RH850/U2Bx

LSTM Network

Summary

This application note describes LSTM (long short-term memory) using the Floating-Point Unit (FPU) and the Extended Floating-Point Unit (FXU) incorporated in RH850/U2Bx.

This application note does not include the specification detail information of FXU. Please refer to the APN “FXU Use For FP-SIMD Calculations” for the details. Also, please check the product specifications before using since the presence or absence of FXU and the position of the CPU equipped with FXU differ depending on the product. Refer to the appendix for the details.

Although the operation of the LSTM network example described in this application note has been confirmed, but please sure to confirm the operation before using it.

Checked Operation Device

RH850/U2Bx

Contents

1. LSTM Network Overview

1.1 LSTM Network

LSTM (Long short-term memory) network is one of the recurrent neural networks using the short-term memory and the long-term memory. The operation example of this application note is configured by an input layer and the two LSTM layers. (Figure 1-1).

Figure 1-2 shows the configuration of the LSTM layer. The LSTM layer is configured by the forget gate, input gate, long-term memory updating, and output gate. Updates the long-term memory "Ct" and calculates the output value "ht" based on the input value "xt" and the previous output value ht-1 (short-term memory).

In the LSTM layer, use the σ (sigmoid) function and the tanh function as the activation function. The following shows each formula.

$$\sigma = \frac{1}{1 + e^{-x}}$$

$$\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

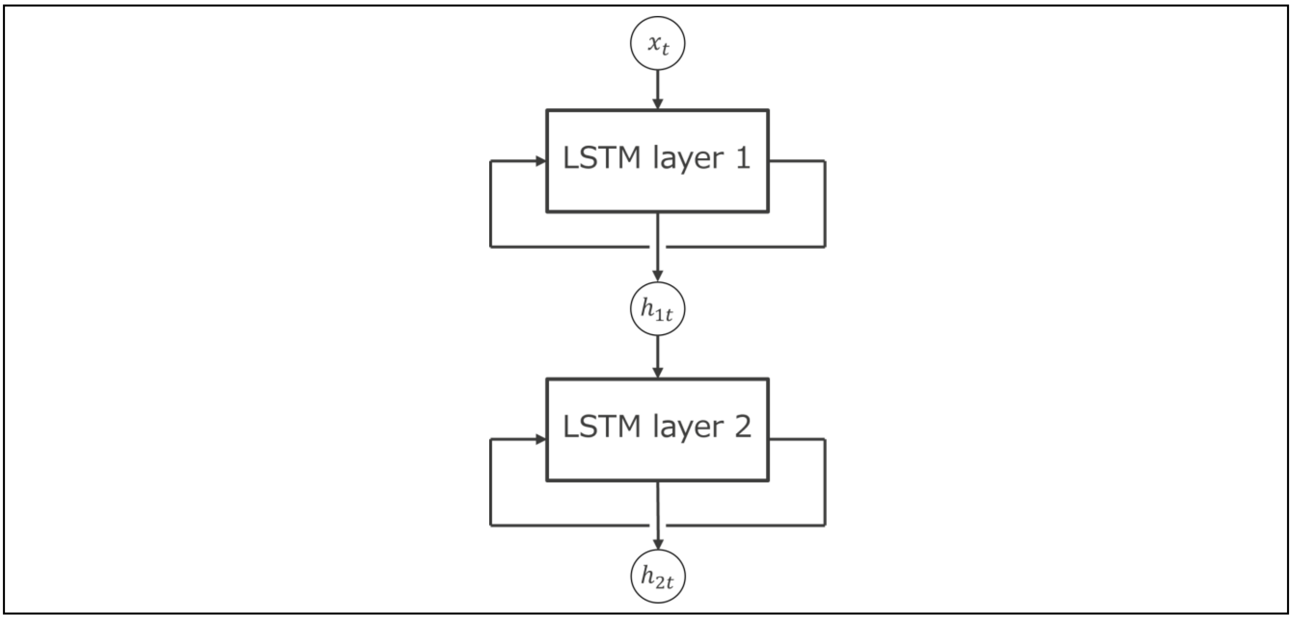


Figure 1-1 LSTM Network

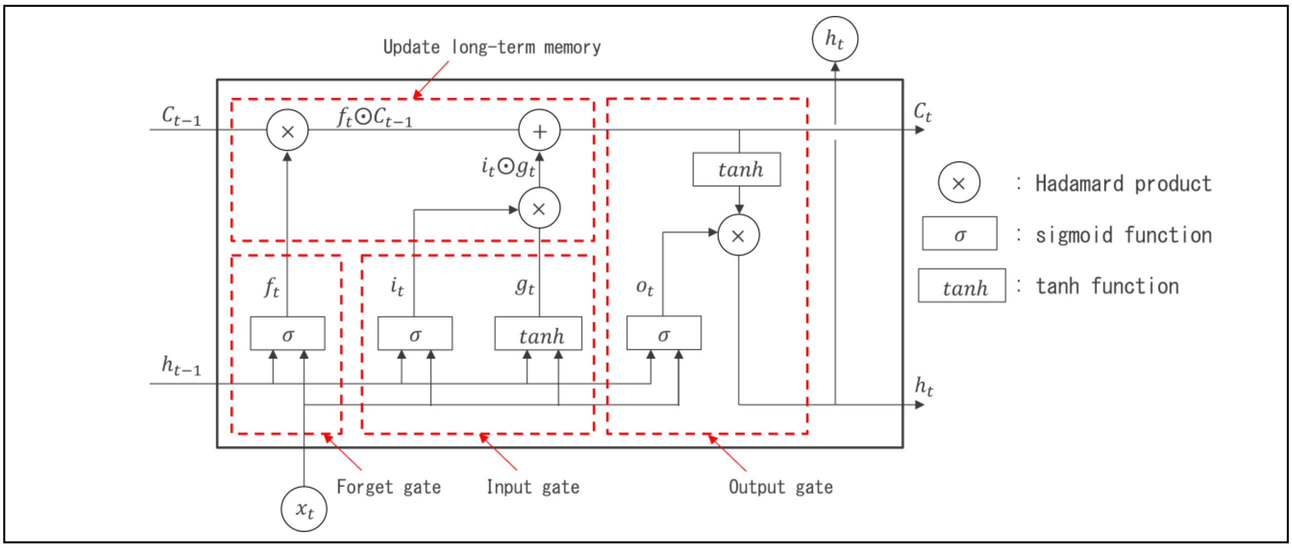


Figure 1-2 LSTM Layer Configuration Diagram

1.1.1 Forget Gate

The red-flamed part of Figure 1-3 is the forget gate. The gate is for scraping the unnecessary information from the long-term memory "Ct-1" based on the previous output value "ht-1" and the current input value "xt". In σ of the diagram, performs the σ (sigmoid) processing of the activation function for the sum of the product of the input "xt" and the weight matrix "Wf", the previous output value "ht-1" and the weight matrix "Rf", and the bias value. The formula is shown below.

$$f_t = \sigma(W_f x_t + R_f h_{t-1} + b_f)$$

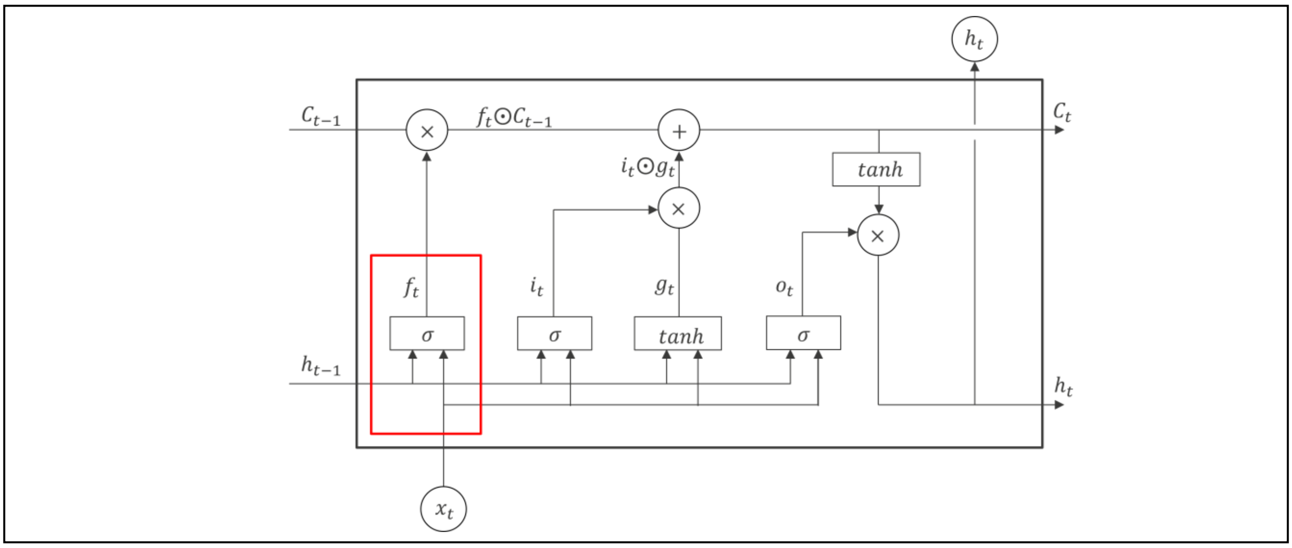


Figure 1-3 Forget Gate

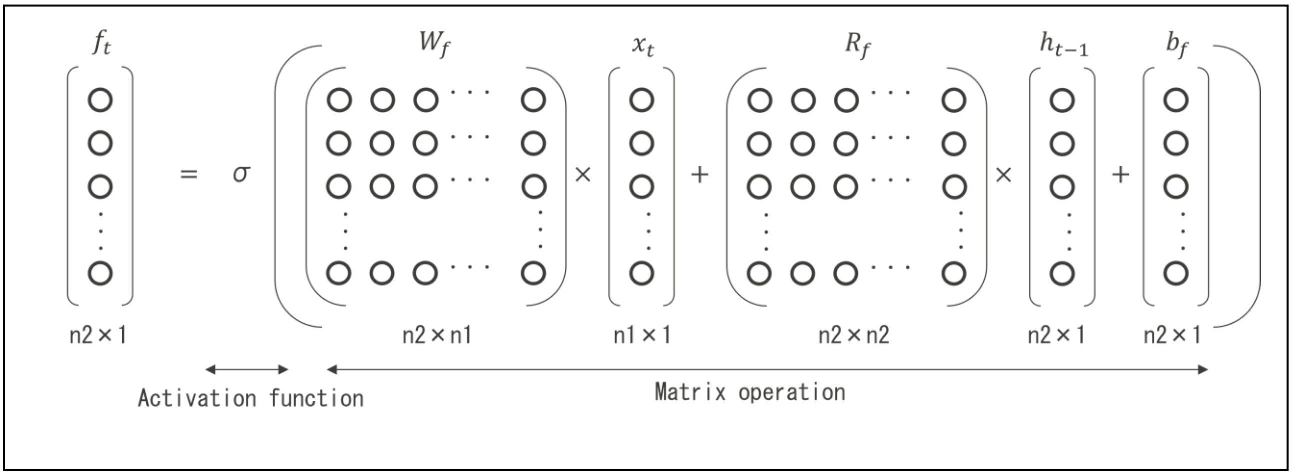


Figure 1-4 Formula of Forget Gate

1.1.2 Input Gate

The red-flamed part of Figure 1-5 is the input gate. The gate is for saving the input data combining “ht-1” and “xt” to the long-term memory “Ct”. The formula is shown below.

