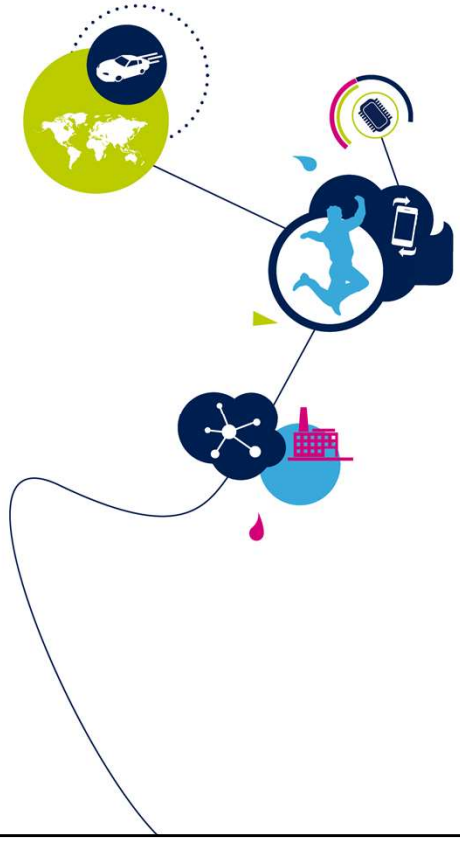
STM32G4 - FMAC

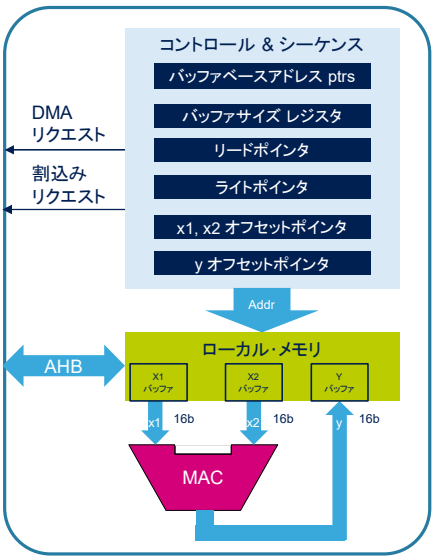
フィルタ数値演算アクセラレータ

1.0版



こんにちは、STM32G4 FMACブロックのプレゼンテーションへようこそ。
自律的にバックグラウンドで信号処理フィルタリングタスクを実行するために使用されるこのブロックの主な特徴について説明します。

概要



- The Filter Math ACcelerator（FMAC）ユニット
  - 乗算/アキュムレータ（MAC）ユニットとサーキュラ·バッファを使ったデジタル·フィルタを搭載（入出力）
- 乗算アキュムレータ（MAC）ユニット
  - 16 x 16ビット乗算
  - 加算と減算が可能な24 + 2ビットアキュムレータ

アプリケーション側の利点

- 固定小数点
- 有限および無限の衝動応答フィルタの両方を実装
- 256 x 16ビットローカル·メモリ

FMACユニットは固定小数点乗算器とアキュムレータ（MAC）を中心に構築されています。
MACユニットは、内部の256x16ビットRAMから2つの固定小数点16ビットオペランドを受け取り、このメモリに結果を書き戻します。

ローカルメモリ内の入力値のアドレスは、ポインタをセットし使用します。
これらのポインタは、内部ハードウェアによってロード、インクリメント、デクリメント、またはリセットできます。
ソフトウェアはそれらに直接アクセスしません。
このユニットは、頻繁にある、または長時間のフィルタリング操作をCPUからオフロードできるようにし、プロセッサを他のタスクに解放します。

機能

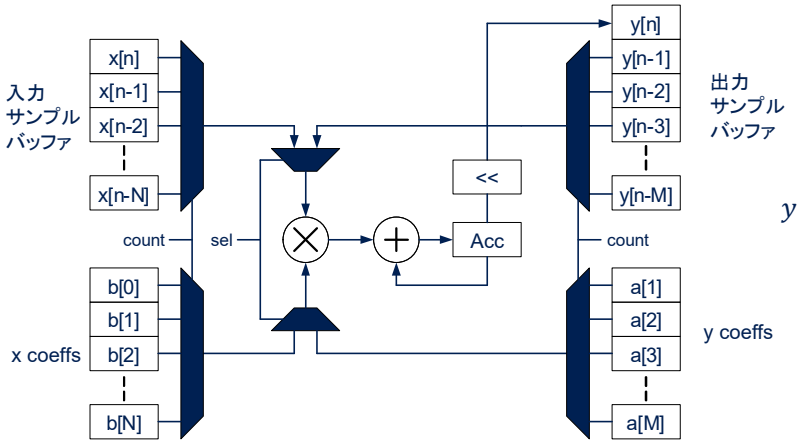
- 有限インパルス応答（FIR）と無限インパルス応答（IIR）フィルタの両方を実現可能
- ターゲット・アプリケーション
  - モータ制御、オーディオ、デジタル電源、照明、アナログ・センシング（ヘルス、フィットネス、ロボティクス、….）
- CPUのオフロード
  - バックグラウンドで信号のフィルタリング・タスクを自律的に実行
  - CPUのMIPSを他のタスクへ開放するため、CPUの処理を最小限に
- 主な目的
  - フィルタの種類、順番、Coeffはプログラミング可能

life.augmented

フィルタ機能FIRとIIRはFMACによって実現することができます。
これらのフィルタを必要とする一般的なアプリケーションは、モータ制御、オーディオ、電源、照明、アナログセンシングです。
FMACは、バックグラウンドの信号のフィルタリングタスクを自律的に実行し、CPUからオフロードし、他のタスクへCPUのMIPSを解放します。
FMACユニットを使用すると、ユーザーはフィルタタイプ、フィルタの順序、Coeff係数をプログラミングできます。

シングルMACアーキテクチャ

- 積の合計は、各出力サンプルに対するN+M+1乗算累積（MAC）演算となる
- コストを節約するために、1つのMACが繰り返し使用



$$y[n] = \sum_{k=0}^{N} b[k]x[n-k] + \sum_{j=1}^{M} a[j]y[n-j]$$

この図は、MACユニットのアーキテクチャの詳細を示しています。
Xは、フィルタリングするローサンプルを含む入力サンプルバッファです。
Bは、Xサンプルに適用されるフィルタのCoeffの配列です。
XとBのサイズは同じです。：N + 1エントリ。
Yは、フィルタリングの結果を含む出力サンプルバッファです。
Aは、Yサンプルに適用されるフィルタの係数の配列です。
YとAのサイズは同じです。：M + 1エントリ。
y[n] =（N + 1）+ Mを取得するMAC操作の数：
·累積ベクトルXとベクトルBを乗算するN + 1 MAC
·累積ベクトルY [n−1：n−M]とベクトルAを乗算するM個のMAC

