RH850/U2Bx

全結合型ニューラルネットワーク

要旨

本アプリケーションノートは、RH850/U2Bx に搭載されている浮動小数点演算ユニット（FPU）、及び拡張浮動小数点演算ユニット（FXU）を使用した全結合型ニューラルネットワークの演算方法について説明します。

本アプリケーションノートには、FXU の仕様詳細情報は含まれていません。詳細は別紙 APN「FXU Use for FP-SIMD Calculations」をご参照ください。また、製品毎に FXU 搭載の有無、及び搭載される CPU の位置が異なるため、ご使用になる場合にはあらかじめ製品仕様をご確認ください。詳細は appendix をご参照ください。

なお、本アプリケーションノートに記載している全結合型ニューラルネットワーク例は動作確認しておりますが、実際にご使用になる場合には、必ず動作確認の上ご使用くださいますようお願いします。

動作確認デバイス

RH850/U2Bx

目次

1. 全結合型ニューラルネットワーク概要

1.1 全結合型ニューラルネットワークの構成

本アプリケーションの動作例における全結合型ニューラルネットワークの構成図を図 1-1 に示します。入力層、中間層×2、出力層で構成されます。

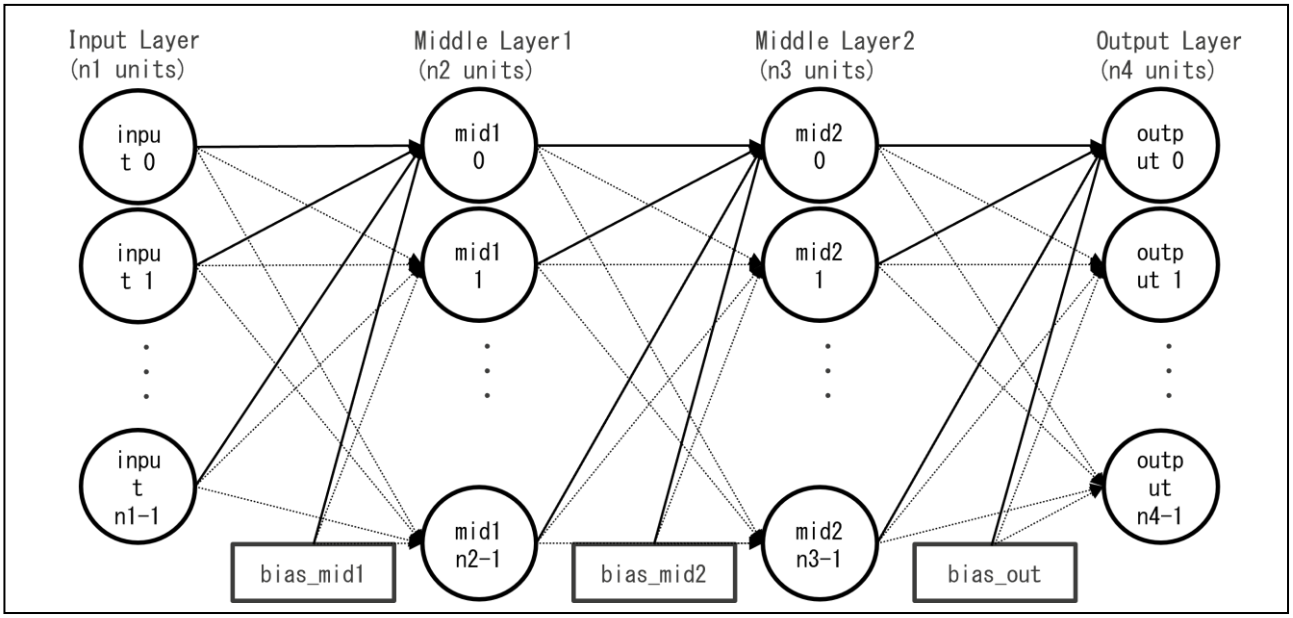


図 1-1 全結合型ニューラルネットワーク構成図

1.2 演算方法

各層で、全結合処理及び活性化関数処理を行います。入力層から中間層 1 への計算式を図 1-2 に示します。

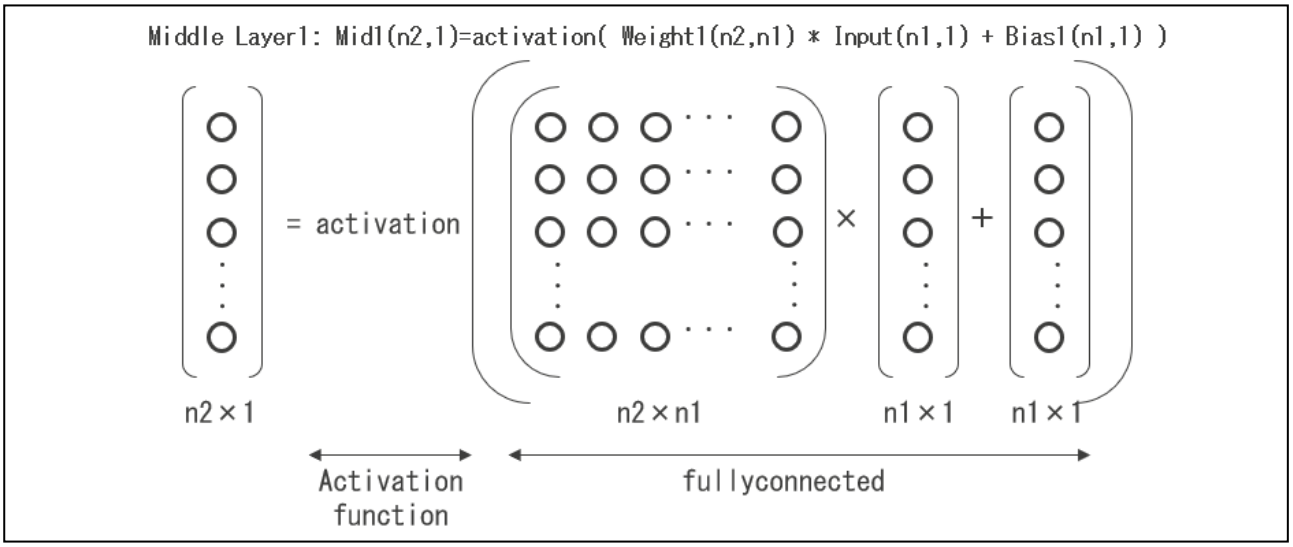


図 1-2 各層の計算式

1.3 サポート関数

本サンプルソフトでは、以下の関数をサポートしています。

表 1-1 サポート関数一覧

機能		FPU	FXU
全結合		fullyconnected	fullyconnected_fxu
活性化関数	tanh	act_tanh	act_tanh_fxu
		act_tanh_usingexp	act_tanh_usingexp_fxu
	sigmoid	act_sigmoid	act_sigmoid_fxu
	ReLU	act_relu	act_relu_fxu

1.4 使用するハードウェア機能

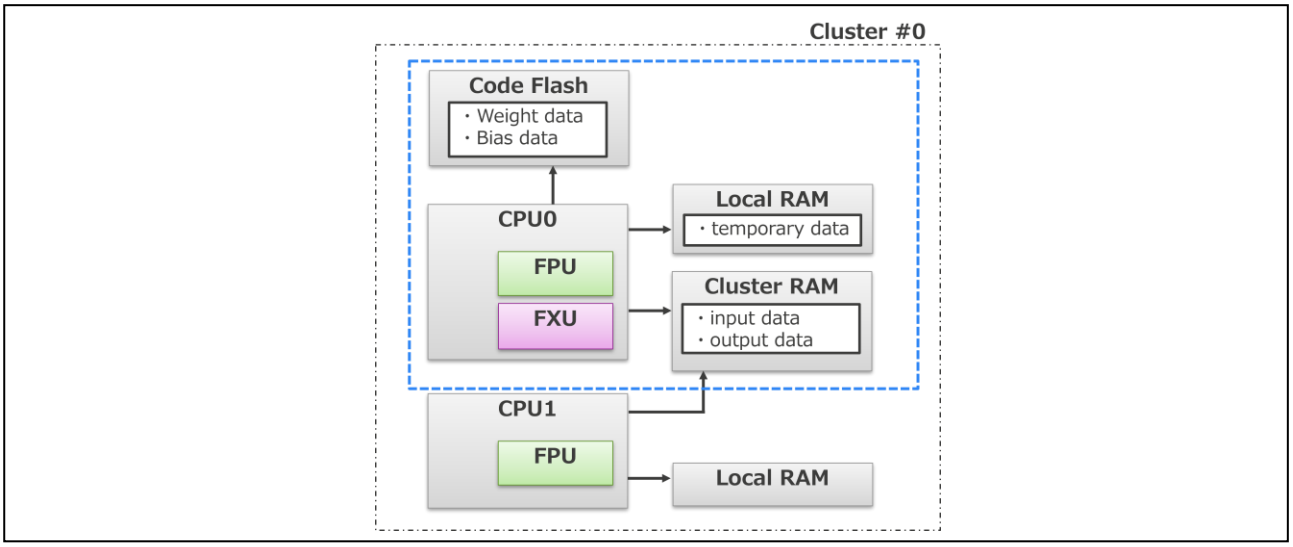
本サンプルソフトで使用する RH850/U2Bx のハードウェア機能を、以下に示します。