$$g_t = \tanh\left(W_g x_t + R_g h_{t-1} + b_g\right)$$
$$i_t = \sigma(W_i x_t + R_i h_{t-1} + b_i)$$

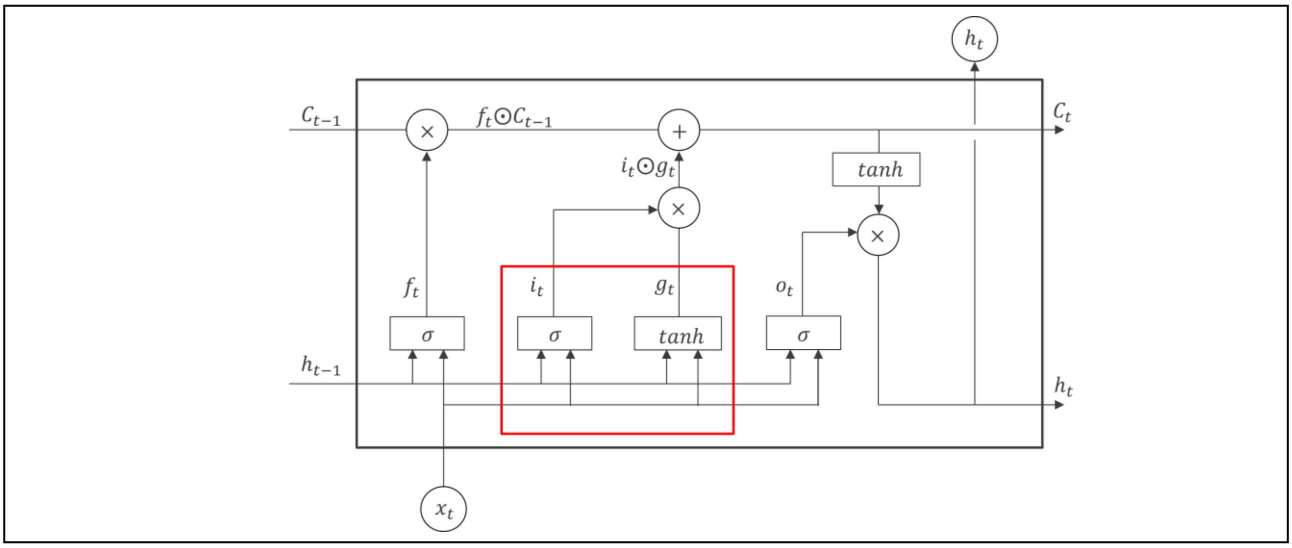


Figure 1-5 Input Gate

1.1.3 Update of Long-term Memory

The red-flamed part of Figure 1-6 is the updates of the long-term memory “Ct”. Updates the status of the long-term memory by the forget gate and the input gate. The formula is shown below.

$$C_t = f_t \odot C_{t-1} + i_t \odot g_t \quad : \odot \text{ is Hadamard Product.}$$

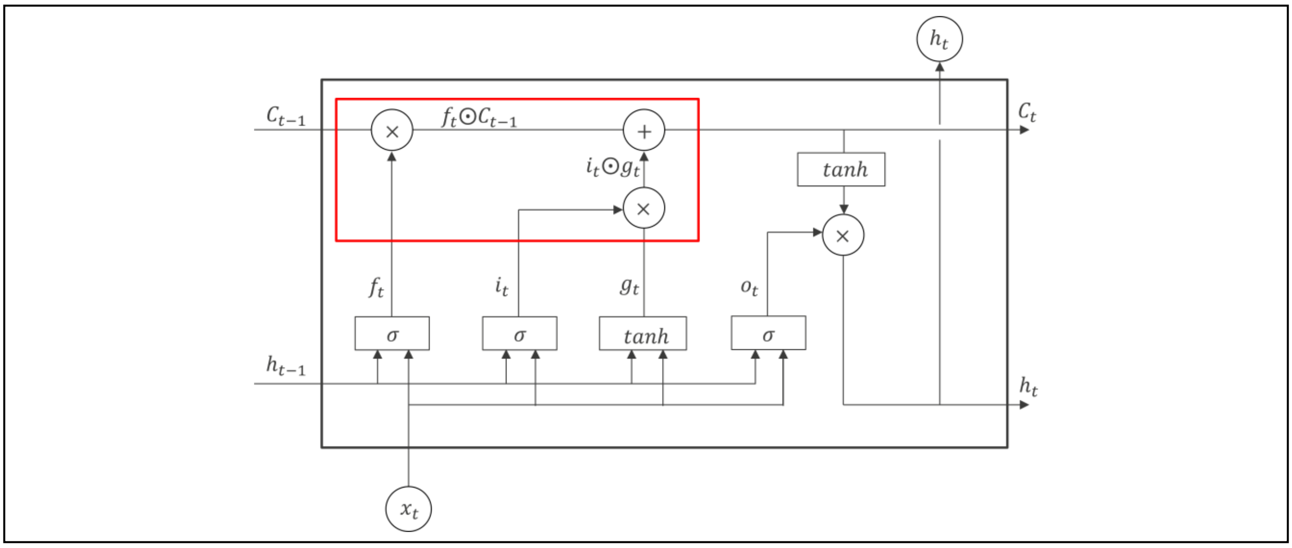


Figure 1-6 Update of Long-term Memory

1.1.4 Output Gate

The red-flamed part of Figure 1-7 is the output gate. Extracts the short-term memory “ht” from the inside of long term-memory based on “ht-1” and “xt”. The formula is shown below.

$$o_t = \sigma(W_o x_t + R_o h_{t-1} + b_o)$$
$$h_t = o_t \odot \tanh(C_t)$$

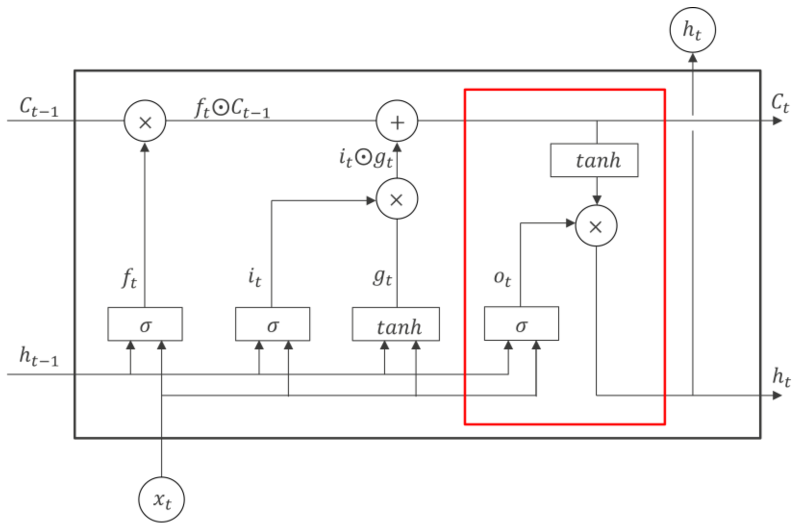


Figure 1-7 Output Gate

1.2 Support Function

This sample software supports the following functions.

Table 1-1 Support Function List

Function	FPU	FXU
Forget gate	forgetgate	forgetgate_fxu
Input gate	inputgate	inputgate_fxu
Update of long-term memory	update_longterm_memory	update_longterm_memory_fxu
Output gate	outputgate	outputgate_fxu

1.3 Use Hardware Function

The hardware functions of RH850/U2Bx using in this sample software are shown below.

- Floating-Point Unit (FPU)
- Extended Floating-Point Unit (FXU)
- Various memories (Code Flash, Cluster RAM, Local RAM)

This sample software performs the processing by inside of a cluster (Cluster #0) using CPU0. Refer to "2.4 Allocation of Constant and Variable エラー! 参照元が見つかりません。" for the details of the constant and variable data allocation.

This sample software supports single precision (32-bit).

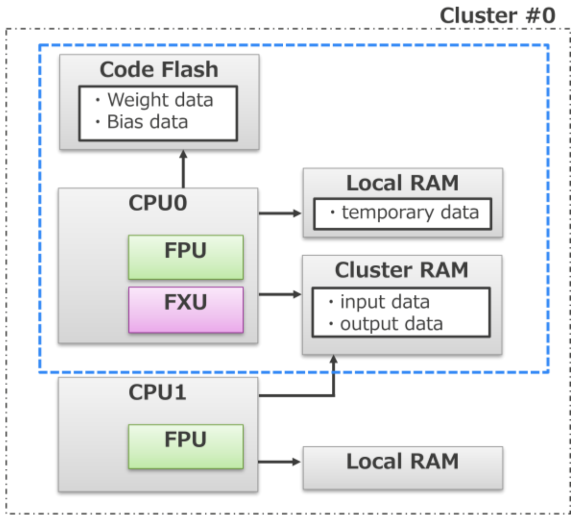


Figure 1-8 System Configuration

2. Software Explanation

2.1 Operation Flow

The operation flow in this sample software is shown below.