データフォーマット

- FMACの入力と出力は、固定小数点符号付き整数q1.15フォーマットを使用
  - q1.15フォーマットでは、数値は1つの符号ビットと15個の小数ビット（2進数）で表す
    - したがって、数値範囲は-1 (0x8000)から1 -2^15 (0x7FFF)

q1.15フォーマット

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
S	c_{14}	c_{13}	c_{12}	c_{11}	c_{10}	c_9	c_8	c_7	c_6	c_5	c_4	c_3	c_2	c_1	c_0

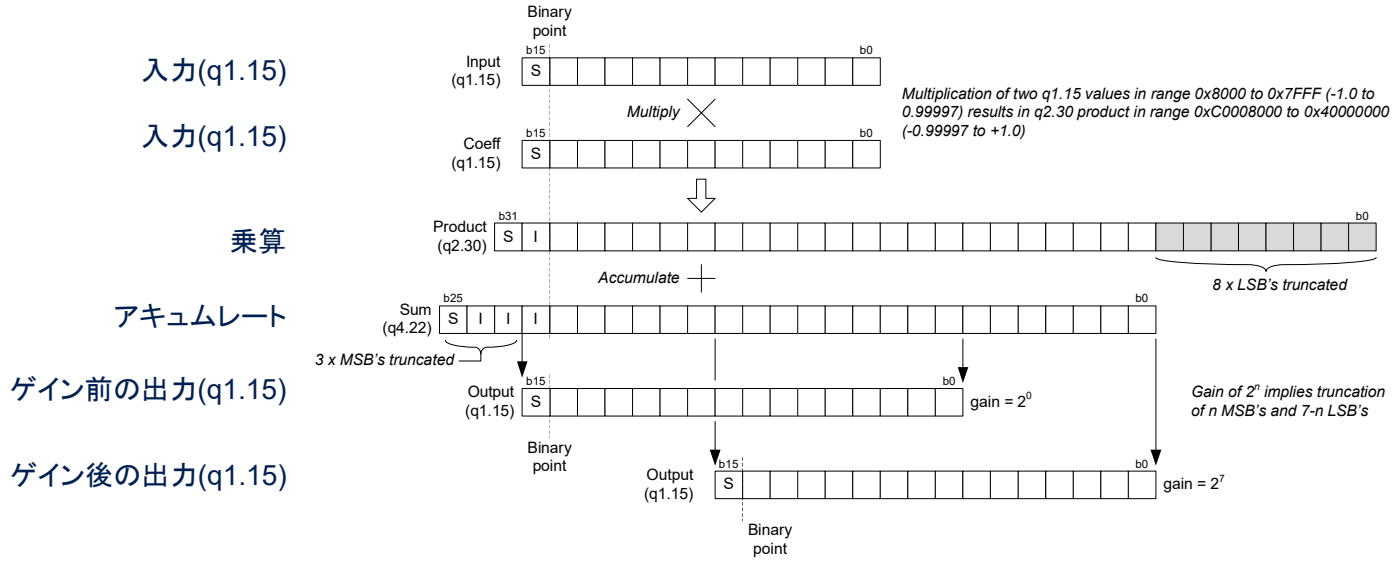
Fractional_number= $(-1)^s \sum_{k=0}^{14} \frac{1}{2^{15-k}} c_k^k$

- 32ビット浮動小数点数は、ソフトウェアの乗算とキャストによって固定小数点に変換可能（Cortex®-M4 FPUを使用）:
  - value_q15 = (int16_t)(value_f32*0x8000); /*f32からq1.15変換–8サイクル必要*/
  - value_f32 = (float)value_q15/(float)0x8000; /*q1.15からf32変換–10サイクル必要*/

life.augmented

FMACの入出力は、固定小数点符号付き整数q1.15形式を使用します。
q1.15形式では、数値範囲は1（0x8000）～1-2^15（0x7FFF）です。
32ビットの単精度浮動小数点数は、Cortex-M4 FPUで実行される専用の変換命令によってq1.15形式との間で変換できます

データフォーマット

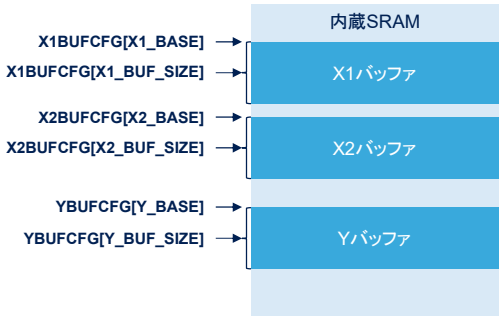


この図は、FMACによって内部的に使用されるさまざまな形式の詳細を示しています。
乗算器の出力（q2.30フォーマット）はq2.22に切り捨てられ、アラインメントされたLSBに追加されます。
アキュムレータには26ビットがあり、そのうち22ビットは小数で、4ビットは整数/符号（q4.22）です。
余分な整数ビットにより、アキュムレータは-8（0x4000000）～+8（0x3FFFFFF）の範囲の部分的な累積合計をサポートできます。
これは、連続する正または負の係数が多数ある場合に発生する可能性があります。
すべての周波数でフィルタゲインが1未満の場合、アキュムレータ値は常に範囲+/-1に戻ります。
部分的な合計がアキュムレータの数値範囲を超える場合（ラップ）、スティッキフラグが設定されてデバッグを支援します。
それでも、その後の追加でラッピングが取り消される場合でも、正しい結果が得られます。

アキュムレータの出力にプログラム可能なゲインを適用できます。
6dBのステップで0dBから42dBまで。
これは、IIRフィルタの実装に必要です。

データバッファ

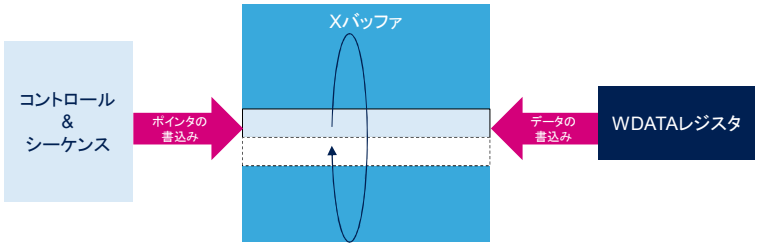
- フィルタ数値演算アクセラレータ･ユニットは、ベクトルに対して算術演算を実行
- データ･バッファ（2つの入力、1つの出力）のメモリ内に最大3つの領域を定義
  プログラム可能なベースアドレス･ポインタおよび関連するサイズ･レジスタによって定義される