- 浮動小数点演算ユニット（FPU）
- 拡張浮動小数点演算ユニット（FXU）
- 各種メモリ（Code Flash、Cluster RAM、Local RAM）

本サンプルソフトは、CPU0 を用いて 1 つのクラスタ内（Cluster #0）で処理を行います。定数、変数データ配置の詳細は「2.4 定数及び変数の配置」を参照ください。

なお、本サンプルソフトでは単精度（32 ビット）で演算を行います。

図 1-3 システム構成



2. ソフトウェア説明

2.1 動作フロー

本サンプルソフトにおける動作フローを図 2-1 に示します。以下の動作フローは活性化関数に tanh 関数を使用した例です。シグモイド関数や ReLU 関数もサポートしていますので、必要に応じて置き換えてご使用ください。

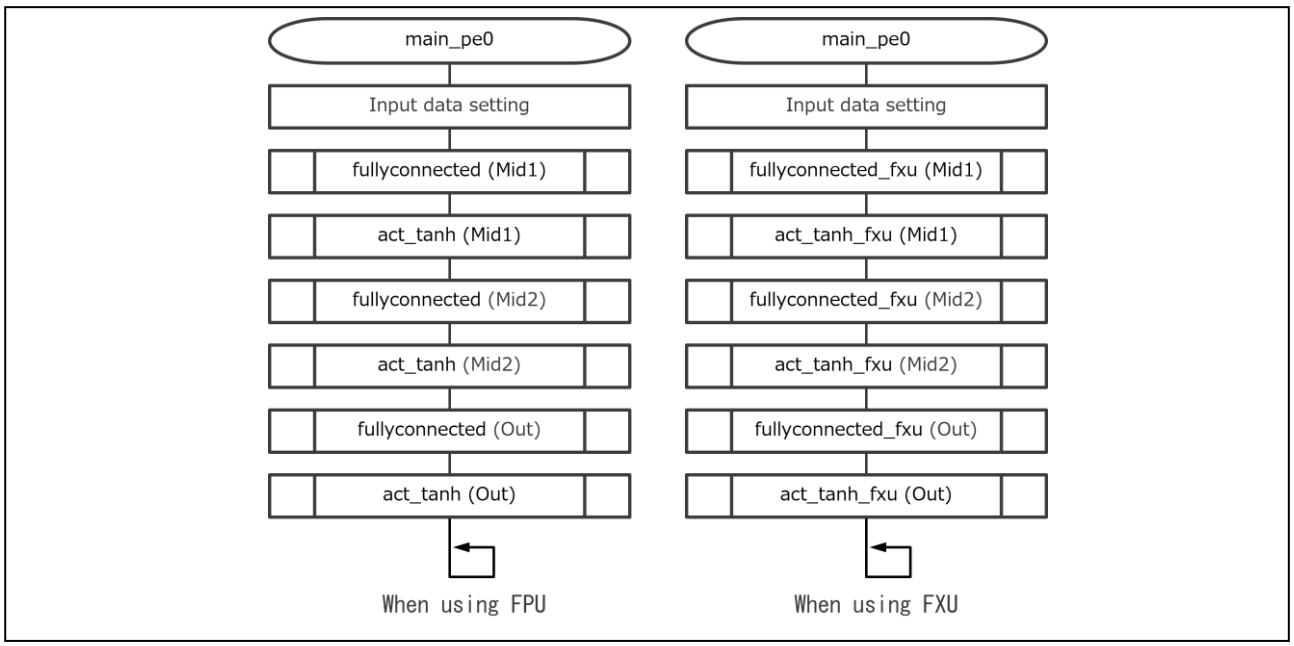


図 2-1 動作フロー

2.2 サンプルソフト構成

サンプルソフトのファイル構成を表 2-1 に示します。

表 2-1 サンプルソフトファイル構成

ファイル名					概要
FNN	FPU	U2Bx_Sample.gpj			親プロジェクトファイル
		U2B10_Sample.gpj			親プロジェクトファイル
		src	U2B10_Sample.gpj		プロジェクトファイル
			core0	fnn_fpu.c	全結合処理関数、活性化関数
				weight_data_fpu(a/b/c).c	各パターン定数データ 「2.5 ユニット数の変更」参照
				weight_data_fpu(a/b/c).h	各パターンのヘッダファイル 「2.5 ユニット数の変更」参照
				sub_timer_benchmark.c	処理負荷計測用ファイル
				sub_timer_benchmark.h	処理負荷計測用ヘッダファイル
				main_pe0.c	CPU0 用の main 関数
				intprg.c	割り込み処理関数 処理内容は特になし
			core1	main_pe0.c	CPU1 用の main 関数
				intprg.c	割り込み処理関数 処理内容は特になし
			core2	main_pe0.c	CPU2 用の main 関数
				intprg.c	割り込み処理関数 処理内容は特になし
			core3	main_pe0.c	CPU3 用の main 関数
				intprg.c	割り込み処理関数 処理内容は特になし
		startup			スタートアップルーチン
	FXU	U2Bx_Sample.gpj			親プロジェクトファイル
		U2B10_Sample.gpj			親プロジェクトファイル
		src	U2B10_Sample.gpj		プロジェクトファイル
			core0	fnn_fxu.c	全結合処理関数、活性化関数
				weight_data_fxu(a/b/c).c	各パターン定数データ

					「2.5 ユニット数の変更」参照
				weight_data_fxu(a/b/c).h	各パターンのヘッダファイル 「2.5 ユニット数の変更」参照
				sub_timer_benchmark.c	処理負荷計測用ファイル
				sub_timer_benchmark.h	処理負荷計測用ヘッダファイル
				main_pe0.c	CPU0 用の main 関数
				intprg.c	割り込み処理関数 処理内容は特になし
			core1	main_pe0.c	CPU1 用の main 関数
				intprg.c	割り込み処理関数 処理内容は特になし
			core2	main_pe0.c	CPU2 用の main 関数
				intprg.c	割り込み処理関数 処理内容は特になし
			core3	main_pe0.c	CPU3 用の main 関数
				intprg.c	割り込み処理関数 処理内容は特になし
		startup			スタートアップルーチン

2.3 関数仕様

2.3.1 FPU 版関数

表 2-2 に本動作例で使用する FPU 版関数一覧を示します。

表 2-2 FPU 版関数一覧

関数名	概要
main_pe0	各関数の呼び出しを行います。
fullyconnected	全結合処理を行います。
act_tanh	活性化関数処理（tanh）を行います。
act_tanh_usingexp	活性化関数処理（tanh）を行います。 以下の変換式に従って、指数関数と四則演算を用いて tanh 関数処理を実行します。 $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
act_sigmoid	活性化関数処理（シグモイド）を行います。
act_relu	活性化関数処理（ReLU）を行います。

表 2-3 から表 2-7 に本動作例で使用する FPU 版関数の仕様を示します。

表 2-3 fullyconnected 関数の仕様

fullyconnected			
概　要	全結合処理を行い、結果を指定された配列に格納します。		
宣　言	`void fullyconnected(float input[], const float weight[], const float bias[], float output[], unsigned int size_in, unsigned int size_out);`		
引　数	[IN]	float input[]	：全結合処理の入力データを指定します。
	[IN]	float weight[]	：全結合処理の重み行列データを指定します。
	[IN]	float bias[]	：全結合処理のバイアスデータを指定します。
	[OUT]	float output[]	：全結合処理の結果を格納します。
	[IN]	unsigned int size_in	：入力データのサイズを指定します。
	[IN]	unsigned int size_out	：出力データのサイズを指定します。
リターン値	-		
備考	- 引数に指定する重み行列データは、転置した状態で配置してください。（「3.3 Code Flash への定数データ配置」参照）		

