリに書き込む。また、Lowのときは、その逆の動作を実行する
we_left	出力	Yes	レフト・メモリに対するライト・イネーブル信号
we_right	出力	Yes	ライト・メモリに対するライト・イネーブル信号
add_left[]	出力	Yes	レフト・メモリのアドレス・バス
add_right[]	出力	Yes	ライト・メモリのアドレス・バス
add_twiddle[]	出力	Yes	回転因子メモリのアドレス・バス
data_out_re[]	出力	Yes	レフト・メモリおよびライト・メモリに入力されるfftからのデータ出力の実数部
data_out_im[]	出力	Yes	レフト・メモリおよびライト・メモリに入力されるfftからのデータ出力の虚数部
exponent[]	出力	Yes	最終データの指数部で、doneがHighになった後で有効となる。この指数はブロック・フローティング・ポイントのフォーマットで表示され、すべてのデータは$2^{exponent[]}$でスケーリングされる

パラメータ

表2はfftに提供されているパラメータをまとめたものです。

表2　fftのパラメータ

名　　称	値	説　　明
PIPE_DTA	整数	add_left[]またはadd_right[]がアクティブになってから、data_left_in_re []、data_left_in_im[]、data_right_in_re[]、data_right_in_im[]のポートのデータが有効になるまでのクロック・サイクル数。
PIPE_TWIDDLE	整数	add_twiddle[]がアクティブになってから、twiddle_re[]またはtwiddle_im[]がアクティブになるまでのクロック・サイクル数。
WIDTH_ADD	整数	アドレス・バスのビット幅。fftのポイント数は2^{WIDTH_ADD}になる。
WIDTH_DATA	整数	データ幅
WIDTH_EXPONENT	整数	指数部のビット幅
WIDTH_TWIDDLE	整数	回転因子の幅

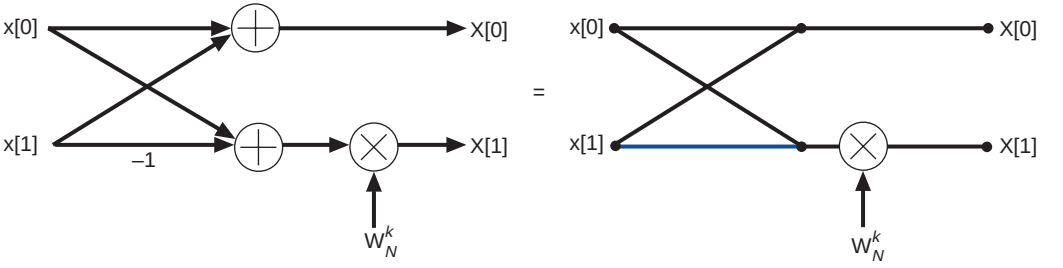
機能の説明

このfftはDecimation In Frequency（DIF）と呼ばれる周波数分割タイプのアルゴリズムを実現しており、FFTの演算に必要とされるすべてのコア・ロジックを含んだものとなっています。最高の柔軟性が得られるようにするために、このfftファンクションにはI/Oインタフェースまたはメモリ・インタフェースが含まれていません。メモリおよびI/Oインタフェースは最終的なアプリケーションで異なるため、各アプリケーションに応じて最適化される必要があります。

図2はDIFアルゴリズムによる基本的なバタフライ演算の動作を示したものです。

図2　DIF FFTの基本バタフライ演算動作

青のラインは負の数を示す。



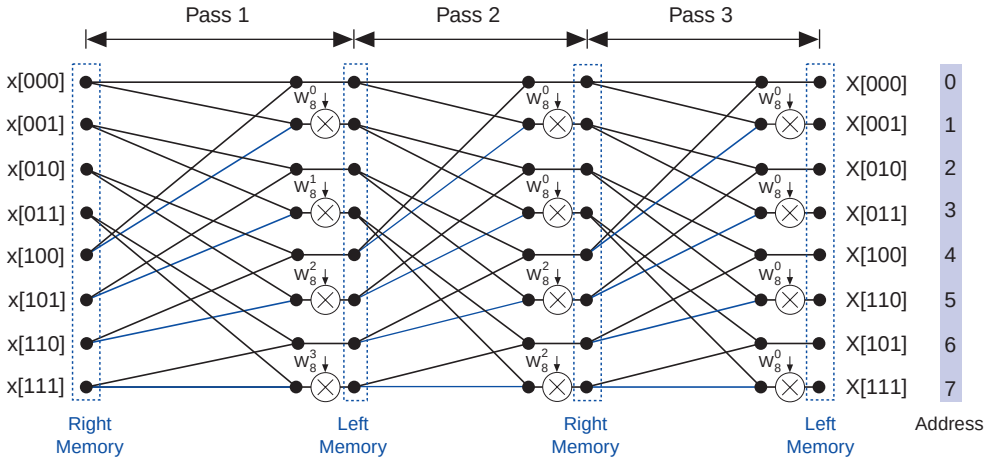
ここで：　X[0] = x[0] + x[1]
X[1] = W_N^k(x[0] – x[1])
W_N^k = Twiddle = $e^{-j\,2\pi k/N} = \cos(2\pi k/N) - j\sin(2\pi k/N)$