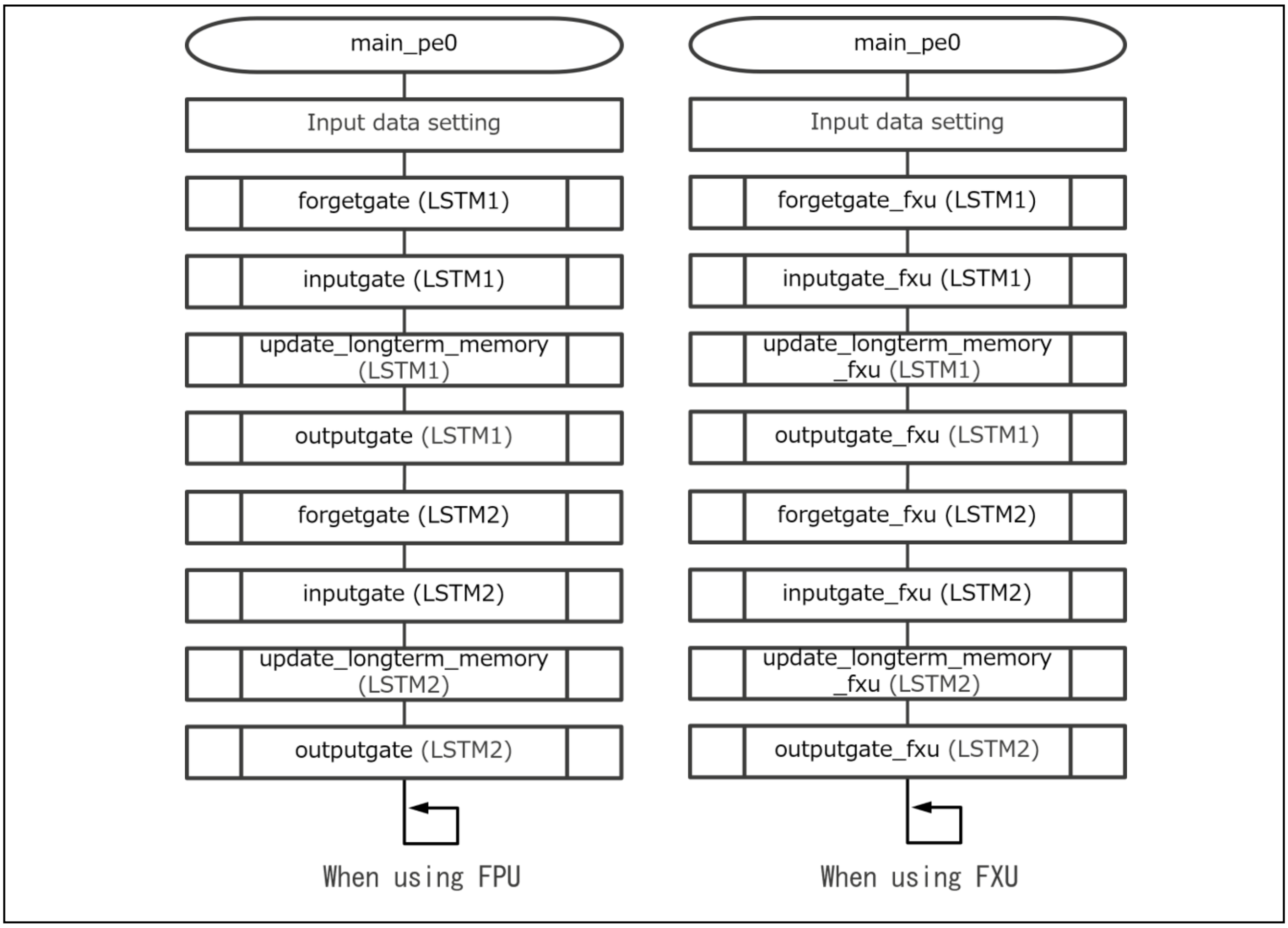


Figure 2-1 Operation Flow

2.2 Sample Software Configuration

Table 2-1 shows the file configuration of the sample software.

Table 2-1 Sample Software File Configuration

File Name					Overview
FNN	FPU	U2Bx_Sample.gpj			Master project file
		U2B10_Sample.gpj			Master project file
		src	U2B10_Sample.gpj		Project file
			core0	lstm_fpu.c	Function for LSTM.
				weight_data_fpu(a/b/c).c	File of each pattern Refer to ”2.5 エラー! 参照元が見つかりません。
				weight_data_fpu(a/b/c).h	Header file of each pattern Refer to ”2.5 エラー! 参照元が見つかりません。”
				sub_timer_benchmark.c	File for processing load measurement
				sub_timer_benchmark.h	Header file for processing load measurement
				main_pe0.c	main function for CPU0
				intprg.c	Interrupt processing function No particular processing content
			core1	main_pe0.c	main function for CPU1
				intprg.c	Interrupt processing function No particular processing content
			core2	main_pe0.c	main function for CPU2
				intprg.c	Interrupt processing function No particular processing content
			core3	main_pe0.c	main function for CPU3
				intprg.c	Interrupt processing function No particular processing content
		startup			Start-up routine
	FXU	U2Bx_Sample.gpj			Master project file
		U2B10_Sample.gpj			Master project file
		src	U2B10_Sample.gpj		Project file
			core0	lstm_fxu.c	Function for LSTM.
				weight_data_fxu(a/b/c).c	File of each pattern Refer to ”2.5 エラー! 参照元が見つかりません。

				weight_data_fxu(a/b/c).h	Header file of each pattern Refer to ”2.5 エラー! 参照元が見つかりません。”
				sub_timer_benchmark.c	File for processing load measurement
				sub_timer_benchmark.h	Header file for processing load measurement
				main_pe0.c	main function for CPU0
				intprg.c	Interrupt processing function No particular processing content
			core1	main_pe0.c	main function for CPU1
				intprg.c	Interrupt processing function No particular processing content
			core2	main_pe0.c	main function for CPU2
				intprg.c	Interrupt processing function No particular processing content
			core3	main_pe0.c	main function for CPU3
				intprg.c	Interrupt processing function No particular processing content
		startup			Start-up routine

2.3 Function Specification

2.3.1 Function for FPU

Table 2-2 shows the functions list of FPU version in this operation example.

Table 2-2 Functions List of FPU Version

Function Name	Overview
main_pe0	Performs the call of each function.
forgetgate	Performs the forget gate processing.
inputgate	Performs the input gate processing.
update_longterm_memory	Updates the long-terms memory.
outputgate	Performs the output gate processing.

Table 2-3 to Table 2-6 show the functions operations of FPU version in this operation example

Table 2-3 Specification of Forget Gate

Forget Gate			
Overview	Performs the forget gate processing and stores the result to the specified array.		
Declaration	`void forgetgate(float input[], float ht[], const float wf[],const float rf[], const float bf[], float output[], unsigned int size_in, unsigned int size_hidden, unsigned int size_out);`		
Argument	[IN]	float input[]	: Specifies the input data.
	[IN]	float ht[]	: Specifies the short-term memory data.
	[IN]	float wf[]	: Specifies the weight matrix data “Wf” of the forget gate processing.
	[IN]	float rf[]	: Specifies the weight matrix data “Rf” of the forget gate processing.
	[IN]	float bf[]	: Specifies the bias data “Rf” of the forget gate processing.
	[OUT]	float output[]	: Stores the result of the forget gate processing.
	[IN]	unsigned int size_in	: Specify the input data (input[]) and the weight array data (wf[]).
	[IN]	unsigned int size_hidden	: Specifies the array size of the weight array data (rf[]).
	[IN]	unsigned int size_out	: Specify the array size of each weight matrix data (wf[], rf[]), the bias data (bf[]) size, the short-term memory data size, and the output data (output[]) size.
Return value	-		
Remarks	- Allocate the weight matrix data specified in the argument in the transposed state. (Refer to “3.3 Constant Data Placement to Code Flash”) - Please note the input range since this function uses the expf function and the output is e^n for the input n.		

Table 2-4 Specification of inputgate Function

inputgate	
Overview	Performs the input gate processing and stores the result to the specified array.
Declaration	`void inputgate(float input[], float ht[], const float wi[],const float ri[], const float bi[],const float wg[],const float rg[], const float bg[], float output_it[], float output_gt[], unsigned int size_in, unsigned int size_hidden, unsigned int size_out);`
Argument	[IN] float input[] : Specifies the input data. [IN] float ht[] : Specifies the short-term memory data. [IN] float wi[] : Specifies the weight matrix data “Wi” of the input gate processing (it). [IN] float ri[] : Specifies the weight matrix data “Ri” of the input gate processing (it). [IN] float bi[] : Specifies the bias data “bi” of the input gate processing (it). [IN] float wg[] : Specifies the weight matrix data “Wg” of the input gate processing (gt). [IN] float rg[] : Specifies the weight matrix data “Rg” of the input gate processing (gt). [IN] float bg[] : Specifies the weight matrix data “bg” of the input gate processing (gt). [OUT] float output_it[] : Stores the result “it” of the input gate processing. [OUT] float output_gt[] : Stores the result “gt“ of the input gate processing. [IN] unsigned int size_in : Specify the input data (input[]) size and the array sizes of the weight matrix data (wi[], wg[]). [IN] unsigned int size_hidden : Specify the array seizes of the weight matrix data (ri[], rg[]). [IN] unsigned int size_out : Specify the array size of each weight matrix data (wf[], ri[], wg[], rg[]), each bias data size (bi[], bg[]), the short-term memory data (ht[]) size, and the output data (output_it[], output_gt[]) sizes.
Return value	-
Remarks	- Allocate the weight matrix data specified in the argument in the transposed state. (Refer to “3.3 Constant Data Placement to Code Flash”) - According to the following, calculates tanh using the formula with the exponential function. $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ - Please note the input range since this function uses the expf function and the output is e^n for the input n.