表 2-4 act_tanh 関数の仕様

act_tanh			
概　要	活性化関数処理（tanh）を行い、結果を指定された配列に格納します。		
宣　言	`void act_tanh(float input[], float output[], unsigned int size_in);`		
引　数	[IN]	float input[]	：活性化関数処理（tanh）の入力データを指定します。
	[OUT]	float output[]	：活性化関数処理（tanh）の結果を格納します。
	[IN]	unsigned int size_in	：入力データのサイズを指定します。
リターン値	-		
備考			

表 2-5 act_tanh_usingexp 関数の仕様

act_tanh_usingexp			
概　要	活性化関数処理（tanh）を行い、結果を指定された配列に格納します。 指数関数と四則演算を用いて tanh 関数処理を実行します。		
宣　言	`void act_tanh_usingexp(float input[], float output[], unsigned int size_in);`		
引　数	[IN]	float input[]	：活性化関数処理（tanh）の入力データを指定します。
	[OUT]	float output[]	：活性化関数処理（tanh）の結果を格納します。
	[IN]	unsigned int size_in	：入力データのサイズを指定します。
リターン値	-		
備考	- tanh 関数を以下に示すように、指数関数を用いた数式に変換して演算を行います。 $\tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$ - 本関数内では expf 関数を使用しており、入力 n に対して e^{n} となるため入力範囲には注意が必要となります。		

表 2-6 act_sigmoid 仕様

act_sigmoid			
概　要	活性化関数処理（シグモイド）を行い、結果を指定された配列に格納します。		
宣　言	`void act_sigmoid(float input[], float output[], unsigned int size_in);`		
引　数	[IN]	float input[]	：活性化関数処理（シグモイド）の入力データを指定します。
	[OUT]	float output[]	：活性化関数処理（シグモイド）の結果を格納します。
	[IN]	unsigned int size_in	：入力データのサイズを指定します。
リターン値	-		
備考	- 本関数内では expf 関数を使用しており、入力 n に対して e^n となるため入力範囲には注意が必要となります。		

表 2-7 act_relu 仕様

act_relu			
概　要	活性化関数処理（ReLU）を行い、結果を指定された配列に格納します。		
宣　言	`void act_relu(float input[], float output[], unsigned int size_in);`		
引　数	[IN]	float input[]	：活性化関数処理（ReLU）の入力データを指定します。
	[OUT]	float output[]	：活性化関数処理（ReLU）の結果を格納します。
	[IN]	unsigned int size_in	：入力データのサイズを指定します。
リターン値	-		
備考			

2.3.2 FXU 版関数

表 2-8 に本動作例で使用する FXU 版関数一覧を示します。

本サンプルソフトでは、GHS で標準サポートされている FXU 命令の組み込み関数を使用しています。組み込み関数の詳細は「3.4.1.2 FXU 組み込み関数の詳細」を参照してください。

表 2-8 FXU 版関数一覧

関数名	概要
main_pe0	各関数の呼び出しを行います。
fullyconnected_fxu	FXU を使用した全結合処理を行います。
act_tanh_fxu	FXU を使用した活性化関数処理（tanh）を行います。
act_tanh_usingexp_fxu	FXU を使用した活性化関数処理（tanh）を行います。 以下の変換式に従って、指数関数と四則演算を用いて tanh 関数処理を実行します。 $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
act_sigmoid_fxu	FXU を使用した活性化関数処理（シグモイド）を行います。
act_relu_fxu	FXU を使用した活性化関数処理（ReLU）を行います。
tanhf_vector	ベクトル要素それぞれに tanhf 関数処理を行います。
expf_vector	ベクトル要素それぞれに expf 関数処理を行います。

表 2-9 から表 2-15 に本動作例で使用する FXU 版関数の仕様を示します。

表 2-9 fullyconnected_fxu 関数の仕様

fullyconnected_fxu			
概　要	FXU を使用した全結合処理を行い、結果を指定された配列に格納します。		
宣　言	`void fullyconnected_fxu(float input[], const float weight[], const float bias[], float output[], unsigned int size_in, unsigned int size_out);`		
引　数	[IN]	float input[]	：全結合処理の入力データを指定します。
	[IN]	float weight[]	：全結合処理の重み行列データを指定します。
	[IN]	float bias[]	：全結合処理のバイアスデータを指定します。
	[OUT]	float output[]	：全結合処理結果を格納します。
	[IN]	uinsigned int size_in	：入力データのサイズを指定します。
	[IN]	unsigned int size_out	：出力データのサイズを指定します。
リターン値	-		
備考	- 引数に指定する重み行列データは、転置した状態で配置してください。（「3.3 Code Flash への定数データ配置」参照） - 引数に指定する重み行列データの列サイズ、及びバイアスデータのサイズが 4 の倍数でない場合、サイズが 4 の倍数になるまで 0 埋めしてください。同時に引数 size_out も 4 の倍数にしてください。（「3.4.2 データサイズ」参照） - 引数 input[]、weight[]、bias[]、output[]に指定するデータの先頭番地は、16 バイト境界に配置してください。（「3.4.3 アライメント指定」参照）		

表 2-10 act_tanh_fxu 関数の仕様

act_tanh_fxu			
概　要	FXU を使用した活性化関数処理（tanh）を行い、結果を指定された配列に格納します。		
宣　言	`void act_tanh_fxu(float input[], float output[], unsigned int size_in);`		
引　数	[IN]	float input[]	：活性化関数処理（tanh）の入力データを指定します。
	[OUT]	float output[]	：活性化関数処理（tanh）の結果を格納します。
	[IN]	unsigned int size_in	：入力データのサイズを指定します。
リターン値	-		
備考	- 引数 input[]、output[]に指定するデータの先頭番地は、16 バイト境界に配置してください。（「3.4.3 アライメント指定」参照）		

表 2-11 act_tanh_usingexp_fxu 関数の仕様

act_tanh_usingexp_fxu			
概　要	FXU を使用した活性化関数処理（tanh）を行い、結果を指定された配列に格納します。 指数関数と四則演算を用いて tanh 関数処理を実行します。		
宣　言	`void act_tanh_usingexp_fxu(float input[], float output[], unsigned int size_in);`		
引　数	[IN]	float input[]	：活性化関数処理（tanh）の入力データを指定します。
	[OUT]	float output[]	：活性化関数処理（tanh）の結果を格納します。
	[IN]	unsigned int size_in	：入力データのサイズを指定します。
リターン値	-		
備考	- tanh 関数を以下に示すように、指数関数を用いた数式に変換して演算を行うことで、処理の高速化が見込まれます。 $\tanh x = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$ - 本関数内では expf 関数を使用しており、入力 n に対して e^{n} となるため入力範囲には注意が必要となります。 - 引数 input[]、output[]に指定するデータの先頭番地は、16 バイト境界に配置してください。 （「3.4.3 アライメント指定」参照）		

表 2-12 act_sigmoid_fxu 関数の仕様

act_sigmoid_fxu			
概　要	FXU を使用した活性化関数処理（シグモイド）を行い、結果を指定された配列に格納します。		
宣　言	`void act_sigmoid_fxu(float input[], float output[], unsigned int size_in);`		
引　数	[IN]	float input[]	：活性化関数処理（シグモイド）の入力データを指定します。
	[OUT]	float output[]	：活性化関数処理（シグモイド）の結果を格納します。
	[IN]	unsigned int size_in	：入力データのサイズを指定します。
リターン値	-		
備考	- 本関数内では expf 関数を使用しており、入力 n に対して e^{n} となるため入力範囲には注意が必要となります。 - 引数 input[]、output[]に指定するデータの先頭番地は、16 バイト境界に配置してください。 （「3.4.3 アライメント指定」参照）		

表 2-13 act_relu_fxu 関数の仕様

act_relu_fxu			
概　要	FXU を使用した活性化関数処理（ReLU）を行い、結果を指定された配列に格納します。		
宣　言	`void act_relu_fxu(float input[], float output[], unsigned int size_in);`		
引　数	[IN]	float input[]	：活性化関数処理（ReLU）の入力データを指定します。
	[OUT]	float output[]	：活性化関数処理（ReLU）の結果を格納します。
	[IN]	unsigned int size_in	：入力データのサイズを指定します。
リターン値	-		
備考	- 引数 input[]、output[]に指定するデータの先頭番地は、16 バイト境界に配置してください。（「3.4.3 アライメント指定」参照）		

表 2-14 tanhf_vector 関数の仕様

tanhf_vector			
概　要	浮動小数点の tanh 関数計算を行います。引数に指定したベクトルの 4 要素それぞれに対して実行され、結果をベクトルに格納します。		
宣　言	`__ev128_f32__ tanhf_vector(__ev128_f32__ x);`		
引　数	[IN]	__ev128_f32__ x	：tanh 関数計算を行うベクトルを指定します。
リターン値	__ev128_f32__型の値		：tanh 関数計算結果を格納したベクトルを返します。
備考			