k = 0 to ($N/2$ – 1)
N = FFTのポイント数
j = $\sqrt{-1}$

図３は８ポイントのDIF FFTアルゴリズムをfftファンクションで実現した例です。入力データのアドレスは通常の順番になっており、出力データのアドレスはビット・リバースの順番になっています。だたし、専用のハードウェアを追加して、入力と出力の双方に対してデータが通常の順番で現れるようにメモリがアドレスされるようにすることもできます。

図３　８ポイントのDIF FFTアルゴリズム

青のラインは負の数を示す。



☞ fft_on_chipのリファレンス・デザインにはメモリおよびI/Oとのインタフェースと共に、入力と出力の双方に対してデータが通常の順番で現れるようにメモリをアドレスするための専用ハードウェアも含まれています。８ページの例１と10ページの例２には、fft_on_chipのリファレンス・デザインを使用したときの実現方法が解説されています。

データを処理するとき、fftは各パスでリード動作とライト動作を行うメモリを入れ換えます。例えば、図３のPass-1で、fftはライト・メモリからの読み出し動作と、バタフライ演算、そしてその結果をレフト・メモリに書き込む動作を同時に実行します。そして、Pass-2では、fftがレフト・メモリからの読み出し動作と、DIFアルゴリズムの次の演算、そしてその結果をライト・メモリに書き込む動作を同時に実行します。fftはこのプロセスを演算が完了するまで継続して実行します。

fftのポイント数によって、fftがどちらのメモリを最初にリードするかが決定されます。ポイント数が２の奇数乗になっている場合（例：2^3＝８、2^5=32）は、fftが常に最初のパスでライト・メモリに対してリード動作を行います。また、ポイント数が２の偶数乗になっている場合は（例：2^4=16、2^6=64）、fftが各演算の完了後に最初にリード動作を行うメモリを切り換えます。data_directionの出力はfftがどちらのメモリに対してリード動作を行っているか、またはライト動作を行っているかを示します。

回転因子の生成

fftの内部をデータが伝達されるごとに、各データには回転因子（係数）が乗算されます。ここで回転因子（W）は下記の式によって計算されます。

$$W_N^k = e^{(-j2\pi k)/N} = \cos(2\pi k/N) - j\sin(2\pi k/N)$$

ここで　N　＝　fft＝$2^{\text{WIDTH_ADD}}$のときのポイント数
　　k　＝　0 to $(N/2-1)$
　　j　＝　$\sqrt{-1}$

回転因子の実数部は$\cos(2\pi k/N)$となり、虚数部は$-\sin(2\pi k/N)$となります。これら双方の回転因子出力は回転因子メモリにストアされている必要があります。

MegaCoreファンクションfftに添付されているユーティリティ・プログラム、 twiddleはすべての回転因子データを含んだEABメモリ・イニシャライゼーション・ファイル（.mif）を自動的に生成します。 このMIFは他のフォーマットにも簡単に変換することができるようになっており、回転因子メモリに外部ROMを使用する場合にも対応することができます。

シンタックスに関するヘルプを表示させる場合は、UNIXまたはDOSのコマンド・プロンプトから、twiddleと入力してください。

データ表記方法

fftのすべての入出力バスは２の補数の浮動小数点表記となっており、－１から１の範囲の値になります。入力データは常に２の補数の浮動小数点表記となり、バイナリ・ポイントの左側には１ビット、右側にはWIDTH_DATA－１ビットが表示されます。例えば、10進数の0.5は0.1000000_Bとして表され、－0.5は1.1000000_Bとして表記されます。２の補数の浮動小数点フォーマットでの８ビットの最小値は－1.0であり、最大値は下記のようになります。

$$\frac{2^{\text{WIDTHTWIDDLE-1}}-1}{2^{\text{WIDTHTWIDDLE-1}}} = \frac{127}{128} = 0.9922$$