Table 2-5 Specification of update_longterm_memory Function

update_longterm_memory			
Overview	Performs the updates of the long-term memory and stores the result to the specified array.		
Deceleration	`void update_longterm_memory(float Ct[], float ft[],float it[],float gt[],unsigned int size_out);`		
Argument	[IN, OUT]	float Ct[]	: Specifies the long-term memory data.
	[IN]	float ft[]	: Specifies the forget gate processing result.
	[IN]	float it[]	: Specifies the input gate processing result “it”.
	[IN]	float gt[]	: Specifies the input gate processing result “gt”.
	[IN]	unsigned int size_out	: Specify each data (Ct[], ft[], it[], gt[]) size specified as the argument.
Return value	-		
Remarks			

Table 2-6 Specification of outputgate Function

outputgate			
Overview	Performs the output gate processing and stores the result to the specified array.		
Deceleration	`void outputgate(float input[], float ht[], float Ct[], const float wo[], const float ro[], const float bo[], float output[], unsigned int size_in, unsigned int size_hidden, unsigned int size_out);`		
Argument	[IN]	float input[]	: Specifies the input data.
	[IN]	float ht[]	: Specifies the short-term memory data.
	[IN]	float Ct[]	: Specifies the long-term memory data.
	[IN]	float wo[]	: Specifies the weight matrix data “Wo” of the input gate processing.
	[IN]	float ro[]	: Specifies the weight matrix data “Ro” of the input gate processing.
	[IN]	float bo[]	: Specifies the bias data of the input gate processing.
	[OUT]	float output[]	: Stores the result of the forget gate processing.
	[IN]	unsigned int size_in	: Specify the input data (input[]) size and the array size of the weight matrix data (wo[]).
	[IN]	unsigned int size_hidden	: Specify the array size of the weight matrix data (ro[]).
	[IN]	unsigned int size_out	: Specify the array size of each weight matrix data (wo[], ro[]), each bias data size (bo[]), the short/long-term memory data (ht[], ct[]) sizes, and the output data (output[]) size.
Return value	-		
Remarks	- Allocate the weight matrix data specified in the argument in the transposed state. (Refer to “3.3 Constant Data Placement to Code Flash”) - According to the following, calculates tanh using the formula with the exponential function. $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ - Please note the input range since this function uses the expf function and the output is e^n for the input n.		

2.3.2 FXU Version Function

Table 2-7 shows the functions list of the FXU versions using in this operation.

In this sample software, the built-in functions of FXU instructions standardly supported in GHS. Refer to “3.4.1.2 Details of FXU Built-in Function” for the built-in function details.

Table 2-7 Function List of FXU Version

Function Name	Overview
main_pe0	Performs the call of each function.
forgetgate_fxu	Performs the forget gate processing using FXU.
inputgate_fxu	Performs the input gate processing using FXU.
update_longterm_memory_fxu	Performs the updates of the long-term memory using FXU.
outputgate_fxu	Performs the output gate processing using FXU.
expf_vector	Performs the expf function processing for each vector element.
tanhf_vector	Performs the tanhf function processing for each vector element.

Table 2-8 to Table 2.13 show the functions operations for FXU version using in this operation example.

Table 2-8 Specification of forgetgate_fxu Function

forgetgate_fxu			
Overview	Performs the forget gate processing to use FXU, and stores the result to the specified array.		
Declaration	`void forgetgate_fxu(float input[], float ht[], const float wf[],const float rf[], const float bf[], float output[], unsigned int size_in, unsigned int size_hidden, unsigned int size_out);`		
Argument	[IN]	float input[]	: Specifies the input data.
	[IN]	float ht[]	: Specifies the short-term memory data.
	[IN]	float wf[]	: Specifies the weight matrix data “Wf” of the forget gate processing.
	[IN]	float rf[]	: Specifies the weight matrix data “Rf” of the forget gate processing.
	[IN]	float bf[]	: Specifies the bias data “Rf” of the forget gate processing.
	[OUT]	float output[]	: Stores the result of the forget gate processing.
	[IN]	unsigned int size_in	: Specify the input data (input[]) and the weight array data (wf[]).
	[IN]	unsigned int size_hidden	: Specifies the array size of the weight array data (rf[]).
	[IN]	unsigned int size_out	: Specify the array size of each weight matrix data (wf[], rf[]), the bias data (bf[]) size, the short-term memory data size, and the output data (output[]) size.
Return value	-		
Remarks	- Allocate the weight matrix data specified in the argument in the transposed state. (Refer to “3.3 Constant Data Placement to Code Flash”.) - If the column size of the weight matrix data and the size of the bias data specified in the argument are not multiples of four, zero pad them until the size is a multiple of four. At the same time, make the argument size_out a multiple of four. (Refer to “3.4.2 Data Size”.) - Allocate the start address of the specified data of the argument: input[], weight[], bias[], and output[] to the 16Byte boundary. (Refer to “3.4.3 Alignment Specification”.) - Please note the input range since this function uses the expf function and the output is e^n for the input n.		

Table 2-9 Specification of inputgate_fxu Function

inputgate_fxu			
Overview	Performs the input gate processing using FXU and stores the result to the specified array.		
Declaration	`void inputgate_fxu(float input[], float ht[], const float wi[],const float ri[], const float bi[],const float wg[],const float rg[], const float bg[], float output_it[], float output_gt[], unsigned int size_in, unsigned int size_hidden, unsigned int size_out);`		
Argument	[IN]	float input[]	: Specifies the input data.
	[IN]	float ht[]	: Specifies the short-term memory data.
	[IN]	float wi[]	: Specifies the weight matrix data “Wi” of the input gate processing (it).
	[IN]	float ri[]	: Specifies the weight matrix data “Ri” of the input gate processing (it).
	[IN]	float bi[]	: Specifies the bias data “bi” of the input gate processing (it).
	[IN]	float wg[]	: Specifies the weight matrix data “Wg” of the input gate processing (gt).
	[IN]	float rg[]	: Specifies the weight matrix data “Rg” of the input gate processing (gt).
	[IN]	float bg[]	: Specifies the weight matrix data “bg” of the input gate processing (gt).
	[OUT]	float output_it[]	: Stores the result “it” of the input gate processing.
	[OUT]	float output_gt[]	: Stores the result “gt“ of the input gate processing.
	[IN]	unsigned int size_in	: Specify the input data (input[]) size and the array sizes of the weight matrix data (wi[], wg[]).
	[IN]	unsigned int size_hidden	: Specify the array seizes of the weight matrix data (ri[], rg[]).
	[IN]	unsigned int size_out	: Specify the array size of each weight matrix data (wf[], ri[], wg[], rg[]), each bias data size (bi[], bg[]), the short-term memory data (ht[]) size, and the output data (output_it[], output_gt[]) sizes.
Return value	-		
Remarks	- Allocate the weight matrix data specified in the argument in the transposed state. (Refer to “3.3 Constant Data Placement to Code Flash”.) - If the column size of the weight matrix data and the size of the bias data specified in the argument are not multiples of four, zero pad them until the size is a multiple of four. At the same time, make the argument size_out a multiple of four. (Refer to “3.4.2 Data Size”.) - Allocate the start address of the specified data of the argument: input[], ht[], wi[], ri[], bi[], wg[], rg[], bg[], output_it[], and output_gt[] to the 16Byte boundary. (Refer to “3.4.3 Alignment Specification”.) - According to the following, calculates tanh using the formula with the exponential function. $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ - Please note the input range since this function uses the expf function and the output is e^n for the input n.		

Table 2-10 Specification of update_longterm_memory_fxu Function

update_longterm_memory_fxu			
Overview	Updates the long-term memory using FXU and stores the result to the specified array.		
Declaration	`void update_longterm_memory_fxu(float Ct[], float ft[],float it[],float gt[],unsigned int size_out);`		
Argument	[INOUT]	float Ct[]	: Specifies the long-term memory data.
	[IN]	float ft[]	: Specifies the forget gate processing result.
	[IN]	float it[]	: Specifies the input gate processing result “it”.
	[IN]	float gt[]	: Specifies the input gate processing result “gt”.
	[IN]	unsigned int size_out	: Specifies each data (Ct[], ft[], it[], gt[]) size to the argument.
Return value	-		
Remarks	- Set the multiple of four to the data size specified to the argument Ct[], ft[], it[], and gt[]. At the same time, make the argument size_out a multiple of four. (Refer to “3.4.2 Data Size”.) - Allocate the start address of the specified data of the argument: Ct[], ft[], it[], and gt[] to the 16Byte boundary. (Refer to “3.4.3 Alignment Specification”.		

Table 2-11 Specification of outputgate_fxu Function

outputgate_fxu			
Overview	Performs the output gate processing using FXU and stores the result to the specified array.		
Declaration	`void outputgate_fxu(float input[], float ht[], float Ct[], const float wo[], const float ro[], const float bo[], float output[], unsigned int size_in, unsigned int size_hidden, unsigned int size_out);`		
Argument	[IN]	float input[]	: Specifies the input data.
	[IN]	float ht[]	: Specifies the short-term memory data.
	[IN]	float Ct[]	: Specifies the long-term memory data.
	[IN]	float wo[]	: Specifies the weight matrix data “Wo” of the input gate processing.
	[IN]	float ro[]	: Specifies the weight matrix data “Ro” of the input gate processing.
	[IN]	float bo[]	: Specifies the bias data “bo” of the forget gate processing.
	[OUT]	float output[]	: Stores the result of the forget gate processing.
	[IN]	unsigned int size_in	: Specify the input data (input[]) size and the array size of the weight matrix data (wo[]).
	[IN]	unsigned int size_hidden	: Specify the array size of the weight matrix data (ro[]).
	[IN]	unsigned int size_out	: Specify the array size of each weight matrix data (wo[], ro[]), each bias data size (bo[]), the short/long-term memory data (ht[], ct[]) sizes, and the output data (output[]) size.
Return value	-		
Remarks	- Allocate the weight matrix data specified in the argument in the transposed state. (Refer to “3.3 Constant Data Placement to Code Flash”.) - If the column size of the weight matrix data and the size of the bias data specified in the argument are not multiples of four, zero pad them until the size is a multiple of four. At the same time, make the argument size_out a multiple of four. (Refer to “3.4.2 Data Size”.) - Allocate the start address of the specified data of the argument: input[], ht[], Ct[], wo[], ro[], bo[], and output[] to the 16Byte boundary. (Refer to “3.4.3 Alignment Specification”. - According to the following, calculates tanh using the formula with the exponential function. $\tanh x = \frac{e^x - e^{-x}}{e^x + e^{-x}}$ - Please note the input range since this function uses the expf function and the output is e^n for the input n.		