表 2-15 expf_vector 関数の仕様

expf_vector			
概　要	浮動小数点の指数関数計算を行います。引数に指定したベクトルの 4 要素それぞれに対して実行され、結果をベクトルに格納します。		
宣　言	`__ev128_f32__ expf_vector(__ev128_f32__ x);`		
引　数	[IN]	__ev128_f32__ x	：指数関数計算を行うベクトルを指定します。
リターン値	__ev128_f32__型の値		：指数関数計算結果を格納したベクトルを返します。
備考			

2.4 定数及び変数の配置

本サンプルソフトでは、CPU0 を用いて 1 つのクラスタ内（Cluster #0）で処理を行います。図 2-2 及び表 2-16 に示すように、入出力データは Cluster RAM、定数（重み行列データ、バイアスデータ）は Code Flash に配置します。

使用する CPU に対応するリソースにデータが適切に配置されない場合、データアクセスが遅くなり処理性能が低下する恐れがあるためご注意ください。

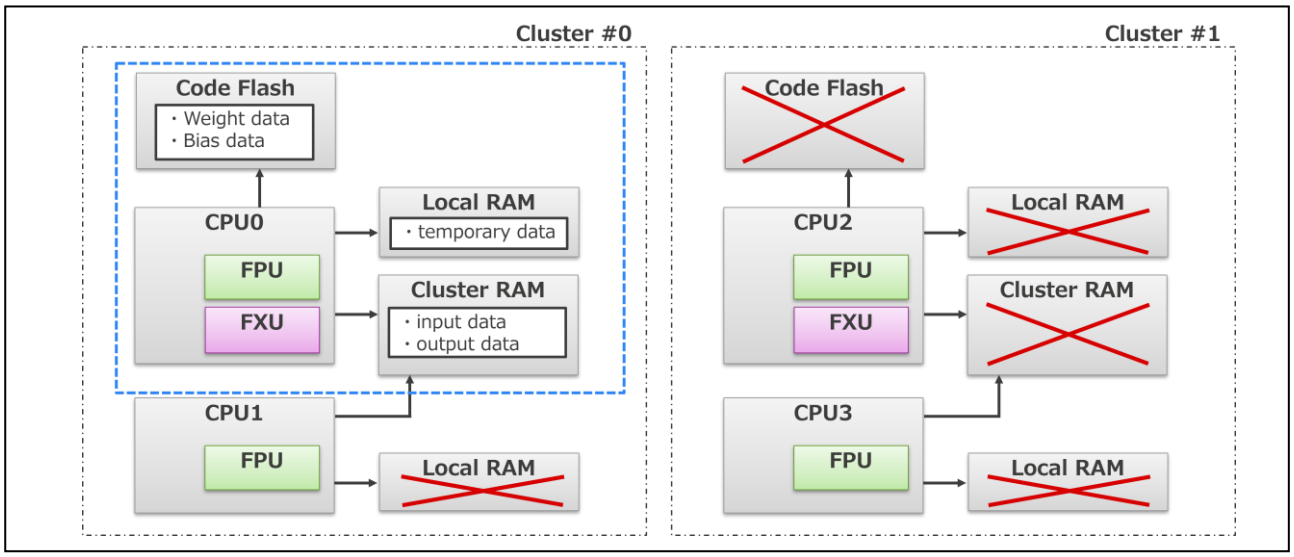


図 2-2 定数及び変数の配置

表 2-16 定数及び変数の種類と配置

種類		データ名	定数/変数	配置
入出力データ		input_data	グローバル変数	クラスタ内 Cluster RAM
		output_data		
重みデータ	中間層 1	weight_mid1	定数	クラスタ内 Code Flash
	中間層 2	weight_mid2		
	出力層	weight_out		
バイアスデータ	中間層 1	bias_mid1		
	中間層 2	bias_mid2		
	出力層	bias_out		
中間データ	中間層 1	mid1_tmp	ローカル変数	Local RAM
		mid1_out		
	中間層 2	mid2_tmp		
		mid2_out		
	出力層	out_tmp		

2.5 ユニット数の変更

本サンプルソフトは、FPU/FXU 共に 3 パターンのデータセットを選択可能です。

表 2-17 ユニット数のパターン

パターン		ユニット数				ファイル	
		入力層	中間層 1	中間層 2	出力層		
FPU	A	10	10	10	10	weight_data_fpua.h	weight_data_fpua.c
	B	10	20	20	10	weight_data_fpub.h	weight_data_fpub.c
	C	10	30	30	10	weight_data_fpuc.h	weight_data_fpuc.c
FXU*1	A	10	10	10	10	weight_data_fxua.h	weight_data_fxua.c
	B	10	20	20	10	weight_data_fxub.h	weight_data_fxub.c
	C	10	30	30	10	weight_data_fxuc.h	weight_data_fxuc.c

【注 1】 FXU は 4 要素ずつ処理を行うため、各層における重み行列データの列サイズ及びバイアスデータサイズは 4 の倍数にする必要があります。その場合、データ要素は 0 埋めを行い、データサイズを示すマクロ定数も変更してください。詳細は「3.4.2 データサイズ」を参照してください。

パターンを変更する場合は、パターンに対応したヘッダファイル及び定数データファイルを指定してください。

(1) main_pe0.c 内にあるマクロ名の定義を変更ください。それにより、読み込むヘッダファイルが変更されます。

例) パターン A に設定する場合は#define PATTERN_A

(2) 定数データファイルは、U2B10_Sample_src.gpj 内で指定してください。

例) パターン A（FPU）に設定する場合は.¥weight_data_fpua.c

3. 注意点及び制約事項

3.1 FPU/FXU の初期設定

FPU/FXU を使用する際は、PSW レジスタを設定する必要があります。また、FXU を使用する際は、オプションバイトの設定も行う必要があります。詳細な設定方法は各製品ユーザーズマニュアルをご参照ください。

また、製品毎に FXU 搭載の有無、及び搭載される CPU の位置が異なるため、ご使用になる場合はあらかじめ製品仕様をご確認ください。。詳細は appendix をご参照ください。

(a) PSW レジスタ

FPU：CPU のプログラムステータスワード（PSW）のビット 16（CU0）を”1”にすると有効になります。

FXU：CPU のプログラムステータスワード（PSW）のビット 17（CU1）を”1”にすると有効になります。

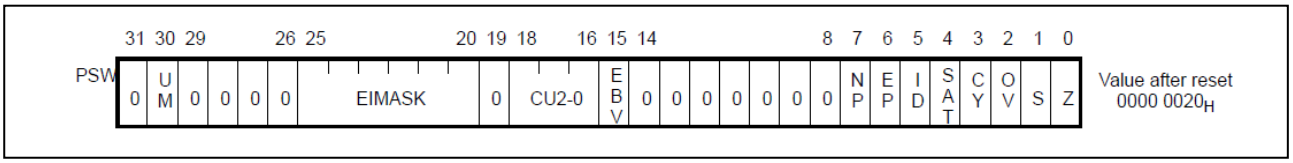


図 3-1 PSW レジスタ

(b) オプションバイト

CPU0 搭載 FXU：OPBT3 のビット 16（PE0_FPSIMD_EN）を”1”にすると有効になります。

CPU2 搭載 FXU：OPBT3 のビット 18（PE2_FPSIMD_EN）を”1”にすると有効になります。

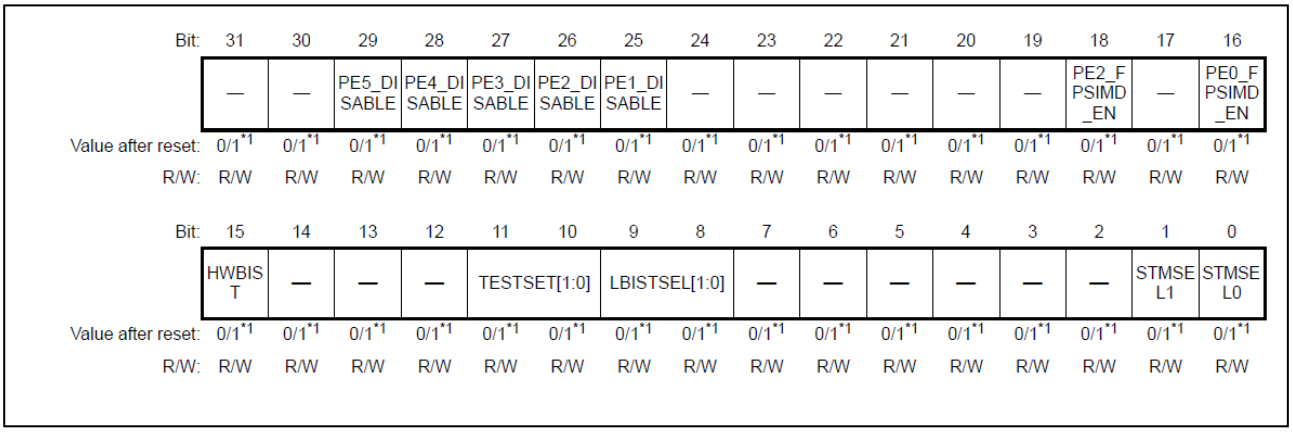


図 3-2 オプションバイト設定

3.2 単精度浮動小数点型の上限及び下限値

単精度浮動小数点型が表現可能な値の範囲には限りがあります。特に指数関数を使用する場合、入力 n に対して e^n となるため入力範囲には注意が必要となります。本サンプルソフトでは以下の関数で指数関数を使用しています。