回転因子データは符号反転された２の補数となっています（10進数の0.5は1.1000000_Bとして表され、－0.5は0.1000000_Bと表記される）。もっとも使用される回転因子、ゼロ（W_N^0）は1.0+*j*0の値を持っており、W_N^0が確定した値で表されるときは累積される誤差が少なくなるため、回転因子にはこのフォーマットが使用されます。$W_{N/4}^0$の精度は若干低下しますが、FFTの演算に使用される回転因子が少なくなるため、最終的な結果がより高い精度になります。

メモリ・インタフェース

このfftには、FFTの演算に必要なコア・ファンクションが含まれていますが、メモリやI/Oとのインタフェースは含まれていません。メモリまたはI/Oインタフェースが含まれていないため、ユーザは各アプリケーションに対応したもっとも柔軟性の高いインタフェースを実現することができます。FLEX 10Kデバイスのメモリ容量は限定されているため、すべてのメモリをFLEX 10Kのオン・チップ・メモリで実現することはできません。表３はEPF10K100とEPF10K50デバイスに実現可能なメモリ構成を示したものです。

表３　EPF10K100 と EPF10K50 デバイスに実現可能なメモリ　*注(1)*

デバイス	内蔵EAB数	データ幅（Bits）	回転因子幅（Bits）	メモリ構成					
				回転因子とデータ入力のメモリにEABを使用		回転因子メモリにEAB、データ入力に外部RAMを使用		回転因子に外部RAM、データ入力にEABを使用	
				Points	EABs	Points	EABs	Points	EABs
EPF10K100	12	≤ 8	≤ 8	512	10	2,048	8	512	8
		9 to 16	≤ 8	256	10	2,048	8	256	8
		9 to 16	9 to 16	256	12	1,024	8	256	8
		> 16	> 16	***注(2)***	***注(2)***	***注(2)***	***注(2)***	***注(2)***	***注(2)***
EPF10K50	10	≤ 8	≤ 8	512	10	2,048	8	512	8
		9 to 16	≤ 8	256	10	2,048	8	256	8
		9 to 16	9 to 16	***注(2)***	***注(2)***	1,024	8	256	8
		> 16	> 16	***注(2)***	***注(2)***	***注(2)***	***注(2)***	***注(2)***	***注(2)***

注：
(1)　データと回転因子の双方に外部のRAMを使用した場合、ポイント数に制限のないデザインが作成できます。
(2)　この構成はデバイスに内蔵されているRAMの容量を超えるため、サポートできません。

パイプライン化

メモリ・パスのパイプラインの段数は選択可能になっており、特定のメモリに対してそのサイズとスピードとのトレード・オフを考慮することができます。例えば、例１に示されている「オン・チップRAM、奇数パス、データ・バッファなし」の構成となっているfft_on_chipファンクションでは、データのパイプライン遅延が３段、回転因子のパイプライン遅延が２段となっています。fft_on_chipファンクションの回転因子メモリは入出力がレジスタ付きの同期式ROMとなっているため、PIPE_TWIDDLEのパラメータは２に設定されています。また、データ用のメモリはレジスタ付きの入力（アドレス）と出力（データ）、そしてデータ、アドレスおよびライト・イネーブル入力を切り換えるためのマルチプレクサを持った同期式RAMとなっています。これらのマルチプレクサは最高の性能が得られるようにパイプライン化されており、アドレスとコントロール信号が確定してからデータ・ポート上のデータが確定するまでのパイプライン遅延は３段になります。このため、PIPE_DATAのパラメータは３に設定されています。