FMACユニットは、16ビットの固定小数点スカラー値の配列であるベクトルに対して算術演算を実行します。
これらのベクトルはローカルSRAMに割り当てられます。
ソフトウェアは、X1BUFCFG、X2BUFCFG、およびYBUFCFGレジスタを介してX1およびX2オペランドバッファとY出力バッファを構成します。
すべてのバッファが内部メモリアドレスの範囲（0x00～0xFF）に収まる場合、ベースメモリは内部メモリのどこでも選択できます。
バッファのベースアドレスとサイズをプログラムする必要があります。
X1、X2、およびYバッファは重複する場合があることに注意してください。
これらのバッファは、CPUマッピングでは表示されません。

入力バッファの初期化

- CPU（またはDMAコントローラ）は、初期化関数を使用して各バッファの内容を初期化し、書込みデータ·レジスタに書き込む
  - 操作の前にベクトルの要素をロード、フィルタとロード·フィルタのCoeffを初期化するために使用



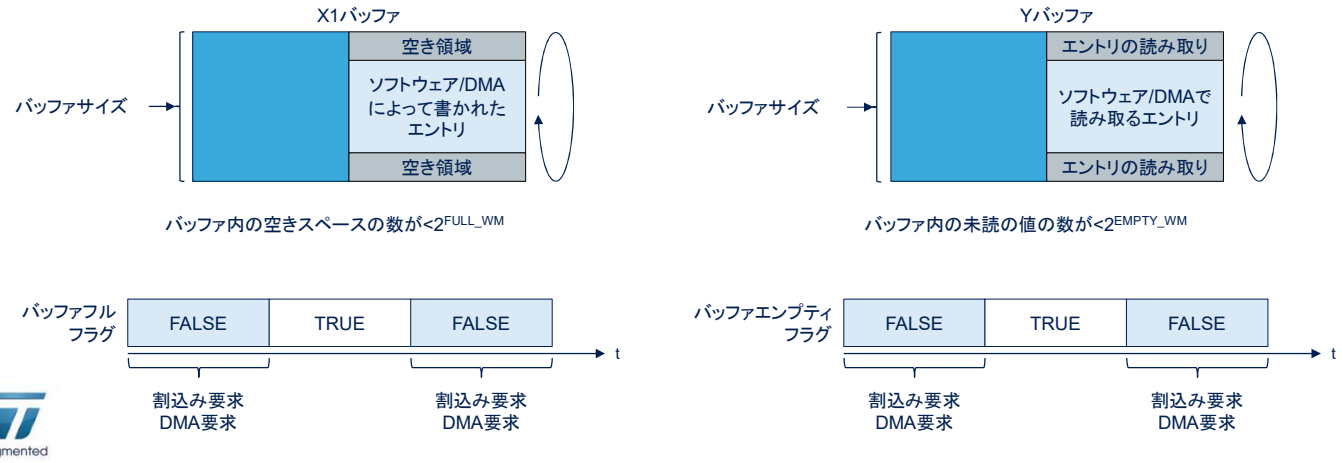
life.augmented

フィルタ操作を開始する前に、CPUまたはDMAコントローラは、初期化関数を使用してWDATAレジスタに書き込むことにより、入力バッファの内容を初期化します。
入力バッファの内容は、フィルタリングされるデータまたはフィルタのcoeffのいずれかです。
データは、書き込みポインタによって示されるターゲットバッファ内の場所に転送されます。
新しい書込みのたびに、書込みポインタがインクリメントされます。
書込みポインタは、割り当てられたバッファスペースの最後に到達すると、ベースアドレスに戻ります。

サーキュラ·バッファ

	X1バッファ	X2バッファ	Yバッファ
サーキュラ·バッファ オペレーション	オプション	N/A	オプション

- CPUまたはDMAアクティビティを調整するため、ウォータマーク·レベルの設定可能

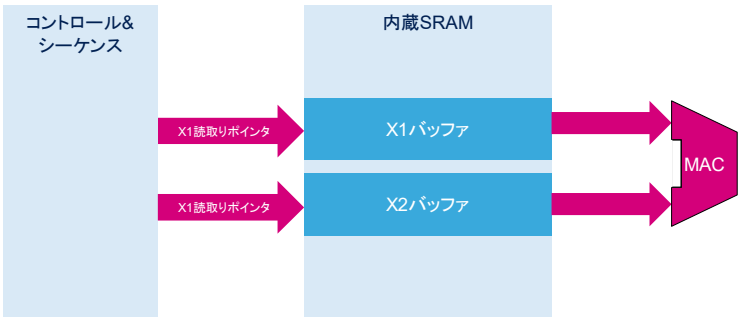


X1バッファに関して、バッファ内の空きスペースの数がFMAC_X1BUFCFGレジスタのFULL_WMフィールドにプログラムされたウォーターマークのしきい値よりも少ない場合、バッファはフルとしてフラグが立てられます。
フルフラグが設定されていない限り、割込みまたはDMA要求有効になっている場合、バッファの追加データを要求するために生成されます。
このウォーターマークにより、オーバーフローの危険なしに、1回の割込みで複数のデータを転送できます。
それでも、オーバーフローが発生した場合、OVFLエラーフラグが設定され、書き込みデータは無視されます。
オーバーフローが発生しても、書き込みポインタは増加しません。

Yバッファについては、バッファ内の未読データの数がFMAC_YBUFCFGレジスタのEMPTY_WMフィールドにプログラムされたウォーターマークのしきい値よりも少ない場合、バッファは空としてフラグが付けられます。
空のフラグが設定されていない限り、割込みまたはDMA要求が生成され、有効になっている場合バッファからの読み取りを要求します。
ウォーターマークにより、アンダーフローの危険なしに、1つの割込みで複数のデータを転送できます。
それでも、アンダーフローが発生すると、UNFLエラーフラグが設定されます。
この場合、読み取りポインタはインクリメントされず、読み取り操作は読み取りポインタアドレスのメモリの内容を返します。