Table 2-12 Specification of expf_vector Function

expf_vector			
Overview	Performs floating-point exponential calculation. Executes processing on each of the four elements of the vector specified in the argument and stores the result in the vector.		
Declaration	`__ev128_f32__ expf_vector(__ev128_f32__ x);`		
Argument	[IN]	__ev128_f32__ x	: Specifies the vector that performs the tanh function calculation.
Return value	Value of __ev128_f32__ type		: Specifies the vector that stores the specified function calculation result.
Remarks			

Table 2-13 Specification of tanhf_vector Function

tanhf_vector			
Overview	Performs floating-point tanh function calculation. Executes processing on each of the four elements of the vector specified in the argument and stores the result in the vector.		
Declaration	`__ev128_f32__ tanhf_vector(__ev128_f32__ x);`		
Argument	[IN]	__ev128_f32__ x	: Specifies the vector that performs the exponential calculation.
Return value	Value of __ev128_f32__ type		: Return the vector that is stored the calculation result of tanh function.
Remarks			

2.4 Change of The Data Size

This sample software is selectable to set the 3 pattern data both FPU and FXU.

Table 2-14 Pattern of the Data Size

Pattern		Data Size			File	
		Input Layer	LSTM Layer 1	LSTM Layer 2		
FPU	A	10	10	10	weight_data_fpua.h	weight_data_fpua.c
	B	10	30	30	weight_data_fpub.h	weight_data_fpub.c
	C	10	50	50	weight_data_fpuc.h	weight_data_fpuc.c
FXU*1	A	10	10	10	weight_data_fxua.h	weight_data_fxua.c
	B	10	30	30	weight_data_fxub.h	weight_data_fxub.c
	C	10	50	50	weight_data_fxuc.h	weight_data_fxuc.c

【Note】The column size of the weight matrix data and the bias data size for each layer must be the multiple of four since the FXU processes four elements at a time. In that case, extended them by filling the data element with zeros and change the macro constant that indicate the data size. These processes are applied to the sample files in advance. Refer to “3.4.2 Data Size.

When change the pattern, specify the header file and the constant data file corresponding the pattern.

(a) Change the definition of the macro name in main_pe0.c. This will change the header file to read.

Ex.) #define PATTERN_A when setting to the pattern A.

Specifies the constant data file in U2B10_Sample_src.gpj.

Ex.) .¥weight_data_fpua.c when setting to the pattern A (FPU).

2.5 Allocation of Constant and Variable

In this sample software, performs the processing in a cluster#0 using CPU0. As shown in Figure 2-2 and Table 2-15, allocate the input/output data to Cluster RAM, and the constant (weight matrix data, bias data) to Code Flash.

Please note that there is a possibility the processing performance is degreased caused by the data access delaying if the data is not allocated properly to the resource corresponding to the CPU used.

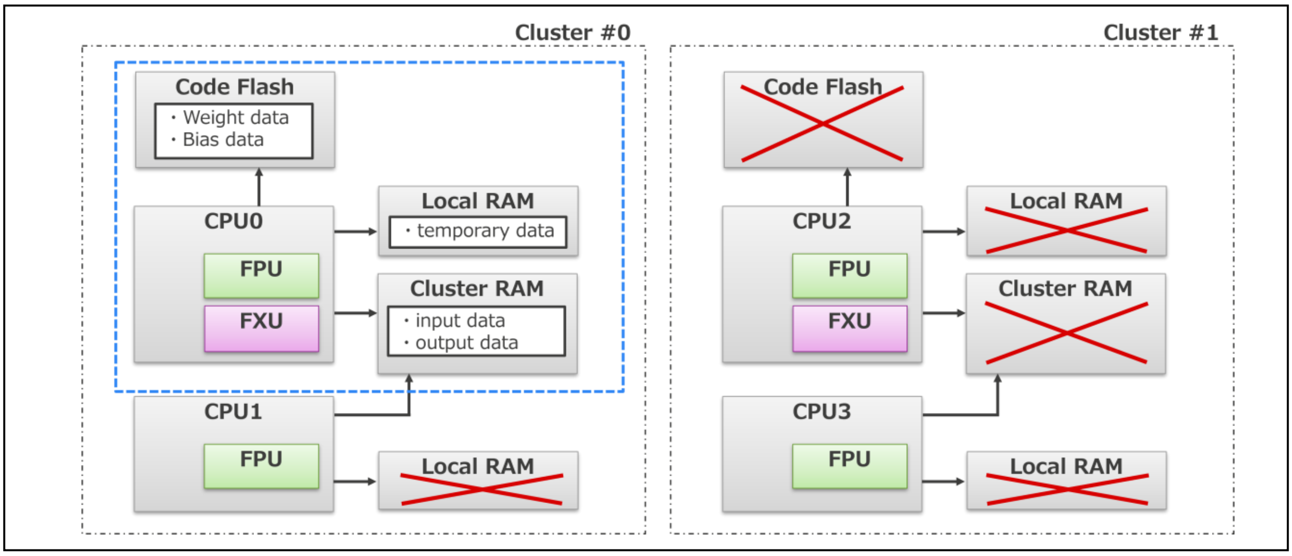


Figure 2-2 Allocation of Constant and Variable

Table 2-15 Type and Allocation of Constant and Variable

Type		Data Name	Constant/Variable	Allocation	Data Size *1
Input/output data		input_data	Global variable	Cluster RAM in the same cluster as the CPU used	■FPU version Pattern A: 0.2 KB Pattern B: 0.7 KB Pattern C: 1.2 KB ■FXU version Pattern A: 0.3 KB Pattern B: 0.8 KB Pattern C: 1.2 KB
		output_data			
Short-term memory data	LSTM layer 1	ht1			
	LSTM layer 2	ht2			
Long-term memory data	LSTM layer 1	Ct1			
	LSTM layer 2	Ct2			
Weight data	LSTM layer1	wf1	Constant	Code Flash in the same cluster as the CPU used	■FPU version Pattern A: 6.6 KB Pattern B: 47.8 KB Pattern C: 126.6 KB ■FXU version Pattern A: 9.0 KB Pattern B: 54.0 KB Pattern C: 136.5 KB
		wg1			
		wi1			
		wo1			
		rf1			
		rg1			
		ri1			
		ro1			
	LSTM layer 2	wf2			
		wg2			

		wi2			
		wo2			
		rf2			
		rg2			
		ri2			
		ro2			
Bias data	LSTM layer 1	bf1			
		bg1			
		bi1			
		bo1			
	LSTM layer 2	bf2			
		bg2			
		bi2			
		bo2			
Intermediate data	LSTM layer 1	ft1	Local variable	Local RAM	■FPU version Pattern A: 0.3 KB Pattern B: 0.8 KB Pattern C: 1.4 KB ■FXU version Pattern A: 0.3 KB Pattern B: 0.9 KB Pattern C: 1.4 KB
		it1			
		gt1			
		output_data_layer1			
	LSTM layer 2	ft2			
		it2			
		gt2			

【Note】 The total capacity used by each variable is shown in each pattern.

Precautions and Restrictions

FPU/FXU Initial Setting

The PSW register setting is required when using FPU/FXU. Also, the option byte setting is required when using FXU. Refer to each product user's manual for the detailed setting method.

Also, please check the product specifications since the presence or absence of FXU and the position of the CPU equipped with FXU differ depending on the product. Refer to the appendix for the details.

PSW Register

FPU：Enabled by setting the bit 16 (CU0) of the program status word (PSW) of the CPU to "1".

FXU：Enabled by setting the bit 17 (CU1) of the program status word (PSW) of the CPU to "1".

Bit	31	30	29	28	27	26	25–20	19	18–16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
PSW	0	UM	0	0	0	0	EIMASK	0	CU2-0	EBV	0	0	0	0	0	0	0	NP	EP	ID	SAT	CY	OV	S	Z

Value after reset $0000\ 0020_H$

Figure 3-1 PSW Register

Option Byte

FXU mounting CPU0：Enabled by setting the bit 16 (PE0_FPSIMD_EN) of the OPBT3 to "1".

FXU mounting CPU2：Enabled by setting the bit 18 (PE2_FPSIMD_EN) of the OPBT3 to "1".

Bit:	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
	—	—	PE5_DISABLE	PE4_DISABLE	PE3_DISABLE	PE2_DISABLE	PE1_DISABLE	—	—	—	—	—	—	PE2_FPSIMD_EN	—	PE0_FPSIMD_EN
Value after reset:	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]
R/W:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Bit:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
	HWBIST	—	—	—	TESTSET[1:0]		LBISTSEL[1:0]		—	—	—	—	—	—	STMSEL1	STMSEL0
Value after reset:	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]	0/1[*1]
R/W:	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W

Figure 3-2 Option Byte

Upper Limit and Lower Limit of Single Precision Floating-Point Type

The range of values that a single-precision floating-point type can represent is limited. Especially, when using an exponential function, it is necessary to note the input range because the output is e^n for the input n. In this sample software, the following function uses an exponential function.

forgetgate、inputgate、outputgate、forgetgate_fxu、inputgate_fxu、outputgate_fxu

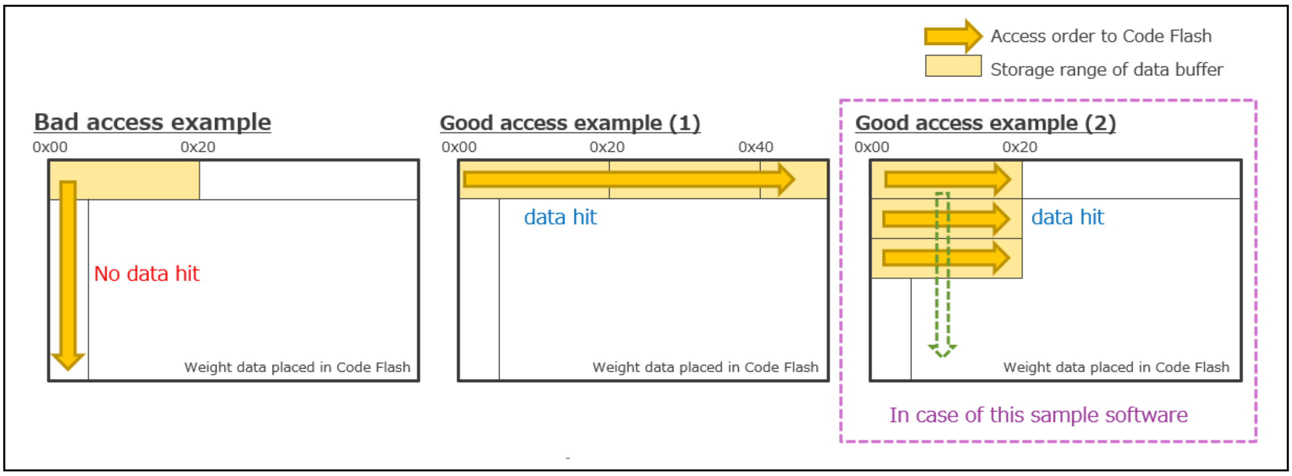
Constant Data Placement to Code Flash

This section describes how to read the constant data from Code Flash in this sample software and how to allocate the constant data in Code Flash.