構成例

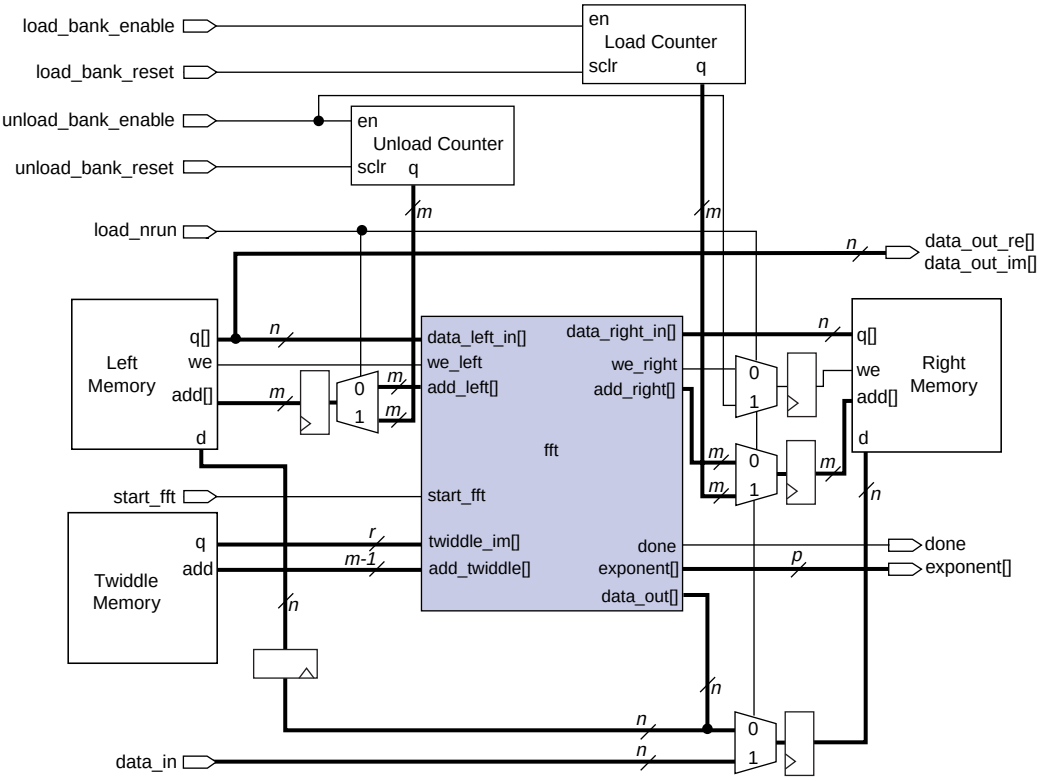
fftを使用した場合、ユーザの主なデザインはメモリとI/Oのインタフェースの実現となります。ユーザは広範囲なメモリとI/Oのインタフェース方法から、システムに最適な構成を選択することができます。このセクションでは、４種類の構成について解説します。

- **例１**： オン・チップRAM、パスの数が奇数（アドレスのビット数が偶数）、データ・バッファなし
- **例２**： オン・チップRAM、パスの数が偶数、データ・バッファなし
- **例３**： 外部RAM、パスの数が奇数、データ・バッファなし
- **例４**： 外部RAM、パスの数が奇数、データ・バッファ付き

例１： オン・チップRAM、パスの数が奇数、データ・バッファなしの構成

この例は、fft_on_chipのリファレンス・デザインに提供されているものと同じメモリ・アーキテクチャとなっています。ライト、レフト、回転因子の各メモリはすべてFLEX 10KのEABに実現されます。データはライト・メモリにロードされ、fftによって処理されます。fftがデータを処理していないときは、双方のメモリに対して同時にデータのロード、アンロードを行うことができます。図４はこの構成を示したブロック・ダイヤグラムです。

図４　例１の構成のブロック・ダイヤグラム



ここで：　*n* = 2 × WIDTH_DATA
m = WIDTH_ADD
p = WIDTH_EXPONENT
r = WIDTH_TWIDDLE

メモリとI/Oのインタフェース

fftの内部ではデータが奇数回のパスで処理されるため、fftは常に新しいデータをライト・メモリから読み込み、結果をレフト・メモリに書き込みます。この方式では、fftとI/Oインタフェースの双方がライト・メモリとレフト・メモリのコントロール・バスおよびデータ・バスをアクセスした場合に問題が発生します。この問題は、データとアドレスのバスにマルチプレクサを配置することで回避することができます。これらのマルチプレクサがパイプライン化されていない場合、これらのマルチプレクサが全体の性能を決定する要因になる可能性があるため、データとアドレスのマルチプレクサの後段にはレジスタが付加されて、パイプライン化が実現されています。この例では、性能を維持するために入出力がレジスタ化されたEABが使用されており、アドレスが確定してからデータが確定するまでのパイプラインの遅延は３段となります。このため、PIPE_DATAのパラメータは３に設定されています。