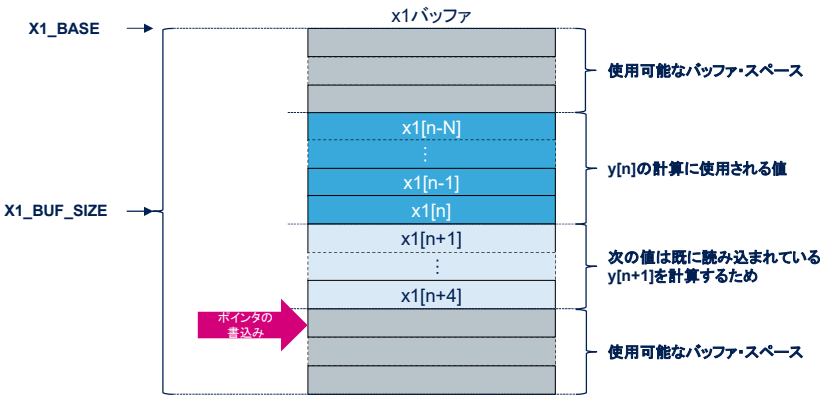
入力バッファ

- X1およびX2バッファは、MACへの入力用のデータを格納するために使用される
  - コントロール・ユニット内のポインタは、各値の読み取りアドレス・オフセット（バッファ・ベース・アドレスに対する相対）を生成
  - ポインタは、現在の関数に従ってハードウェアによって管理される



各乗算は、X1バッファから値を受け取り、そしてX2バッファから値を受け取り、それらを乗算します。
コントロールユニット内のポインタは、各値の読み取りアドレスオフセット（バッファベースアドレスに相対する）を生成します。
ポインタは、現在の関数に従ってハードウェアによって管理されます。

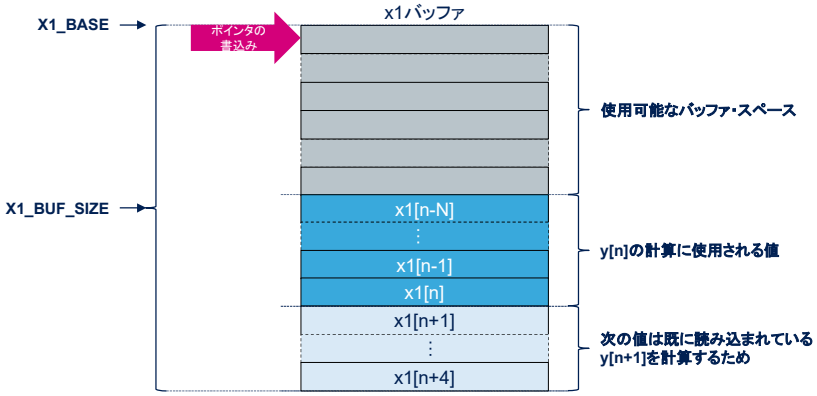
入力データX1バッファ



- フィルタは、出力サンプルy[n]を計算するためにN+1サンプル（入力セット）のセットを使用
- 計算が完了すると、次のサンプル（x1[n+1]）が入力セットに追加
  最も新しいサンプル（x1[n-N]）が削除され、バッファ内の領域が解放される
- 入力データ・レジスタに到着した新しいサンプルは、書込みポインタのバッファに書込まれる
  - これは常に、入力セット内の最新のサンプルx[n]の後の最初の使用可能なスペースを示す

この図は、X1バッファ操作を説明しています。
書き込みポインタがバッファの最後に到達すると、先頭に戻ります。
バッファ内の使用可能なスペースが転送サイズより小さい場合、入力バッファのフルフラグがアクティブになります。
入力セットの最上部であるx[n]が書き込みポインタに等しい場合（つまり、利用可能な新しいサンプルがない場合）、新しいサンプルが利用可能になるまでフィルタは停止します。
プロセッサ、またはDMAコントローラは、必要に応じて新しいサンプルx[n + 1]がバッファスペースで利用できるようにする必要があります。
そうでない場合、バッファには空のフラグが付けられ、新しいサンプルが追加されるまでユニットの実行が停止します。
X1バッファではアンダーフロー状態は通知されません。

入力データX1バッファ



- X1バッファはサーキュラ・バッファとして使用可能
- このバッファの事前読込みは、デジタル・フィルタではオプション
  - プリロードは、ベクトル演算の場合には役に立つ

life.augmented

X1バッファは、サーキュラバッファとして使用できます。
スペースが利用可能になると、新しいデータが入力バッファに継続的に転送されます。
書き込みポインタは、図に示すように、バッファ内の最後の16ビットエントリに到達すると自動的に折り返されます。
操作の開始時に入力サンプルがバッファに書き込まれていない場合は空としてフラグが付けられ、操作を開始するのに十分になるまでCPUまたはDMAをトリガして新しいサンプルをロードするため、デジタルフィルタの場合、このバッファのプリロードはオプションです。
それにもかかわらず、プリロードはベクトル演算の場合に役立ちます。つまり、入力データはすでにシステムメモリで利用可能であり、サーキュラ演算は必要ありません。

入力データX2バッファ

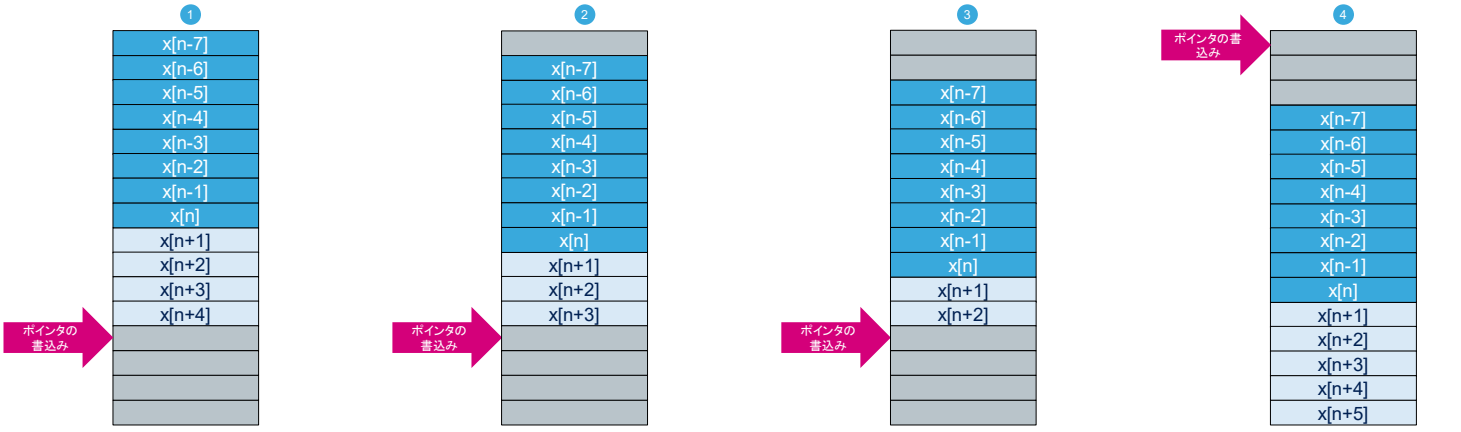
- X2バッファはベクトル·モード（サーキュラ·モードではない）でのみ使用でき、バッファの内容が1つの操作から次の操作に変更されない場合を除き、プリロードする必要がある
  - フィルタ関数の場合、X2バッファを使用してフィルタのCoeffを格納