Effective Use of Data Buffer

Describes the optimization method when the data reading of the Code Flash. When reading the data allocated in Code Flash, if the data is not placed continuously in the memory, the data hit get bad, and it takes long time to read the data, resulting in lower processing performance. Therefore, this sample software performs the data reading by the configuration as shown in Figure 3-3. Thereby, it is possible to read data while making effective use of the data buffer. The next section, "3.3.2 Transpose of Weight Matrix Data" describes the details.

Figure 3-3 Access Order Optimization to Code Flash



Transpose of Weight Matrix Data

As shown in Figure 3-4, allocate the weight matrix data with the rows and columns transposed.

The FXU instruction processes four elements at a time. Therefore, when calculating the product of the matrix and the vector in the general data processing direction (matrix column direction), it is necessary to sum the four elements of vector register in order to calculate one element of the output value. In this sample software, as shown in Figure 3-4, transposes the rows and columns of the matrix data, and the product sum of multiple output values is performed in the parallel. Thereby, it is no longer necessary to sum the four elements of the vector register, and faster processing speed can be expected.

In this sample software, the data processing is performed with the same configuration even in the case of FPU. Therefore, transpose and allocate the weight matrix data regardless of the whether you use FPU or FXU.

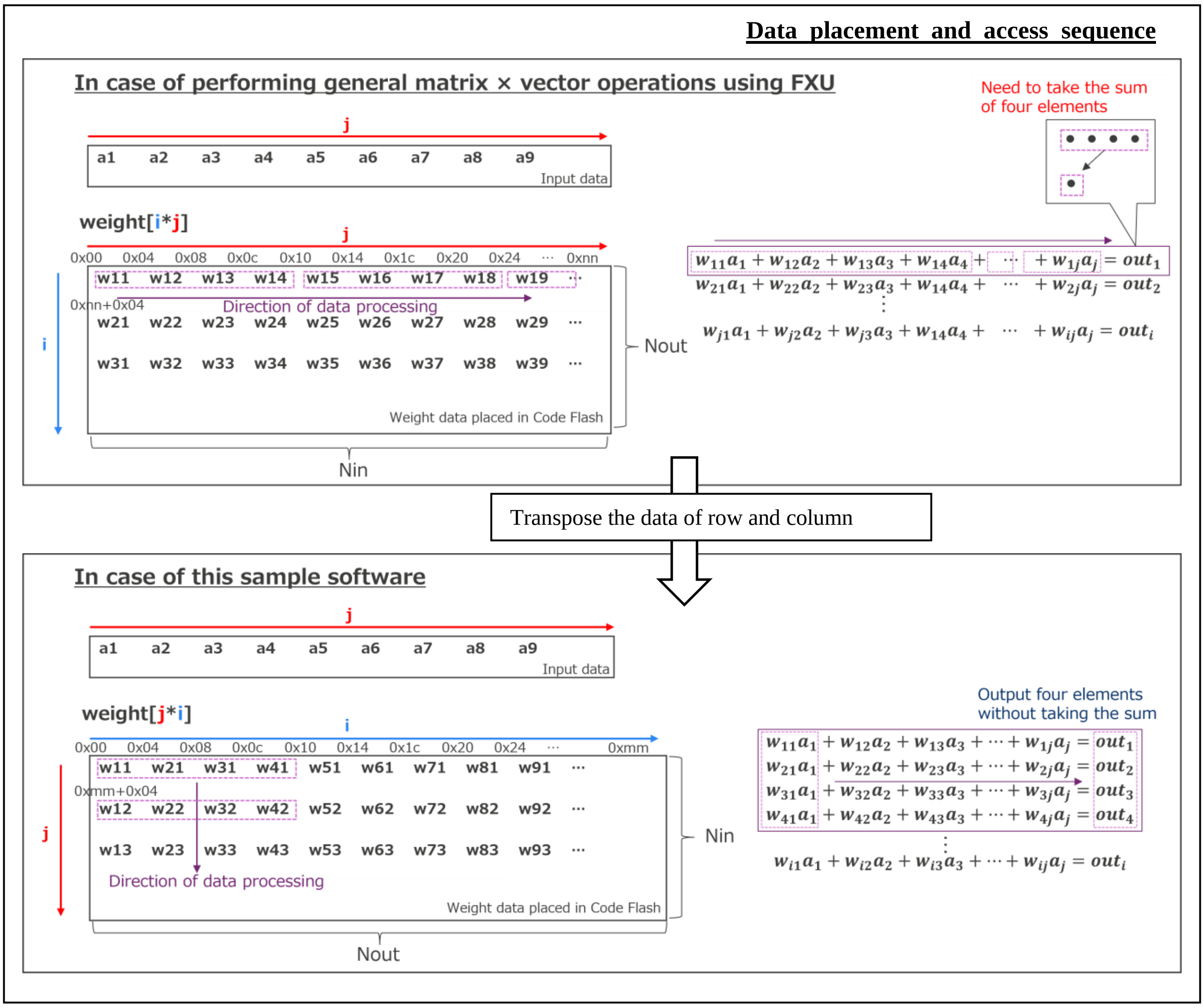


Figure 3-4 Placement Optimization of Weight Matrix Data

【Note】The above diagram is for illustration purposes. Actually, loop unrolling is performed to speed up the processing.

Notes on FXU use

FXU Built-in Functions

Setting when Compiling

To use the FXU built-in functions, it is necessary to enable the FXU support and include the header file v800_fxu.h or the header file v800_ghs.h. The latter automatically includes the former when the FXU support is enabled.

- FXU Support Enabling：-rh850_fxu
- Header File Including：#include <v800_fxu.h> or <v800_ghs.h>

Details of FXU Built-in Function

Table 3-1 shows the built-in function of FXU using in this sample software.

Table 3-1 FXU Built-in Function List

Built-in Function Name	Overview
__ev128_ldvqw	Loads the quad word to the vector register.
__ev128_ldvw_mask	Loads the word to the specified element of the vector register.
__ev128_stvqw	Stores the quad word of the vector register to the specified address.
__ev128_addfs_4	Performs single-precision floating-point addition for each element of the vector register.
__ev128_subfs_4	Performs single-precision floating-point subtraction for each element of the vector register.
__ev128_mulfs_4	Performs single-precision floating-point multiplication for each element of the vector register.
__ev128_divfs_4	Performs single-precision floating-point division for each element of the vector register.
__ev128_fmafs_4	Performs single-precision floating-point fused multiply-addition for each element of the vector register.
__ev128_negfs_4	Returns the negation of the floating-point to each element of the vector register.
__ev128_get_f32	Extracts the element of the specified vector register.

In the FXU built-in function, vector data type "__ev128_f32__" is used. It represents the vector with four 32-bit single-precision floating-point elements.

Table 3-2 to 3-11 show the specification of FXU built-in function using in this operation example.

Table 3-2 Specification of __ev128_ldvqw Function

__ev128_ldvqw			
Overview	Loads the quad word of the vector register. This instruction reads the quad word at the address specified in ptr and stores the value to the result vector register.		
Declaration	`__ev128_f32__ __ev128_ldvqw(void *ptr);`		
Argument	[IN]	void *ptr	: Specifies the start address of the quad word loads to the vector register.
Return value	Value of __ev128_f32__ type		: Returns the vector containing the quad words.
Remarks			

Table 3-3 Specification of __ev128_ldvw_mask Function

__ev128_ldvw_mask			

Overview	Loads/updates the word to the specified element of the vector register. This instruction reads the word at the address specified in ptr, and returns the vector whose elements are combined from the word and elements in vector register, according to the 4-bit immediate values in mask, as following:
	`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]`		

Declaration	`__ev128_f32__ __ev128_ldvw_mask(ghs_c_int__ mask, void *ptr, __ev128_f32__ x);`		
Argument	[IN]	__ghs_c_int__ mask	: Specifies the element of the vector register to update.
	[IN]	void *ptr	: Specifies the address of the word to load.
	[IN]	__ev128_f32__ x	: Specifies the vector register to load/update.
Return value	Value of __ev128_f32__ type		: Returns the vector containing the quad words.
Remarks			

Table 3-4 Specification of __ev128_stvqw Function

__ev128_stvqw			
Overview	Stores the quad word of the vector register to the specified address by ptr.		
Declaration	`void __ev128_stvqw(__ev128_f32__ x, void *ptr);`		
Argument	[IN]	__ev128_f32__ x	: Specifies the vector register to read.
	[IN]	void *ptr	: Specifies the address to store the read quad word.
Return value			
Remarks			