act_tanh_usingexp、act_tanh_usingexp_fxu、act_sigmoid、act_sigmoid_fxu

3.3 Code Flash への定数データ配置

本項目では、本サンプルソフトにおける Code Flash からの定数データ読み出し方法と、Code Flash への定数データ配置の方法について説明します。

3.3.1 データバッファの有効利用

Code Flash のデータ読み出し時の最適化手法について説明します。Code Flash に配置したデータを読み出す際に、データがメモリ上に連続して配置されていなかった場合、データバッファのデータヒット性が悪く、読み出しに時間がかかり処理性能が低下します。そのため、本サンプルソフトでは図 3-3 に示すような構成でデータの読み出しを行います。これにより、データバッファを有効利用したデータの読み出しが可能となります。詳細は次項「3.3.2 重み行列データの配置」で説明します。

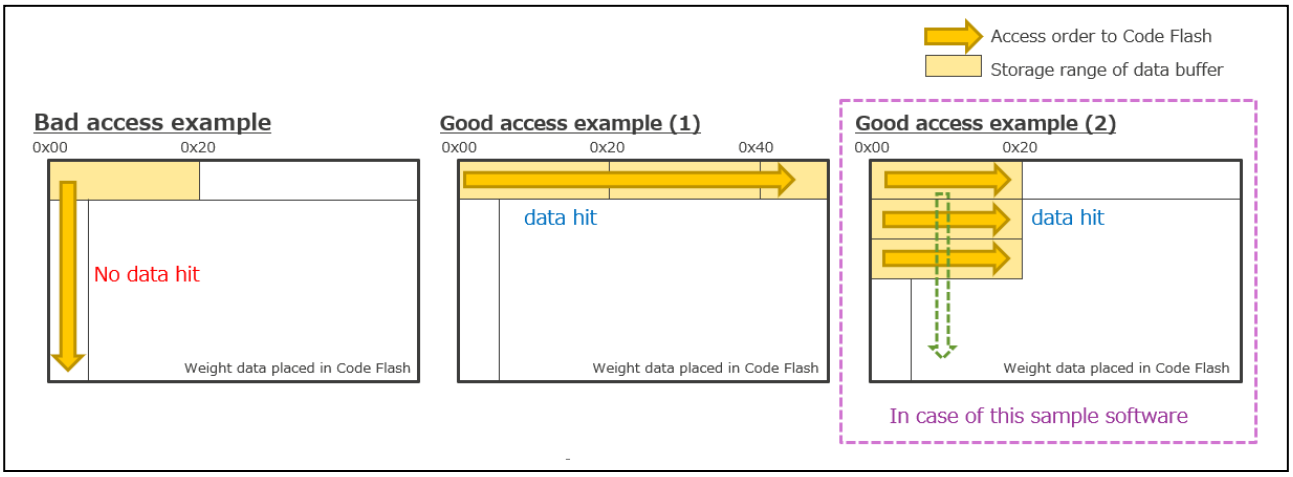


図 3-3 Code Flash へのアクセス順序最適化

3.3.2 重み行列データの転置

重み行列データは図 3-4 に示すように、行と列を入れ替えて転置した状態で配置してください。

FXU 命令は 4 要素ずつ処理を行います。そのため、行列とベクトルの積を一般的なデータ処理方向(行列の列方向)で演算する場合、出力値 1 要素を計算するためにベクトルレジスタ 4 要素の和をとる処理が必要になります。本サンプルソフトでは、図 3-4 に示すように行列データの行と列を入れ替えて、出力値複数要素分の積和を並列に行います。これにより、ベクトルレジスタ 4 要素の和をとる必要がなくなり、処理の高速化が見込めます。

また本サンプルソフトでは、FPU を使用する場合も、同様の構成でデータ処理を行います。そのため、FPU/FXU どちらを使用する場合も重み行列データは転置して配置してください。

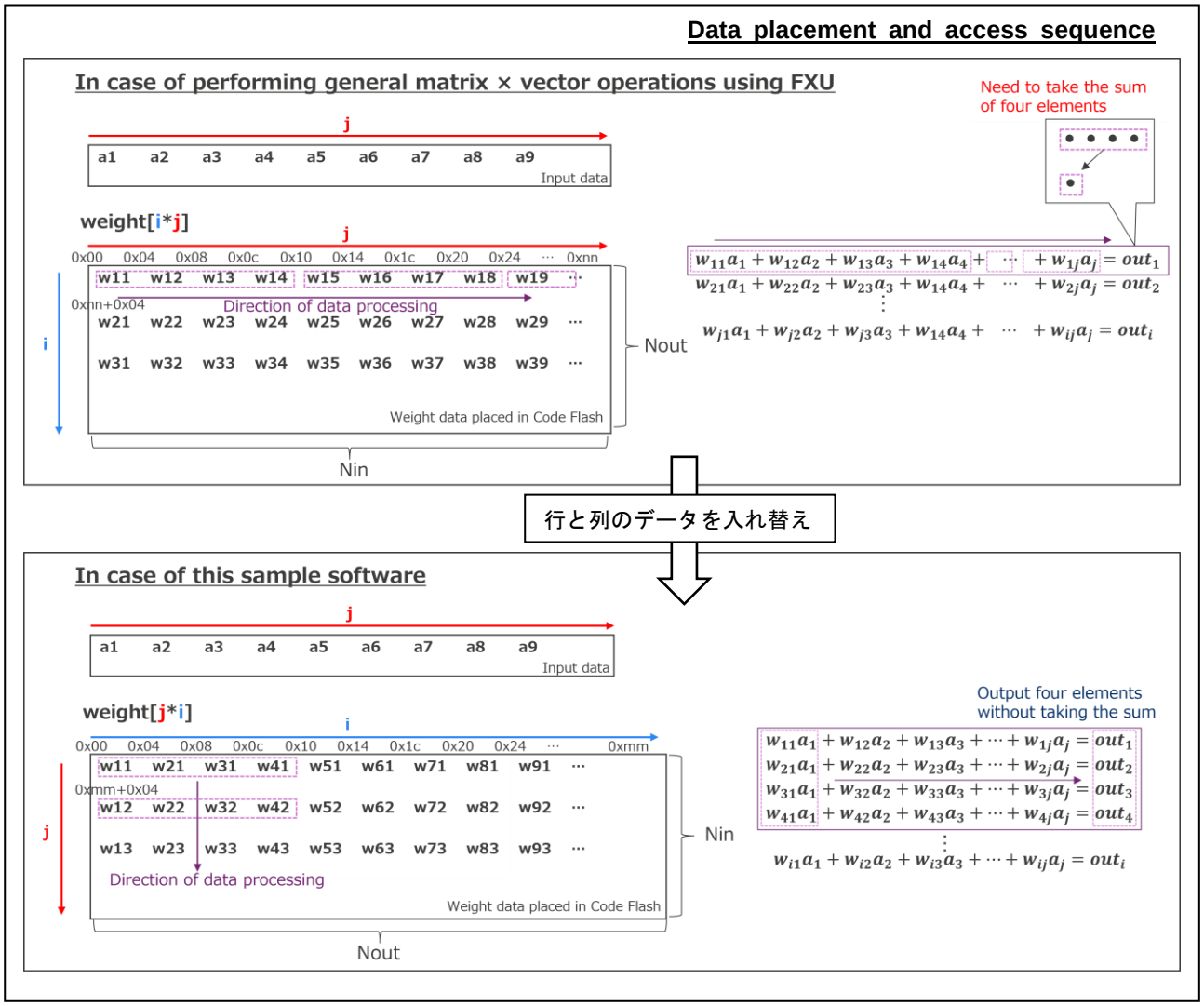


図 3-4 重み行列データの配置最適化

【注】 上記の図はイメージ例です。実際には、処理高速化のためループ展開を行っています。

3.4 FXU 使用時の注意点

3.4.1 FXU 組み込み関数

3.4.1.1 コンパイル時の設定

FXU 組み込み関数を使用するには、FXU サポートを有効にし、ヘッダファイル v800_fxu.h または、ヘッダファイル v800_ghs.h をインクルードする必要があります。FXU サポートが有効な時には、後者は前者を自動的にインクルードします。