life.augmented

X2バッファは、Coeffを格納するために使用されます。
これは通常、FMACの初期化時に1回ロードされます。
したがって、それはサーキュラアドレッシングモードをサポートしていません。

入力バッファX1オペレーション

- N=8および4サンプルDMA転送を使用したフィルタ図



8サンプルx[n-7]~x[n]を使用してフィルタ処理
- 次の4つのサンプルは既に読込まれている
- 4つのバッファ位置が利用可能
- フルフラグが設定されていない

n ← n+1
フィルタは現在の出力サンプルを終了し、最も古いサンプルは不要になるためバッファ内の次のサンプルを使用して領域を解放

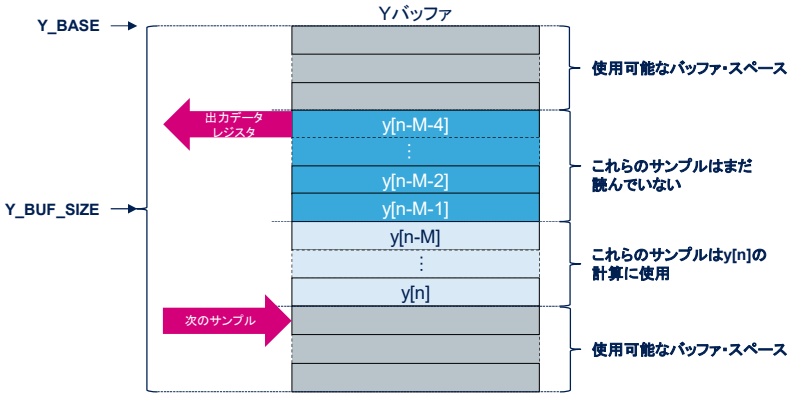
n ← n+1
フィルタは現在の出力サンプルを終了し、最も古いサンプルは不要になるためバッファ内の次のサンプルを使用して領域を解放

バッファに書込まれた4つの新しいサンプル
- 4ずつインクリメントされた書込みポインタ
- 書込みポインタが末尾に達すると、先頭に折り返される

life.augmented

この図は、入力バッファの動作をまとめたものです。
ステップ1の間、フィルタはx[n−7]からx[n]までy[n]を計算し、次の4つのサンプルを読み込みます。
ステップ2の間にy[n]が計算されるようになると、サンプルx[n−7]が削除されます。次に、nがインクリメントされます。フィルタは、y[n]をx[n−7]からx[n]に計算します。新しいサンプルは読み込まれません。
ステップ3の間にy[n]が計算されるようになると、サンプルx[n−7]が削除されます。次に、nがインクリメントされます。フィルタは、y[n]をx[n−7]からx[n]に計算します。新しいサンプルは読み込まれません。
ステップ4の間にy[n]が計算されるようになると、サンプルx[n−7]が削除されます。次に、nがインクリメントされます。フィルタは、y[n]をx[n−7]からx[n]に計算します。4つの新しいサンプルが読み込まれます。バッファの上のアドレスに達したので、バッファの先頭に折り返しが発生します。

出力サンプル・バッファ



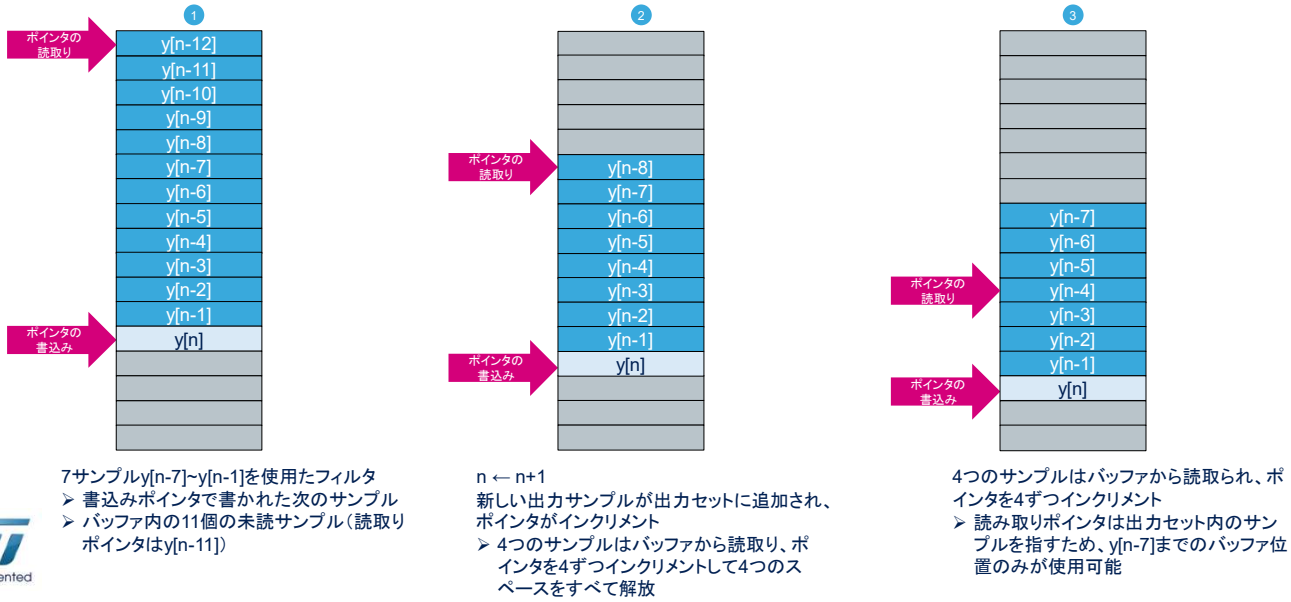
- Y（出力）バッファは、アキュムレータの出力を格納するために使用
  - 新しい各出力値は、プロセッサまたはDMAコントローラによって読み取られるまでバッファに格納
- フィルタは、出力サンプルy[n]を計算するためにM出力サンプル（出力セット）のセットを使用
- 計算が完了すると、新しいサンプル（y[n]）がセットに追加され、最も新しいサンプル（y[n-M]）がセットから削除される
- 新しいサンプルは書込みポインタのバッファに書き込まれる
  - これは常に、セット内の最新のサンプルy[n-1]の後の最初の空き領域を示す
- Yバッファは、サーキュラ・バッファとしても動作