ロードおよびアンロードのアドレッシング

データは通常のビットの順番でfftに入力され、その出力データがビット・リバースの順番でレフト・メモリに書き込まれます。メモリに対するロード、アンロードを行うために1個または2個のカウンタを設けることが可能です。1個のカウンタを使用する場合は、ライト・メモリがロード・カウンタの出力を通常のビットの順番で受信し、レフト・メモリはロード・カウンタの出力をビット・リバースの順番で受信します。I/Oインタフェースが1個のアドレス・カウンタを持っている場合は、ロードとアンロードは同時に完了しなければなりません。また、図4のように2個のカウンタを使用すると、doneの信号がアサートされている状態でロードとアンロードを個別に行うことができます。

回転因子メモリ

回転因子メモリは常時、リード・オンリとなるため、そのメモリ・インタフェースのデザインはシンプルなものとなります。回転因子メモリにはライト・イネーブル信号やI/Oインタフェースが必要なく、回転因子メモリをfftにダイレクトに接続することができます。回転因子メモリの入出力はレジスタ付きとなっているため、PIPE_TWIDDLEのパラメータは2に設定されています。

使用方法

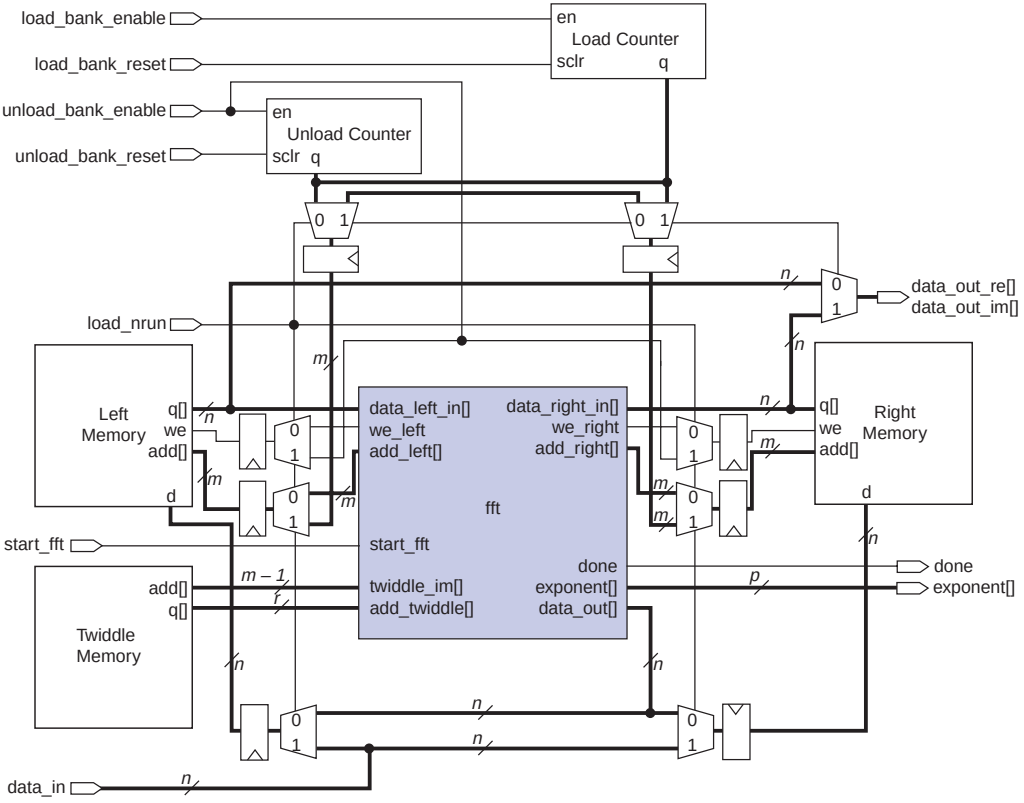
ライト・メモリにデータをロードするときは、load_nrunをN+1クロック・サイクルの期間にわたってアサートし、load_bank_enableをNクロック・サイクルの期間与えながらデータをdata_inのポートに入力します。次にload_nrun入力をディアサートしてfftのコントロールに戻します。次のクロック・サイクルでstart_fftを1クロック・サイクルの期間だけアサートし、計算を開始させます。fftがデータの処理を完了し、計算結果をレフト・メモリに書き込んだ後で、fftはdoneの信号を出力します。