- FXU サポートの有効化：-rh850_fxu
- ヘッダファイルのインクルード：#include <v800_fxu.h> or <v800_ghs.h>

3.4.1.2 FXU 組み込み関数の詳細

表 3-1 に本サンプルソフトで使用する FXU の組み込み関数一覧を示します。

表 3-1 FXU 組み込み関数一覧

組み込み関数名	概要
__ev128_ldvqw	ベクトルレジスタにクワッドワードをロードします。
__ev128_ldvw_mask	ベクトルレジスタの指定した要素にワードをロードします。
__ev128_stvqw	ベクトルレジスタのクワッドワードを指定アドレスに保存します。
__ev128_addfs_4	ベクトルレジスタの各要素に対して浮動小数点の加算を行います。
__ev128_subfs_4	ベクトルレジスタの各要素に対して浮動小数点の減算を行います。
__ev128_divfs_4	ベクトルレジスタの各要素に対して浮動小数点の除算を行います。
__ev128_fmafs_4	ベクトルレジスタの各要素に対して浮動小数点の融合積和を行います。
__ev128_get_f32	指定されたベクトルレジスタの要素を抽出します。

FXU 組み込み関数では、ベクトル・データ型である”__ev128_f32__”を使用します。これは、4 つの 32 ビット単精度浮動小数点要素のベクトルを表します。

表 3-2 から表 3-9 に本動作例で使用する FXU の組み込み関数の仕様を示します。

表 3-2 __ev128_ldvqw 関数の仕様

__ev128_ldvqw			
概　要	ベクトルレジスタにクワッドワードをロードします。この命令は、ptr で指定されたアドレスにあるクワッドワードを読み取り、値を結果ベクトルレジスタに格納します。		
宣　言	`__ev128_f32__ __ev128_ldvqw(void *ptr);`		
引　数	[IN]	void *ptr	：ベクトルレジスタにロードするクワッドワードの先頭アドレスを指定します。
リターン値	__ev128_f32__型の値		：クワッドワードを格納したベクトルを返します。
備考			

表 3-3 __ev128_ldvw_mask 関数の仕様

__ev128_ldvw_mask			

概　要	ベクトルレジスタの指定した要素にワードをロード/更新します。この命令は、ptr で指定されたアドレスのワードを読み取り、ベクトルを返します。その要素は、次のようにワードとベクトルレジスタの要素から mask の 4 ビットのイミディエイト値に従って結合されます。
	`val = *ptr`
	`res[w0] = ((mask & (1<<0)) == 1) ? val : x[w0]`
	`res[w1] = ((mask & (1<<1)) == 1) ? val : x[w1]`
	`res[w2] = ((mask & (1<<2)) == 1) ? val : x[w2]`
	`res[w3] = ((mask & (1<<3)) == 1) ? val : x[w3]`		

宣　言	`__ev128_f32__ __ev128_ldvw_mask(ghs_c_int__ mask, void *ptr, __ev128_f32__ x);`		
引　数	[IN]	__ghs_c_int__ mask	：更新するベクトルレジスタの要素を指定します。
	[IN]	void *ptr	：ベクトルレジスタにロードするワードのアドレスを指定します。
	[IN]	__ev128_f32__ x	：ロード/更新するベクトルレジスタを指定します。
リターン値	__ev128_f32__型の値		：クワッドワードを格納したベクトルを返します。
備考			

表 3-4 __ev128_stvqw 関数の仕様

__ev128_stvqw			
概　要	ベクトルレジスタのクワッドワードを ptr で指定されたアドレスに保存します。		
宣　言	`void __ev128_stvqw(__ev128_f32__ x, void *ptr);`		
引　数	[IN]	__ev128_f32__ x	：読み出すベクトルレジスタを指定します。
	[IN]	void *ptr	：読み出したクワッドワードを保存するアドレスを指定します。
リターン値			
備考			

表 3-5 __ev128_addfs_4 関数の仕様

__ev128_addfs_4			
概　要	浮動小数点の加算を行います。引数に指定したベクトルレジスタの 4 つの要素それぞれに対して実行され、結果をベクトルレジスタに格納します。		
宣　言	`__ev128_f32__ __ev128_addfs_4(__ev128_f32__ x, __ev128_f32__ y);`		
引　数	[IN]	__ev128_f32__ x	：加算を行うベクトルレジスタを指定します。
	[IN]	__ev128_f32__ y	：加算を行うベクトルレジスタを指定します。
リターン値	__ev128_f32__型の値		：加算結果を格納したベクトルを返します。
備考			

表 3-6 __ev128_subfs_4 関数の仕様

__ev128_subfs_4			
概　要	浮動小数点の減算を行います。引数に指定したベクトルレジスタの 4 つの要素それぞれに対して実行され、結果をベクトルレジスタに格納します。		
宣　言	`__ev128_f32__ __ev128_subfs_4(__ev128_f32__ x, __ev128_f32__ y);`		
引　数	[IN]	__ev128_f32__ x	：減数となるベクトルレジスタを指定します。
	[IN]	__ev128_f32__ y	：被減数となるベクトルレジスタを指定します。
リターン値	__ev128_f32__型の値		：減算結果を格納したベクトルを返します。
備考			

表 3-7 __ev128_divfs_4 関数の仕様

__ev128_divfs_4			
概　要	浮動小数点の除算を行います。引数に指定したベクトルレジスタの 4 つの要素それぞれに対して実行され、結果をベクトルレジスタに格納します。		
宣　言	`__ev128_f32__ __ev128_divfs_4(__ev128_f32__ x, __ev128_f32__ y);`		
引　数	[IN]	__ev128_f32__ x	：除数となるベクトルレジスタを指定します。
	[IN]	__ev128_f32__ y	：被除数となるベクトルレジスタを指定します。
リターン値	__ev128_f32__型の値		：除算結果を格納したベクトルを返します。
備考			

表 3-8 __ev128_fmafs_4 関数の仕様

__ev128_fmafs_4			
概　要	浮動小数点の融合積和を行います。引数に指定したベクトルレジスタの 4 つの要素それぞれに対して実行され、結果をベクトルレジスタに格納します。		
宣　言	`__ev128_f32__ __ev128_fmafs_4(__ev128_f32__ x, __ev128_f32__ y, __ev128_f32__ z);`		
引　数	[IN]	__ev128_f32__ x	：乗算を行うベクトルレジスタを指定します。
	[IN]	__ev128_f32__ y	：乗算を行うベクトルレジスタを指定します。
	[IN]	__ev128_f32__ z	：加算を行うベクトルレジスタを指定します。
リターン値	__ev128_f32__型の値		：融合積和結果を格納したベクトルレジスタを返します。
備考			

表 3-9 __ev128_get_f32 関数の仕様

__ev128_get_f32			
概　要	eid によって指定されたベクトルレジスタの要素を抽出して、32 ビット単精度浮動小数点データとして返します。		
宣　言	`float __ev128_get_f32(__ev128_f32__ x,int eid);`		
引　数	[IN]	__ev128_f32__ x	：抽出を行うベクトルレジスタを指定します。
	[IN]	int eid	：抽出を行うベクトルレジスタの要素を指定します。
リターン値	float 型の値		：抽出した 32 ビット単精度浮動小数点データを返します。
備考			

3.4.2 データサイズ

FXU 命令は 4 要素ずつ処理を行います。そのため、重み行列データ列サイズ及びバイアスデータ列サイズが 4 の倍数でない場合、0 埋めを行い 4 の倍数に拡張してください。

