SigImp
Specification

Imp Core

Revision 1.0

14 July 2025

Copyright 2018 ArchiTek＆Safflow-Techno All Rights Reserved

English　ver.

Confidential and Proprietary

Table of Contents

1. Overview

1.1. Introduction

- This embedded core calculates the inverse of a square matrix of floating-point data in memory using a pivot selection + Gauss–Jordan elimination algorithm.

- Supported data formats:
  - 16-bit floating point (IEEE754)
  - 32-bit floating point (IEEE754)

- The number of data points in the matrix can be configured for square matrices ranging from 2×2 to 16×16, including odd sizes. However, each row must be stored in contiguous memory, while columns can use address offsets to jump to arbitrary locations.
  Output data is also written in the same row-major format; columns follow the same structure via offsets.

- Because of the algorithm's nature, values close to infinity may be generated during computation.
  If an infinite value is detected, an error bit will be output to a designated memory location.
  ※ This case is rare due to pivot selection logic.

- This IP represents:
  - Rows as the X-axis
  - Columns as the Y-axis
  - Repetitions as the Z-axis

  Repeat control can be performed on the Z-axis via PSS.

- Once this IP is started, it does not stop until the operation is completed and the inverse matrix is output.

- Note:
  - The memory interface must be customized to your system.
  - This IP can also be used without memory I/F as a standalone converter.

1.2 Main Parameters

Item	Description
Memory Bus	– Imp Data Read: 64-bit × 1
	– Imp Data Write: 64-bit × 1
	– Command List Read: 64-bit × 1
Supported Format	16-bit IEEE754 Float、32-bit IEEE754 Float
Supported Size	Up to 16×16 square matrix
Clock	Undefined (depends on implementation)

1.3 Implementation Parameters

Parameter Name	Description	Default Value
—	(No implementation parameters)	—

2. Signal Lines

2.1. Control Bus Interface

Signal Name	IO	Pol	Source	Description
cntlReq	I	+	clk	• Request signal • Evaluate cntlGnt
cntlGnt	O	+	clk	• Grant signal
cntlRxw	I	+	clk	• R/W signal • Evaluate cntlReq & cntlGnt 0: Write 1: Read
cntlAddr[31:0]	I	+	clk	• Address signal • Evaluate cntlReq & cntlGnt
cntlWrAck	O	+	clk	• Writ acknowledge signal
cntlWrData[31:0]	I	+	clk	• Write data signal • Evaluate cntlWrAck
cntlRdAck	O	+	clk	• Read acknowledge signal
cntlRdData[31:0]	O	+	clk	• Read data signal • Sync cntlRdAck
cntlIrq	O	+	clk	• Interrupt signal • Level hold type

2.2. PSS Interface

Signal Name	IO	Pol	Source	Description
iVld	I	+	clk	• Pipeline start valid signal
iStall	O	+	clk	• Pipeline start stall signal
iAddr[31:4]	I	+	clk	• Address to fetch context data • Evaluate iVld & !iStall
iDelta[15:0]	I	+	clk	• Transfer volume • Evaluate iVld & !iStall
iIndex[64:0]	I	+	clk	• Five coodinates to specify the processing • Evaluate iVld & !iStall
oVld	O	+	clk	• Pipeline end valid signal
oStall	I	+	clk	• Pipeline end stall signal

2.3. Memory Interface (Data Read Use)

Signal Name	IO	Pol	Source	Description
miReq	O	+	clk	• Request signal
miGnt	I	+	clk	• Grant signal
miAddr[31:0]	O	+	clk	• Address signal
miStrb	O	+	clk	• Read strobe signal
miAck	I	+	clk	• Read acknowledge signal
miData[63:0]	I	+	clk	• Read data signal

2.4. Memory Interface (Data Write Use)

Signal Name	IO	Pol	Source	Description
moReq	O	+	clk	• Request signal
moGnt	I	+	clk	• Grant signal
moAddr[31:0]	O	+	clk	• Address signal
moStrb	O	+	clk	• Write strobe signal
moAck	I	+	clk	• Write acknowledge signal
moData[63:0]	O	+	clk	• Write data signal

2.5. Memory Interface (Parameter Read Use)

Signal Name	IO	Pol	Source	Description
meReq	O	+	clk	• Request signal
meGnt	I	+	clk	• Grant signal
meAddr[31:0]	O	+	clk	• Address signal
meStrb	O	+	clk	• Read strobe signal
meAck	I	+	clk	• Read acknowledge signal
meData[63:0]	I	+	clk	• Read data signal
meFlush	O	+	clk	• Last Data signal