Table 3-5 Specification of __ev128_addfs_4 Function

__ev128_addfs_4			
Overview	Performs the single-precision floating-point addition. It is executed to the four elements of the vector register specified as the argument, and the result is stored to the vector register.		
Declaration	`__ev128_f32__ __ev128_addfs_4(__ev128_f32__ x, __ev128_f32__ y);`		
Argument	[IN]	__ev128_f32__ x	: Specifies the vector register that is added.
	[IN]	__ev128_f32__ y	: Specifies the vector register that is added.
Return value	Value of __ev128_f32__ type		: Returns the vector containing the addition result.
Remarks			

Table 3-6 Specification of __ev128_subfs_4 Function

__ev128_subfs_4			
Overview	Performs the single-precision floating-point subtraction. It is executed to the four elements of the vector register specified as the argument, and the result is stored to the vector register.		
Declaration	`__ev128_f32__ __ev128_subfs_4(__ev128_f32__ x, __ev128_f32__ y);`		
Argument	[IN]	__ev128_f32__ x	: Specifies the vector register that is subtrahend.
	[IN]	__ev128_f32__ y	: Specifies the vector register that is minuend.
Return value	Value of __ev128_f32__ type		: Returns the vector containing the subtraction result.
Remarks			

Table 3-7 Specification of __ev128_mulfs_4 Function

__ev128_mulfs_4			
Overview	Performs the single-precision floating-point multiplication. It is executed to the four elements of the vector register specified as the argument, and the result is stored to the vector register.		
Declaration	`__ev128_f32__ __ev128_mulfs_4(__ev128_f32__ x, __ev128_f32__ y);`		
Argument	[IN]	__ev128_f32__ x	: Specifies the vector register that is multiplied.
	[IN]	__ev128_f32__ y	: Specifies the vector register that is multiplied.
Return value	Value of __ev128_f32__ type		: Returns the vector containing the multiplication result.
Remarks			

Table 3-8 Specification of __ev128_divfs_4 Function

__ev128_divfs_4			
Overview	Performs the single-precision floating-point division. It is executed to the four elements of the vector register specified as the argument, and the result is stored to the vector register.		
Declaration	`__ev128_f32__ __ev128_divfs_4(__ev128_f32__ x, __ev128_f32__ y);`		
Argument	[IN]	__ev128_f32__ x	: Specifies the vector register that is divisor.
	[IN]	__ev128_f32__ y	: Specifies the vector register that is dividend.
Return value	Value of __ev128_f32__ type		: Returns the vector containing the division result.
Remarks			

Table 3-9 Specification of __ev128_fmafs_4 Function

__ev128_fmafs_4			
Overview	Performs the single-precision floating-point fused multiply-addition. It is executed to the four elements of the vector register specified as the argument, and the result is stored to the vector register.		
Declaration	`__ev128_f32__ __ev128_fmafs_4(__ev128_f32__ x, __ev128_f32__ y, __ev128_f32__ z);`		
Argument	[IN]	__ev128_f32__ x	: Specifies the vector register that is multiplied.
	[IN]	__ev128_f32__ y	: Specifies the vector register that is multiplied.
	[IN]	__ev128_f32__ z	: Specifies the vector register that is added.
Return value	Value of __ev128_f32__ type		: Returns the vector register containing the fused multiply-add result.
Remarks			

Table 3-10 Specification of __ev128_negfs_4 Function

__ev128_negfs_4			
Overview	Returns the negation of the floating-point. It is executed to the four elements of the vector register specified as the argument, and the result is stored to the vector register.		
Declaration	`float __ev128_negfs_4 (__ev128_f32__ x);`		
Argument	[IN]	__ev128_f32__ x	: Specifies the vector register that is returned the negation.
Return value	Value of __ev128_f32__ type		: Returns the vector register that is stored the result of the negation returning.
Remarks			

Table 3-11 Specification of __ev128_get_f32 Function

__ev128_get_f32			
Overview	Extracts the element specified by eid in vector register and returns it as a 32-bit single-precision floating-point data.		
Declaration	`float __ev128_get_f32(__ev128_f32__ x,int eid);`		
Argument	[IN]	__ev128_f32__ x	: Specifies the vector register that is extracted.
	[IN]	int eid	: Specifies the elements of the vector register that is extracted.
Return value	Value of float type		: Returns the contained 32bit single precision floating-point data.
Remarks			

2.5.1 Data Size

The FXU instruction processes the four elements at a time. Therefore, If the weight matrix data column size and bias data size are not multiples of four, you need to expand to multiples of four by zero padding.

Figure 3-5 shows the example of zero padding in the operation that adds a bias to the product of a matrix and a vector for the size of pattern A (number of input elements: 10, number of output elements: 10).

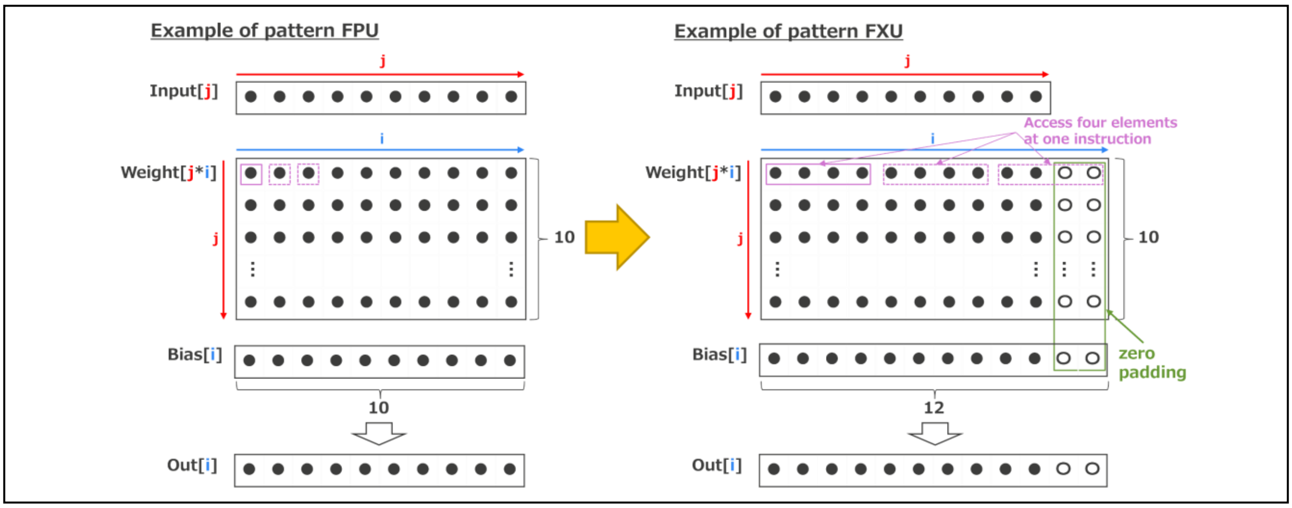


Figure 3-5 Zero padding of Weight Matrix Data and Bias Data

Also, the macro constants corresponding to the weight matrix data column size and the bias data array size must be the multiples of 4 for the data size changes. At the same time, the size of the temporally variable that is stored the calculation result must be the multiples of four. In this sample software, all of these are defined by XX_OUTUNIT. The setting example of this sample software is shown below.

Table 3-12 Macro Constant Setting when Using FXU

Pattern		Input Data	LSTM Layer 1 Data			LSTM Layer 2 Data		
		INPUT_UNIT	LSTM1_INUNIT	LSTM1_HIDDENUNIT	LSTM1_OUTUNIT	LSTM2_INUNIT	LSTM2_HIDDENUNIT	LSTM2_OUTUNIT
			Row size of weight matrix data W	Row size of weight matrix data R	Column size of weight matrix data W/R	Row size of weight matrix data W	Row size of weight matrix data R	Column size of weight matrix data W/R
FXU	A	10	10	10	12	10	10	12
	B	10	10	30	32	30	30	32
	C	10	10	50	52	50	50	52

Alignment Specification

The data that becomes the source and the destination of the instruction for FXU must be aligned properly. If not, the misaligned error will occur. Table 3-13 shows the proper data alignment conditions.

Table 3-13 Data Align Condition

Execution Instruction		Execution Instruction		
FXU-Specific Instruction	Instruction FXU-only	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

The following is the example of allocating data on the 128bit (16bite) boundary by GHS compiler.

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

Performance Comparison of FPU and FXU

Measures the processing time of this sample software when FPU or FXU is used and compares them.

Measurement Condition

In this measurement example, the processing time is measure by the following conditions.

- OS timer is used for the measurement of processing time.

（１） Compiler Condition

- Using GHS Compiler v2021.1.5
- Option：-cpu=rh850g4mh -sda=all -large_sda -Ospeed -Onounroll -rh850_fxu -fastmath -prepare_dispose -no_callt

（２） Evaluation Environment

- Integrated Development Environment：GHS MULTI
- Emulator：E2 emulator
- Evaluation Board：RH850/U2B-468BGA PiggyBack board (Y-RH850-U2B-468PIN-PB-T1-V1)
- MCU：RH850/U2B10-FCC (R7F702Z21EDBG)

2.6 Measurement Result

Table 4-1 shows the processing time when using FPU and FXU. Although, Figure 4-1 shows the graph plotted with the horizontal axis as the number of the Fused Multiply-add). In this measurement, to check the dependency between the number of Fused Multiply-add operations and the processing time, the measurement of the pattern (D) with the changed number of units is also added.