図 3-5 にパターン A の行列×ベクトル+バイアス演算（入力要素数：10、出力要素数：10）において、0 埋めを行う例を示します。

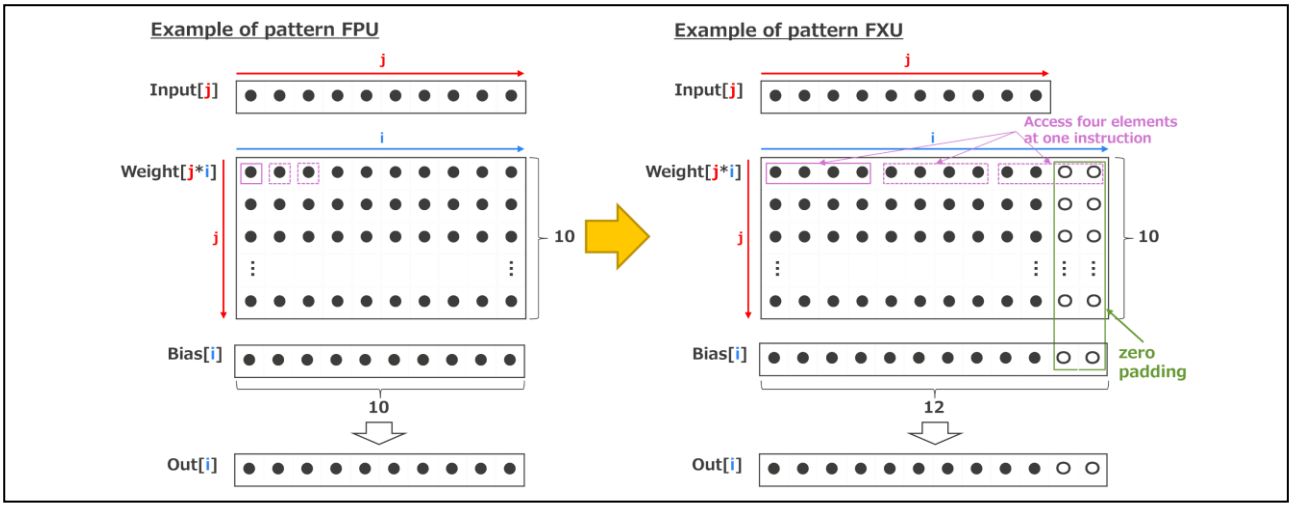


図 3-5 重み行列データ及びバイアスデータの 0 埋め

またデータサイズ変更に伴い、重み行列データ列サイズ及びバイアスデータ列サイズに対応するマクロ定数は 4 の倍数にする必要があります。同時に演算結果を格納するテンポラリ変数のサイズも 4 の倍数にする必要があります。本サンプルソフトでは、これらはすべて〇〇_OUTUNIT によって定義されています。本サンプルソフトの設定例を以下に示します。

表 3-10 FXU 使用時のマクロ定数設定

パターン		入力データ	中間層 1 の重み行列データ		中間層 2 の重み行列データ		出力層の重み行列データ	
			行サイズ	列サイズ	行サイズ	列サイズ	行サイズ	列サイズ
		INPUT_UNIT	MID1_INUNIT	MID1_OUTUNIT	MID2_INUNIT	MID2_OUTUNIT	OUTPUT_INUNIT	OUTPUT_OUTUNIT
FXU	A	10	10	12	10	12	10	12
	B	10	10	20	20	20	20	12
	C	10	10	32	30	32	30	12

3.4.3 アライメント指定

FXU 専用命令のソースまたはディスティネーションになるデータは、適切にアライメントする必要があります。そうでない場合、ミスアラインエラーが発生します。表 3-11 に適切なデータアライン条件を示します。

表 3-11 データアライン条件

実行命令		データアライン条件		
FXU 専用命令	データアクセスサイズ	32b	64b	128b
LDV.W, STV.W	32b	OK	OK	OK
LDV.DW, STV.DW, LDVZ.H4, STVZ.H4	64b	NG	OK	OK
LDV.QW, STV.QW	128b	NG	NG	OK

以下に GHS コンパイラでデータを 128 ビット（16 バイト）境界に配置する例について示します。

```
#pragma alignvar (16)
float data[8];
```

4. FPU と FXU の性能比較

本サンプルソフトにおいて、FPU 及び FXU を使用した場合の処理時間の計測と比較を行います。

4.1 測定条件

本計測例では、以下の条件で処理時間の計測を行っています。

- 処理時間の計測には OS タイマを使用します。

（1） コンパイル条件

- GHS コンパイラ v2021.1.5 使用
- オプション：-cpu=rh850g4mh -sda=all -large_sda -Ospeed -Onounroll -rh850_fxu -fastmath -prepare_dispose -no_callt

（2） 評価環境

- 統合開発環境：GHS MULTI
- エミュレータ：E2 エミュレータ
- 評価ボード：RH850/U2B-468BGA PiggyBack board (Y-RH850-U2B-468PIN-PB-T1-V1)
- マイコン：RH850/U2B10-FCC (R7F702Z21EDBG)

4.2 測定結果

FPU を使用して実行した場合と、FXU を使用して実行した場合の処理時間を表 4-1 に示します。本計測では、層数を増やしたパターン（C-2、C-3）の計測も追加しています。また横軸を積和演算（FMA）回数としてプロットしたグラフを図 4-1 に示します。

表 4-1 処理時間計測結果

パターン		ユニット数				処理実行回数			処理時間 [us]	
		入力層	中間層	出力層	層数	FMA	tanh	exp	FPU	FXU
tanh	A	10	10	10	3	300	30	0	**11.0**	**12.5**
	B	10	20	10	3	800	50	0	**20.9**	**19.4**
	C-1	10	30	10	3	1500	70	0	**31.5**	**27.9**
	C-2	10	30	10	5	3300	130	0	**61.9**	**51.0**
	C-3	10	30	10	7	5100	190	0	**92.4**	**74.2**
tanh_usingexp	A	10	10	10	3	300	0	30	**7.5**	**8.0**
	B	10	20	10	3	800	0	50	**14.9**	**11.7**
	C-1	10	30	10	3	1500	0	70	**24.1**	**18.1**
	C-2	10	30	10	5	3300	0	130	**49.9**	**34.6**
	C-3	10	30	10	7	5100	0	190	**75.8**	**51.4**
sigmoid	A	10	10	10	3	300	0	30	**7.0**	**8.3**
	B	10	20	10	3	800	0	50	**14.3**	**12.1**
	C-1	10	30	10	3	1500	0	70	**23.1**	**18.4**
	C-2	10	30	10	5	3300	0	130	**47.5**	**35.0**
	C-3	10	30	10	7	5100	0	190	**71.9**	**51.7**
ReLU	A	10	10	10	3	300	0	0	**3.4**	**3.3**
	B	10	20	10	3	800	0	0	**8.1**	**4.6**
	C-1	10	30	10	3	1500	0	0	**14.4**	**7.5**
	C-2	10	30	10	5	3300	0	0	**31.2**	**14.4**
	C-3	10	30	10	7	5100	0	0	**48.0**	**21.4**