2.6. Utility

Signal Name	IO	Pol	Source	Description
rstReq	O	+	clk	• Internal reset signal to reset the external system
rstAck	I	+	clk	• Acknowledge of rstReq
fReq	I	+	clk	• 1 clock early request against the iVld signal • Use to generate gate signal (for ***pss***)
pReq	O	+	clk	• 1 clock early request against the all memory access signal • Use to generate gate signal (for memory)
gate[x:0]	O	+	clk	• Gated clock control signal signifying condition of each internal block
gclk[x:0]	I	+	clk	• Gated clock
Clk	I	+	clk	• Clock
reset_n	I	-	-	• Asynchronous reset signal

3. Architecture and Operation Overview

3.1. System Architecture

- The Pipeline Slice Scheduler (***PSS***) fetches the necessary context data from memory and generates coordinate and control information to start the ***SigImp*** core.
  For more details on ***PSS***, please refer to the ***PSS*** specification.
  The interface is simple, so using ***PSS*** is not mandatory—you may replace it with your own control logic if desired.

- ***SigImp*** operates as a pipeline as shown in Figure 1 (Block Diagram).
  An Initiator retrieves parameters from the command list in memory and manages overall control.
  Data is repeatedly processed by transferring between memory and banked SRAM and by performing operations in the processing unit.

- In ***SigImp***, the iDelta signal is not used..The relevant input signals are:iIndex、iAddr and iVld.

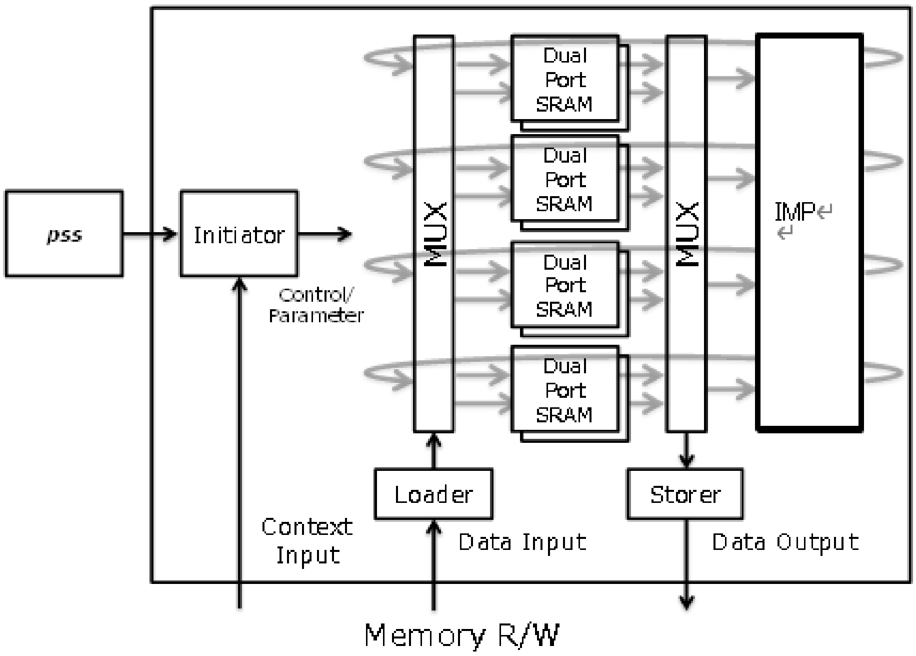


Figure 1 *SigImp* Block Diagram

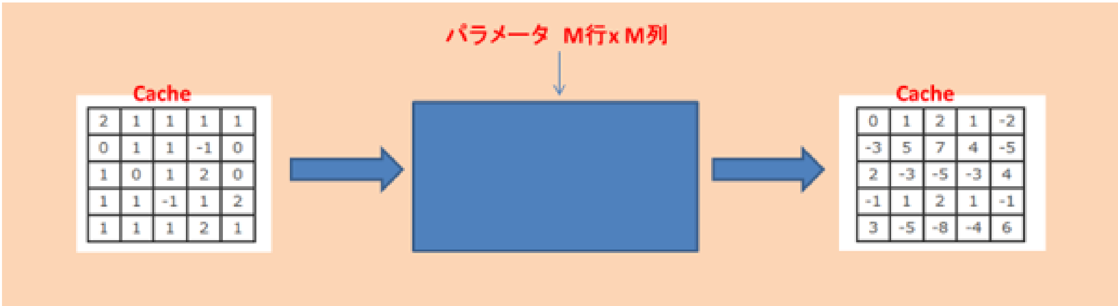


Figure 2 *Imp* Function Diagram

- The circuit block diagram is as follows

  The matrix data retrieved from memory (chache) is inversed using the Divider and MultiSub arithmetic units, and the result of the operation is written back to memory (chache).