例2： オン・チップRAM、パスの数が偶数、データ・バッファなしの構成

例2の構成はfftが偶数のパスでデータを処理する点を除き、例1に類似したものとなっています。この方式でのスループットを最大にするため、fftは各FFTの演算の完了後に最初のリード動作を行うメモリを入れ換えます（fftは最初のFFTの演算を行う場合にライト・メモリから新しいデータをリードし、2回目のFFTの演算を行うときはレフト・メモリから新しいデータをリードする）。データがfft内部で偶数のパスで処理される場合、fftは新しいデータを一方のメモリから読み込み、処理した後で、その結果を同じメモリに戻します。この場合、各メモリは新しいデータか計算結果のいずれかをストアしていることになるため、I/Oインタフェースは新しいデータを一方のメモリにロードし、もう一方のメモリから結果をアンロードできるようになっている必要があります。

図５は例２の構成をブロック・ダイヤグラムで示したものです。この例では、ロードとアンロードのアドレスの双方でライトとレフトのメモリをアクセスできるようにするために、マルチプレクサが使用されています。このマルチプレクサを使用することによって、双方のメモリに対してデータをロードおよびアンロードすることが可能になります。性能を維持するためにマルチプレクサはパイプライン化されているため、アドレス・カウンタとレフトおよびライト・メモリとの間には１段のパイプライン遅延が追加されます。また、load_bank_enable、data_in、left_address、right_addressの各信号は同じクロック・サイクルですべてアクティブとなるため、data_inのパスにはさらに１段のパイプライン遅延が追加されます。

図５　例２の構成のブロック・ダイヤグラム



例３： 外部RAM、パスの数が奇数、データ・バッファなしの構成

RAMを外部に設けることで、fftのポイント数を256以上にすることが可能になります。外部に同期型のスタティックRAM（SSRAM）を使用することによって、高速のクロック・レートを維持しながら、fftを使用してさらに大きなポイント数のFFTファンクションを構成することができます。この例に使用されるSSRAMは高速のマイクロプロセッサ用に提供されている同期型のキャッシュRAMです。この種のRAMは多くのベンダから供給されています。

このアーキテクチャは、外部にRAMを使用している点を除けば、例１と例２に使用されているものと類似しています。データ、アドレスのマルチプレクス、パイプライン化の方式などは例１と例２の場合と同じになっています。例１や例２との違いは、RAMがdata_inとdata_outのポートを個別に持っておらず、双方向のポートとなっている点だけです。このため、このデザインにはFLEX 10Kデバイスの双方向ピンを使用して、これらのピンが書き込み時には出力ピンに、それ以外のときは入力ピンとなるようにする必要があります。

例４： 外部RAM、パスの数が奇数、データ・バッファ付きの構成

例１、例２、および例３の構成では、 fftがデータを処理しているときにすべてのRAMが使用されるため、 新しいデータをfftに取り込むことはできません。この例では、４つのメモリ（ライト-１、ライト-２、レフト-１、レフト-２の４種類のメモリ）を使用して、fftがデータを処理中にも新しいデータが取り込めるようにしています。fftがライト-１メモリとレフト-１メモリからのデータを処理しているときは、新しいデータがレフト-２メモリにロードされます。レフト-２メモリがフルになり、fftがデータの処理を完了した状態になったときに、レフト-１メモリとレフト-２メモリがスワップされ、ライト-１メモリとライト-２メモリがスワップされます。次にstart_fftがすぐにアサートされ，レフト-２とライト-２からのデータに対する処理が開始されます。fftがレフト-２メモリとライト-２メモリからのデータを処理しているときは、レフト-１メモリとライト-１メモリにロードおよびアンロードの動作が行われます。

日本アルテラ株式会社

〒163-0436
東京都新宿区西新宿2-1-1
新宿三井ビル私書箱261号
TEL. 03-3340-9480 FAX. 03-3340-9487
http: / / www.altera.com / japan /

本社 Altera Corporation

101 Innovation Drive,
San Jose, CA 95134
TEL : (408) 544-7000
http://www.altera.com