Table 4-1 Processing Time Measurement Result

Pattern	Number of units			Number of processing executions		Processing time [us]	
	Input layer	LSTM layer 1	LSTM layer 2	FMA	exp	FPU	FXU
A	10	10	10	1640	100	**29.9**	**30.0**
D	10	20	20	5680	200	**81.2**	**48.1**
B	10	30	30	12120	300	**148.6**	**84.0**
C	10	50	50	32200	500	**303.1**	**183.9**

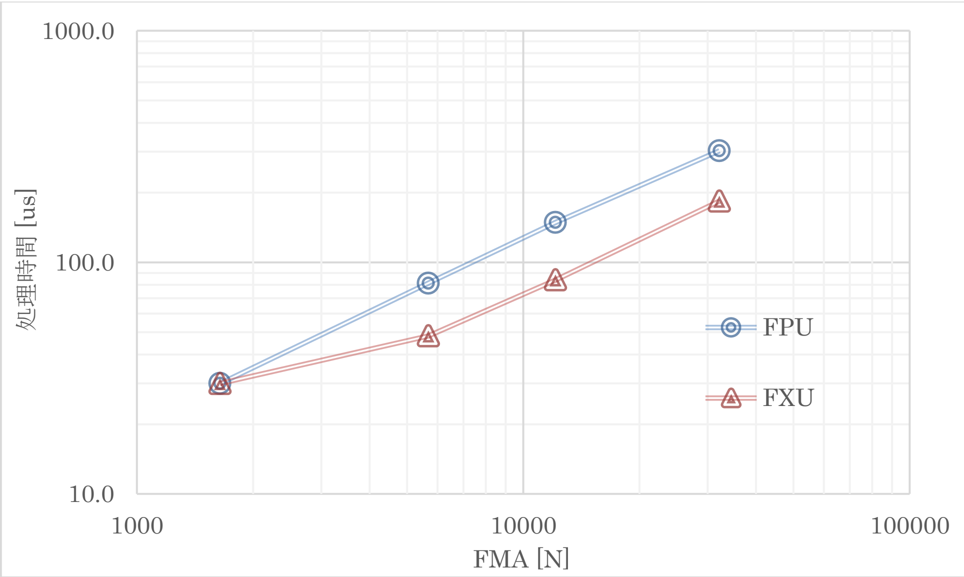


Figure 4-1 Processing Time Measurement Result Graph

Appendix

CPU Configuration of RH850/U2Bx Series

Table 5-1 shows the CPU configuration of RH850/U2Bx.

Please note that the placement of the FXU-equipped CPU is different for each product.

Table 5-1 CPU Configuration of RH850/U2Bx

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	-

【Note】DCLS：Dual Core Lockstep Core
SNGL：Single Core

Note 1. FXU is only in FCC device.

Revision History

Rev.	Date	Description	
		Page	Summary
1.00	July 16, 2024	-	New Release

General Precautions in the Handling of Microprocessing Unit and Microcontroller Unit Products

The following usage notes are applicable to all Microprocessing unit and Microcontroller unit products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Precaution against Electrostatic Discharge (ESD)

   A strong electrical field, when exposed to a CMOS device, can cause destruction of the gate oxide and ultimately degrade the device operation. Steps must be taken to stop the generation of static electricity as much as possible, and quickly dissipate it when it occurs. Environmental control must be adequate. When it is dry, a humidifier should be used. This is recommended to avoid using insulators that can easily build up static electricity. Semiconductor devices must be stored and transported in an anti-static container, static shielding bag or conductive material. All test and measurement tools including work benches and floors must be grounded. The operator must also be grounded using a wrist strap. Semiconductor devices must not be touched with bare hands. Similar precautions must be taken for printed circuit boards with mounted semiconductor devices.

2. Processing at power-on

   The state of the product is undefined at the time when power is supplied. The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the time when power is supplied. In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the time when power is supplied until the reset process is completed. In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the time when power is supplied until the power reaches the level at which resetting is specified.

3. Input of signal during power-off state

   Do not input signals or an I/O pull-up power supply while the device is powered off. The current injection that results from input of such a signal or I/O pull-up power supply may cause malfunction and the abnormal current that passes in the device at this time may cause degradation of internal elements. Follow the guideline for input signal during power-off state as described in your product documentation.

4. Handling of unused pins

   Handle unused pins in accordance with the directions given under handling of unused pins in the manual. The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of the LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible.

5. Clock signals

   After applying a reset, only release the reset line after the operating clock signal becomes stable. When switching the clock signal during program execution, wait until the target clock signal is stabilized. When the clock signal is generated with an external resonator or from an external oscillator during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Additionally, when switching to a clock signal produced with an external resonator or by an external oscillator while program execution is in progress, wait until the target clock signal is stable.

6. Voltage application waveform at input pin

   Waveform distortion due to input noise or a reflected wave may cause malfunction. If the input of the CMOS device stays in the area between V_{IL} (Max.) and V_{IH} (Min.) due to noise, for example, the device may malfunction. Take care to prevent chattering noise from entering the device when the input level is fixed, and also in the transition period when the input level passes through the area between V_{IL} (Max.) and V_{IH} (Min.).

7. Prohibition of access to reserved addresses

   Access to reserved addresses is prohibited. The reserved addresses are provided for possible future expansion of functions. Do not access these addresses as the correct operation of the LSI is not guaranteed.

8. Differences between products

   Before changing from one product to another, for example to a product with a different part number, confirm that the change will not lead to problems. The characteristics of a microprocessing unit or microcontroller unit products in the same group but having a different part number might differ in terms of internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

Notice

1. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation or any other use of the circuits, software, and information in the design of your product or system. Renesas Electronics disclaims any and all liability for any losses and damages incurred by you or third parties arising from the use of these circuits, software, or information.
2. Renesas Electronics hereby expressly disclaims any warranties against and liability for infringement or any other claims involving patents, copyrights, or other intellectual property rights of third parties, by or arising from the use of Renesas Electronics products or technical information described in this document, including but not limited to, the product data, drawings, charts, programs, algorithms, and application examples.
3. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
4. You shall be responsible for determining what licenses are required from any third parties, and obtaining such licenses for the lawful import, export, manufacture, sales, utilization, distribution or other disposal of any products incorporating Renesas Electronics products, if required.
5. You shall not alter, modify, copy, or reverse engineer any Renesas Electronics product, whether in whole or in part. Renesas Electronics disclaims any and all liability for any losses or damages incurred by you or third parties arising from such alteration, modification, copying or reverse engineering.
6. Renesas Electronics products are classified according to the following two quality grades: "Standard" and "High Quality". The intended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below.
   "Standard": Computers; office equipment; communications equipment; test and measurement equipment; audio and visual equipment; home electronic appliances; machine tools; personal electronic equipment; industrial robots; etc.
   "High Quality": Transportation equipment (automobiles, trains, ships, etc.); traffic control (traffic lights); large-scale communication equipment; key financial terminal systems; safety control equipment; etc.
   Unless expressly designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not intended or authorized for use in products or systems that may pose a direct threat to human life or bodily injury (artificial life support devices or systems; surgical implantations; etc.), or may cause serious property damage (space system; undersea repeaters; nuclear power control systems; aircraft control systems; key plant systems; military equipment; etc.). Renesas Electronics disclaims any and all liability for any damages or losses incurred by you or any third parties arising from the use of any Renesas Electronics product that is inconsistent with any Renesas Electronics data sheet, user's manual or other Renesas Electronics document.
7. No semiconductor product is absolutely secure. Notwithstanding any security measures or features that may be implemented in Renesas Electronics hardware or software products, Renesas Electronics shall have absolutely no liability arising out of any vulnerability or security breach, including but not limited to any unauthorized access to or use of a Renesas Electronics product or a system that uses a Renesas Electronics product. RENESAS ELECTRONICS DOES NOT WARRANT OR GUARANTEE THAT RENESAS ELECTRONICS PRODUCTS, OR ANY SYSTEMS CREATED USING RENESAS ELECTRONICS PRODUCTS WILL BE INVULNERABLE OR FREE FROM CORRUPTION, ATTACK, VIRUSES, INTERFERENCE, HACKING, DATA LOSS OR THEFT, OR OTHER SECURITY INTRUSION ("Vulnerability Issues"). RENESAS ELECTRONICS DISCLAIMS ANY AND ALL RESPONSIBILITY OR LIABILITY ARISING FROM OR RELATED TO ANY VULNERABILITY ISSUES. FURTHERMORE, TO THE EXTENT PERMITTED BY APPLICABLE LAW, RENESAS ELECTRONICS DISCLAIMS ANY AND ALL WARRANTIES, EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT AND ANY RELATED OR ACCOMPANYING SOFTWARE OR HARDWARE, INCLUDING BUT NOT LIMITED TO THE IMPLIED WARRANTIES OF MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE.
8. When using Renesas Electronics products, refer to the latest product information (data sheets, user's manuals, application notes, "General Notes for Handling and Using Semiconductor Devices" in the reliability handbook, etc.), and ensure that usage conditions are within the ranges specified by Renesas Electronics with respect to maximum ratings, operating power supply voltage range, heat dissipation characteristics, installation, etc. Renesas Electronics disclaims any and all liability for any malfunctions, failure or accident arising out of the use of Renesas Electronics products outside of such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of Renesas Electronics products, semiconductor products have specific characteristics, such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Unless designated as a high reliability product or a product for harsh environments in a Renesas Electronics data sheet or other Renesas Electronics document, Renesas Electronics products are not subject to radiation resistance design. You are responsible for implementing safety measures to guard against the possibility of bodily injury, injury or damage caused by fire, and/or danger to the public in the event of a failure or malfunction of Renesas Electronics products, such as safety design for hardware and software, including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult and impractical, you are responsible for evaluating the safety of the final products or systems manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. You are responsible for carefully and sufficiently investigating applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive, and using Renesas Electronics products in compliance with all these applicable laws and regulations. Renesas Electronics disclaims any and all liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. Renesas Electronics products and technologies shall not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations. You shall comply with any applicable export control laws and regulations promulgated and administered by the governments of any countries asserting jurisdiction over the parties or transactions.
12. It is the responsibility of the buyer or distributor of Renesas Electronics products, or any other party who distributes, disposes of, or otherwise sells or transfers the product to a third party, to notify such third party in advance of the contents and conditions set forth in this document.
13. This document shall not be reprinted, reproduced or duplicated in any form, in whole or in part, without prior written consent of Renesas Electronics.
14. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products.

(Note1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its directly or indirectly controlled subsidiaries.
(Note2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

(Rev.5.0-1 October 2020)

Corporate Headquarters

TOYOSU FORESIA, 3-2-24 Toyosu,
Koto-ku, Tokyo 135-0061, Japan
www.renesas.com

Trademarks

Renesas and the Renesas logo are trademarks of Renesas Electronics Corporation. All trademarks and registered trademarks are the property of their respective owners.

Contact information

For further information on a product, technology, the most up-to-date version of a document, or your nearest sales office, please visit: www.renesas.com/contact/.