life.augmented

この図は、Yバッファの動作を説明します。
書き込みポインターがバッファの末尾に達すると、先頭に折り返されます。
リードポインタは、データが出力レジスタに対応し、最も古い未読み取りサンプルを指示します。
サンプルが読み取られ、出力セットの一部ではない場合、スペースは空になります。
書き込みポインタが読み取りポインターまたは出力セット内の最も最新でないサンプル（y[n−M]）と等しい場合、フィルタは停止し、出力バッファのフルフラグが設定されます。

出力バッファ・オペレーション

- M=7および4サンプルDMA転送を使用したIIRフィルタ



7サンプルy[n-7]~y[n-1]を使用したフィルタ
- 書込みポインタで書かれた次のサンプル
- バッファ内の11個の未読サンプル（読取りポインタはy[n-11]）

n ← n+1
新しい出力サンプルが出力セットに追加され、ポインタがインクリメント
- 4つのサンプルはバッファから読取り、ポインタを4ずつインクリメントして4つのスペースをすべて解放

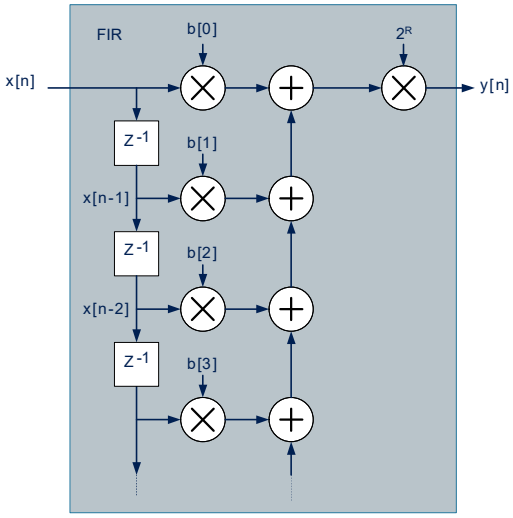
4つのサンプルはバッファから読取られ、ポインタを4ずつインクリメント
- 読み取りポインタは出力セット内のサンプルを指すため、y[n-7]までのバッファ位置のみが使用可能

life.augmented

この図は、出力バッファの動作をまとめたものです。
ステップ1の間、フィルタはy[n−7]からy[n−1]までy[n]を計算します。11個のサンプルが未読です。
ステップ2の間にnがインクリメントされると、フィルタはy[n−7]からy[n−1]までy[n]を計算します。ソフトウェアまたはDMAは4つのサンプルを読取り、読取りポインタは最も古いサンプルに移動します。
ステップ3の間にnがインクリメントされると、フィルタはy[n−7]からy[n−1]までy[n]を計算します。ソフトウェアまたはDMAは4つのサンプルを再度読取り、読取りポインタは最も古いサンプルに移動します。ただし、サンプルy[n−7]~y[n−5]は現在の計算で使用されるため、割り当て解除されません。

汎用デジタル・フィルタ

有限インパルス応答(FIR)フィルタ、ダイレクトフォーム1ストラクチャ



- 有限インパルス応答(FIR)デジタルフィルタは、入力サンプリング・データストリームx[n]と係数b[k]のセットとの間の畳み込み

$$y[n] = 2^R \sum_{k=0}^{N} b[k]x[n-k]$$

バッファ	利用
X1	サンプリングされたデータ
X2	フィルタCoeff
Y	出力値

パラメータ	説明
P	ベクトルBの長さ
Q	未使用
R	出力に適用されるゲイン

FIR関数は、フィルタ係数を含む長さN+1のベクトルBと、サンプリングされたデータを含む無期限の長さのベクトルXの畳み込みを実行します。
FMACでFIRを実装するには、バッファを次のように使用します。
·X1バッファにはベクトルXの要素が含まれます。長さN+1+dのサーキュラバッファである。
·X2バッファにはベクトルBの要素が含まれます。長さN+1の固定バッファである。
·Yバッファには出力値ynが含まれます。長さdのサーキュラバッファである。
ここでパラメータは次のとおりです。
·パラメータPには、CoeffベクトルBの長さN+1が範囲[2:127]で含まれます。
·パラメータRには、アキュムレータ出力に適用されるゲインが含まれます。Yバッファに出力される値は2^Rで乗算され、Rは範囲内にある[0:7]
·パラメータQは使用されません。
この機能は、FMAC_PARAM・レジスタ内のSTARTビットがソフトウェアによってリセットされると完了します。

汎用デジタル・フィルタ

有限インパルス応答(FIR)フィルタ、ダイレクトフォーム1ストラクチャ

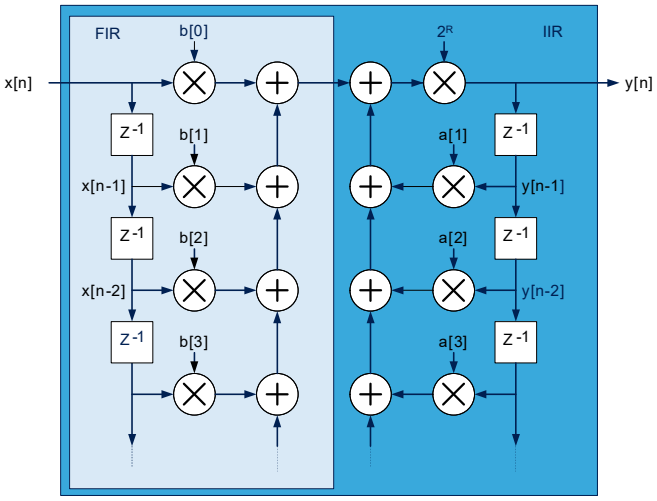
- メモリ・リクアイアメント
  - N CoeffとN個の入力サンプルを計算
  - 各入力サンプルはN回使用
  - スループットを最適化するには、入力バッファ・サイズをN + εにする必要がある
    - これにより計算に使用されていない間に1つのDMA要求でεサンプルを読み込むことが可能
  - 出力バッファ・サイズは、DMA転送サイズεと一致するようにする必要がある
  - 合計メモリ・リクアイアメント:2(N + ε)256

life.augmented

FIRでは、1つの出力サンプルを計算するためにN CoeffとN個の入力サンプルが必要です。
スループットを最適化するには、フィルタが現在のセットで作業している間、次のサンプルを読み込むために、入力バッファサイズをNより大きくする必要があります。
たとえば、4ビートDMA転送を使用する場合、ε は4に設定する必要があります。
また、出力バッファのサイズを ε に設定して、結果のサンプルを一意のAHBバーストトランザクションで転送する必要があります。