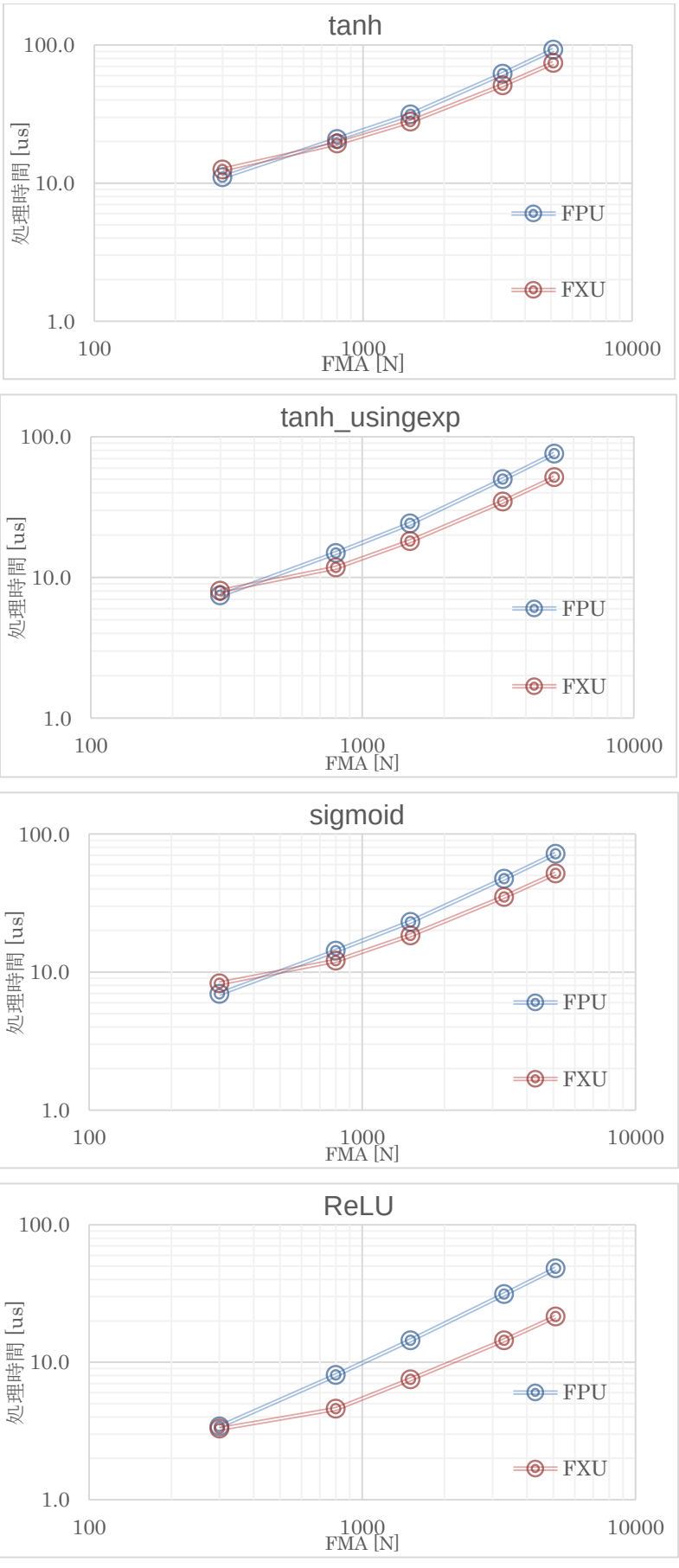


図 4-1 処理時間計測結果グラフ

5. Appendix

5.1 RH850/U2Bx シリーズの CPU 構成

表 5-1 に RH850/U2Bx の CPU 構成を示します。

各製品で、FXU 搭載 CPU の配置が異なるため、ご注意ください。

表 5-1 RH850/U2Bx の CPU 構成

Cluster	CPU (PEID)	U2B6	U2B10	
		3+2	4+2	3+3
0	0	DCLS **w/ FXU**	DCLS **w/ FXU**	DCLS **w/ FXU**
	1	DCLS	DCLS	DCLS
1	2	SNGL	SNGL **w/ FXU** *1	DCLS **w/ FXU** *1
	3	-	SNGL	-

【注】 DCLS：Dual Core Lockstep Core
SNGL：Single Core
Note 1. FXU is only in FCC device.

改訂記録

Rev.	発行日	改訂内容	
		ページ	ポイント
1.00	2024.7.15	-	初版発行

製品ご使用上の注意事項

ここでは、マイコン製品全体に適用する「使用上の注意事項」について説明します。個別の使用上の注意事項については、本ドキュメントおよびテクニカルアップデートを参照してください。

1. 静電気対策
   CMOS 製品の取り扱いの際は静電気防止を心がけてください。CMOS 製品は強い静電気によってゲート絶縁破壊を生じることがあります。運搬や保存の際には、当社が出荷梱包に使用している導電性のトレーやマガジンケース、導電性の緩衝材、金属ケースなどを利用し、組み立て工程にはアースを施してください。プラスチック板上に放置したり、端子を触ったりしないでください。また、CMOS 製品を実装したボードについても同様の扱いをしてください。
2. 電源投入時の処置
   電源投入時は、製品の状態は不定です。電源投入時には、LSI の内部回路の状態は不確定であり、レジスタの設定や各端子の状態は不定です。外部リセット端子でリセットする製品の場合、電源投入からリセットが有効になるまでの期間、端子の状態は保証できません。同様に、内蔵パワーオンリセット機能を使用してリセットする製品の場合、電源投入からリセットのかかる一定電圧に達するまでの期間、端子の状態は保証できません。
3. 電源オフ時における入力信号
   当該製品の電源がオフ状態のときに、入力信号や入出力プルアップ電源を入れないでください。入力信号や入出力プルアップ電源からの電流注入により、誤動作を引き起こしたり、異常電流が流れ内部素子を劣化させたりする場合があります。資料中に「電源オフ時における入力信号」についての記載のある製品は、その内容を守ってください。
4. 未使用端子の処理
   未使用端子は、「未使用端子の処理」に従って処理してください。CMOS 製品の入力端子のインピーダンスは、一般に、ハイインピーダンスとなっています。未使用端子を開放状態で動作させると、誘導現象により、LSI 周辺のノイズが印加され、LSI 内部で貫通電流が流れたり、入力信号と認識されて誤動作を起こす恐れがあります。
5. クロックについて
   リセット時は、クロックが安定した後、リセットを解除してください。プログラム実行中のクロック切り替え時は、切り替え先クロックが安定した後に切り替えてください。リセット時、外部発振子(または外部発振回路)を用いたクロックで動作を開始するシステムでは、クロックが十分安定した後、リセットを解除してください。また、プログラムの途中で外部発振子(または外部発振回路)を用いたクロックに切り替える場合は、切り替え先のクロックが十分安定してから切り替えてください。
6. 入力端子の印加波形
   入力ノイズや反射波による波形歪みは誤動作の原因になりますので注意してください。CMOS 製品の入力がノイズなどに起因して、V_{IL}(Max.)から V_{IH}(Min.)までの領域にとどまるような場合は、誤動作を引き起こす恐れがあります。入力レベルが固定の場合はもちろん、V_{IL}(Max.)から V_{IH}(Min.)までの領域を通過する遷移期間中にチャタリングノイズなどが入らないように使用してください。
7. リザーブアドレス(予約領域)のアクセス禁止
   リザーブアドレス(予約領域)のアクセスを禁止します。アドレス領域には、将来の拡張機能用に割り付けられている リザーブアドレス(予約領域)があります。これらのアドレスをアクセスしたときの動作については、保証できませんので、アクセスしないようにしてください。
8. 製品間の相違について
   型名の異なる製品に変更する場合は、製品型名ごとにシステム評価試験を実施してください。同じグループのマイコンでも型名が違うと、フラッシュメモリ、レイアウトパターンの相違などにより、電気的特性の範囲で、特性値、動作マージン、ノイズ耐量、ノイズ幅射量などが異なる場合があります。型名が違う製品に変更する場合は、個々の製品ごとにシステム評価試験を実施してください。

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。回路、ソフトウェアおよびこれらに関連する情報を使用する場合、お客様の責任において、お客様の機器・システムを設計ください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品または本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を組み込んだ製品の輸出入、製造、販売、利用、配布その他の行為を行うにあたり、第三者保有の技術の利用に関するライセンスが必要となる場合、当該ライセンス取得の判断および取得はお客様の責任において行ってください。
5. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
6. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準：　コンピュータ、OA 機器、通信機器、計測機器、AV 機器、家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準：輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment 向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
7. あらゆる半導体製品は、外部攻撃からの安全性を 100%保証されているわけではありません。当社ハードウェア／ソフトウェア製品にはセキュリティ対策が組み込まれているものもありますが、これによって、当社は、セキュリティ脆弱性または侵害（当社製品または当社製品が使用されているシステムに対する不正アクセス・不正使用を含みますが、これに限りません。）から生じる責任を負うものではありません。当社は、当社製品または当社製品が使用されたあらゆるシステムが、不正な改変、攻撃、ウイルス、干渉、ハッキング、データの破壊または窃盗その他の不正な侵入行為（「脆弱性問題」といいます。）によって影響を受けないことを保証しません。当社は、脆弱性問題に起因しまたはこれに関連して生じた損害について、一切責任を負いません。また、法令において認められる限りにおいて、本資料および当社ハードウェア／ソフトウェア製品について、商品性および特定目的との合致に関する保証ならびに第三者の権利を侵害しないことの保証を含め、明示または黙示のいかなる保証も行いません。
8. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
9. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment 向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
10. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制する RoHS 指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
11. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
12. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
13. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
14. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注 1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注 2. 本資料において使用されている「当社製品」とは、注 1 において定義された当社の開発、製造製品をいいます。

(Rev.5.0-1　2020.10)

本社所在地

〒135-0061 東京都江東区豊洲 3-2-24（豊洲フォレシア）

www.renesas.com

お問合せ窓口

弊社の製品や技術、ドキュメントの最新情報、最寄の営業お問合せ窓口に関する情報などは、弊社ウェブサイトをご覧ください。

www.renesas.com/contact/

商標について

ルネサスおよびルネサスロゴはルネサス エレクトロニクス株式会社の商標です。すべての商標および登録商標は、それぞれの所有者に帰属します。