汎用デジタル・フィルタ

無限インパルス応答(IIR)フィルタ、ダイレクトフォーム1ストラクチャ



- 無限インパルス応答(IIR)フィルタは、FIR出力y[n]と2番目の係数セットa[j]の間のFIRと2番目の畳み込みの組合せ

$$y[n] = 2^R \left(\sum_{k=0}^{N} b[k]x[n-k] + \sum_{j=1}^{N} a[j]y[n-j] \right)$$

バッファ	利用
X1	サンプリングされたデータ
X2	連結されたCoeffベクトルBとA(b0、b1、...、bN、a1、 ...、aM)
Y	出力値

パラメータ	説明
P	ベクトルBの長さ
Q	ベクトルAの長さ
R	出力に適用されるゲイン

IIRフィルタ出力ベクトルYは、長さN+1のCoeffベクトルBと無期限長のベクトルXの畳み込みに加えて、2番目のCoeffベクトルAの遅延出力ベクトルY‘の畳み込みに加えて、長さMの結合です。
FMACでIIRを実装するには、バッファを次のように使用します。
·X1バッファにはベクトルXの要素が含まれます。長さN+1+dのサーキュラバッファである。
·X2バッファには、CoeffベクトルBとAを連結した要素が含まれます(b0、b1、b2...、bN、a1、a2、...、aM)。長さM+N+1の固定バッファである。
·Yバッファには出力値ynが含まれます。長さM+dのサーキュラバッファである。

パラメータは次のとおりです。
·パラメータPには、CoeffベクトルBの長さN+1の範囲[2:64]が含まれます。
·パラメータQには、CoeffベクトルAの長さMが範囲[1:63]で含まれます。
·パラメータRには、アキュムレータ出力に適用されるゲインが含まれます。Yバッファに出力される値は2^Rで乗算され、Rは[0:7]の範囲になります。
この機能は、FMAC_PARAMレジスタ内のSTARTビットがソフトウェアによってリセットされると完了します。

無限インパルス応答（IIR）フィルタ、ダイレクトフォーム1ストラクチャ

- メモリ・リクアイアメント
  - NフィードフォワードCoeffとMフィードバックCoeff（M ~ N）
  - N個の入力サンプルとM出力サンプルで1つの出力サンプルを計算
  - DMA転送サイズε
  - 合計メモリ・リクアイアメント:2(N + M + ε) < 256

FIRにはNフィードフォワードCoeffとMフィードバックCoeffが必要で、MはNより小さくなります。
入力バッファサイズはN + ε 、ε はDMAバースト内のデータ数である必要があります。
出力バッファサイズはM+ ε である必要があります。

スループット

- Nタップフィルタには、出力サンプルごとにN乗算と加算が必要
  - 1サイクルあたり1回の乗算+加算を想定すると、最大スループット=サンプルあたりのNサイクル
  - 実際には、1つの乗算は、メモリからの2つのフェッチを必要とする（Coeff+サンプル）
    - メモリは単一ポートであるため、MACオペレーションごとに2つのサイクルが必要
- したがって:
  - 最大サンプル･レート、Fs < Fclk÷(2N)
  - 最大フィルタ･サイズ、N < Fclk÷(2Fs)

life.augmented

FMACモジュールのクロックリファレンスは、Cortex®-M4のFPUフリーランニングクロックFCLKです。
FIRなどのNタップフィルタでは、出力サンプルごとにNの乗算と加算が必要で、それぞれMACが2クロックサイクルを必要とし、2つのメモリ値を読み取ります。
その結果、最大サンプルレートはFclk/(2*N)になります。
NはFclk周波数よりも低く、最大サンプルレート周波数の2倍で割った値にする必要があります。

STM32G4のFclk周波数が170MHzであると仮定すると、以下が得られます。

- Fs = 2Mspsでの最大フィルタサイズはN < 42タップです。
- N = 128タップの最大サンプルレートはFsの664kHzです。

FMAC対ソフトウェア

- ARM Fast mathライブラリと比較
  - STM32G474は150MHz動作
  - IAR EWARM 7.82、最適化はハイスピード

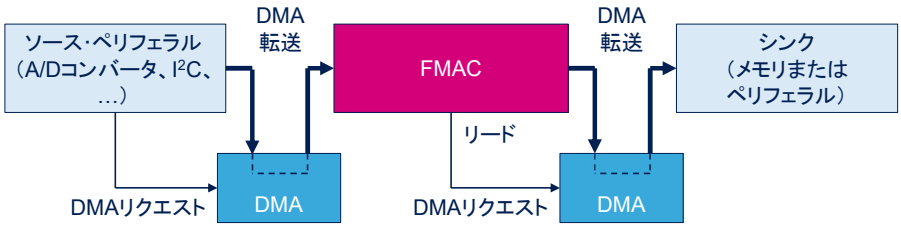
フィルタ	CMSIS DSP（ソフトウェア）関数	CMSIS DSP 実行時間（クロックサイクル/サンプル）	FMAC 実行時間（クロックサイクル/サンプル）
単一ステージバイクワッド/2p2z（IIR with N=3, M=2）	arm_biquad_cascade_df1_fast_q15()	12	14
51タップFIR（N=51）	arm_fir_fast_q15()	92	104

このスライドの表は、FMACハードウェアブロックのパフォーマンスとフィルタソフトウェアの実装を比較しています。
CMSIS DSPライブラリは、最適化されたフィルタ関数が提供されます。
デュアルMAC Cortex-M4により、ソフトウェアは約15%高速です。
しかし、高いサンプルレート/大きなフィルタの場合、CPUはフィルタリングタスクに多くの時間を費やしてしまいます。
FMAC + DMAを使用すると、CPUは他のタスクを自由に実行できます。

フロー・コントロール

ソースドリブン・フロー・コントロール

- サンプルのソース（A/Dコンバータ、I²C、…）は、サンプル・データレートを定義

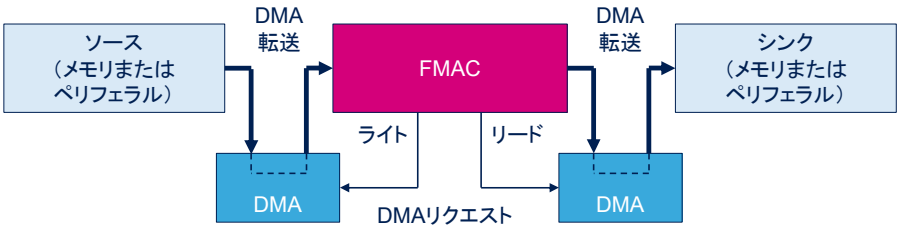


フローコントロールは、ソース、シンク、またはフィルタドリブンにすることができます。
このスライドでは、ソース・ドリブン・フロー・コントロールシーケンスについて説明します。
サンプルのソース（A/Dコンバータ、I2C、…）は、サンプルデータレートを定義します。
ソースは、フィルタ入力バッファにデータを転送するDMA（またはCPU）にリクエストします。
フィルタは、ソースサンプルレートの2N倍よりも速いクロックレートで動作します。
入力バッファが空の場合（次のサンプルは使用できない）、フィルタは停止し、新しいデータが来るまで待機します。
出力バッファが空でない場合（1つ以上のサンプルが使用可能）、出力チャネルDMAリクエスト（または割込み）が生成されます。
DMA（またはCPU）は、出力サンプルをメモリまたはD/Aコンバータ やPWMなどの別のペリフェラルに転送します。

フロー・コントロール

フィルタ・ドリブン・フロー・コントロール

- フィルタ・クロック・レートによってスループットが決定

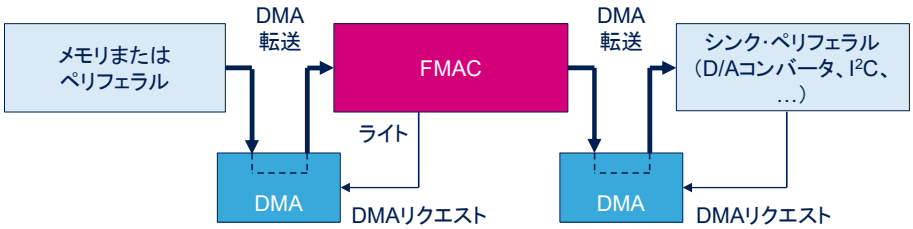


このスライドでは、フィルタ・ドリブン・フロー・コントロールシーケンスについて説明します。
フィルタクロックレートによってスループットが決まります。
入力バッファがいっぱいでないと、入力チャネルDMAリクエスト（または割込み）が生成されます。
DMA（またはCPU）は、メモリまたは別のペリフェラルから入力バッファにデータを転送します。
入力バッファでデータが使用可能である限り、フィルタは新しい出力サンプルを生成します。
出力バッファが空でない場合は、出力チャネルDMAリクエスト（または割込み）が生成されます。
DMA（またはCPU）は、出力バッファからメモリまたは別のペリフェラルにデータを転送します。

フロー・コントロール

シンク・ドリブン・フロー・コントロール

- サンプルの送信先（D/Aコンバータ、I²C、…）が、サンプル・データレートを定義



このスライドでは、シンク・ドリブン・フロー・コントロールシーケンスについて説明します。
サンプルの送信先（D/Aコンバータ、I2C、…）は、サンプルデータレートを定義します。
送信先は、フィルタ出力からデータを転送するようにDMA（またはCPU）を要求します。
フィルタは、送信先のサンプルレートの2N倍よりも速いクロックレートで動作します。
出力バッファがいっぱいになると、フィルタは停止します。
入力バッファがいっぱいでない場合は、入力チャネルDMAリクエスト（または割込み）が生成されます。
DMA（またはCPU）は、メモリまたは別のペリフェラルから入力バッファにサンプルを転送します。

フロー・コントロール

サマリー

- 入力バッファが空ではなく、出力バッファがいっぱいでない場合、フィルタが動作
  - 入力バッファのフル条件と出力バッファの空の条件は、それぞれの入力フラグと出力フラグを設定
  - 入力バッファのフルフラグが非アクティブの場合に入力チャネルDMAリクエストを生成するようにフィルタを構成
  - 出力バッファの空フラグが非アクティブの場合に出力チャネルDMAリクエストを生成するようにフィルタを構成出来る
- フィルタは、いずれかのフラグが非アクティブの場合に割込みリクエストを生成することも可能
- フィルタクロック周波数は、選択したフローコントロールスキームに従って選択する必要がある

life.augmented

入力バッファが空ではなく、出力バッファがいっぱいでない場合、FMACはフィルタ・アルゴリズムを実行します。
使用可能なスペースの数がX1バッファの場合、X1FULLというフラグがFULL_WMしきい値より少ない場合は、X1FULLというフラグが設定されます。
このフラグがX1バッファを埋めるために設定されていない場合は、DMAリクエストを生成できます。
未読データの数がEMPTY_WMしきい値より少ない場合は、YEMPTYというフラグが設定されます。
DMAリクエストは、このフラグがYバッファを空にするために設定されていない場合に生成できます。
バッファの管理は、いずれかのフラグが非アクティブなときにアサートできる割込みリクエストに依存するソフトウェアによっても実行できます。
フィルタクロック周波数は、選択したフローコントロールスキームに従って選択する必要があります。

低電力モード

モード	説明
RUN	有効
SLEEP	有効 • ペリフェラルの割込みにより、デバイスがSLEEPモードを終了
低電力RUN	有効
低電力SLEEP	有効 • ペリフェラルの割込みにより、デバイスが低電力SLEEPモードを終了
STOP0/STOP1	使用不可
STOP2	
STANDBY	
SHUTDOWN	

life.augmented

FMACユニットは、RUN、低電力RUN、SLEEP、低電力SLEEPモードで有効です。
他の低電力モードでは使用できません。

関連ペリフェラル

- 必要に応じて、このペリフェラルにリンクされているこれらのトレーニングを参照してください。
  - DMA - ダイレクトメモリアクセスコントローラ
  - 割込み - ネスト化されたベクタ割込みコントローラ

life.augmented

これらのペリフェラルは、FMACブロックと正しく使用するために特別に設定する必要がある場合があります。
より詳細な情報は、対応するペリフェラルのトレーニング資料を参照してください。