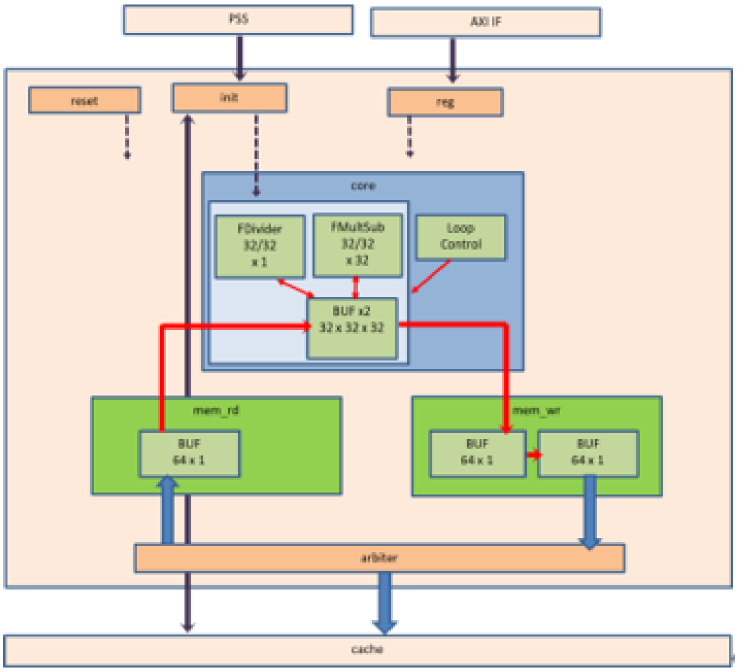


Figure 2 *Imp* Function Block Diagram

階層構造

				備考
SigImp				TOPモジュール
	gpReset			リセット生成ブロック
	sigImp_reg			レジスタブロック
	sigImp_init			PSS IF/コマンドロードブロック
		gpFIFO_2		gpFIFO_2
		gpSig_2		gpSig_2
		sigImp_initBlocker		コマンドブロック
		sigImp_initLoader		コマンドロード
	sigImp_ctrl			Imp制御部
	sigImp_mem_rd			メモリ読出し、FIFO書き込み
		sigImp_sram_fifo		FIFO書き込み読出し
			sram	FIFO(F/F構成)
	sigimp_core			逆行列演算ブロック
		sigImp_fdiv		除算器
		sigImp_fmul		乗算器
		sigImp_fsub		減算器
	sigimp_mem_wr			FIFO読出し、メモリ書き込み
		sigImp_sram_fifo		FIFO書き込み読出し
			sram	FIFO(F/F構成)

Figure 2 *Imp* Function Block Construction

3.2. Operation Overview (Processing Flow)

- The ***PSS*** scans destination coordinates along arbitrary axes and sends the result to the Initiator. Settings for ***PSS*** (like processing units) are preloaded into memory. ***PSS*** manages up to 256 configurations (depending on implementation), and schedules ***SigImp*** execution in time-division manner.

- ***SigImp*** uses:
  - iIndex (Z-axis, i.e. repetitions),
  - ADDR, and
  - VLD signals.

- You can repeat inverse matrix computations with the same matrix size multiple times in a single launch. If using iIndexY for control (configurable via command), set the input to (number of repetitions - 1). Setting it to 0 performs a single execution. Read/write addresses for each repetition change based on the Address Offset set in the Cntl Command.

- The Initiator:

- Reads context info from ***PSS***,
- Performs pipeline setup.

Context parameters are double-buffered, ensuring no performance degradation unless the ***PSS*** specifies extremely short operations.

➢ When ・ Initiator starts, mem_read begins transferring data from memory. mem_read converts the data from the specified format to the internal comparison format and then stores the data in internal memory.

➢ After transferring the mem_read data, pivot selection is performed in the arithmetic block (comparison and reordering). After the pivot selection process, inverse matrix operations are performed in the sweep method. This is done for each column, and the final result is calculated.

➢ After the operation is finished, memory_wr starts to transfer data to memory. memory_wr stores data (address information and data) in memory after converting to the original format. *Address information can be selected from the command.

3.3. Input/Output Format

➢ ***SigImp*** supports the following floating-point formats for input and output:

- 32-bit single-precision floating point (IEEE 754)
- 16-bit half-precision floating point (IEEE 754)

※ the internal arithmetic unit uses a 32-bit arithmetic unit.

Half-precision floating-point numbers are converted once to single-precision floating-point numbers, and then converted back to half-precision floating-point numbers after the operation is completed.

➢ Basically, mappings in memory are stored in order from the LSB direction. To swap the order, manipulate the Byte Swap and Word Swap parameters in the memory operation.

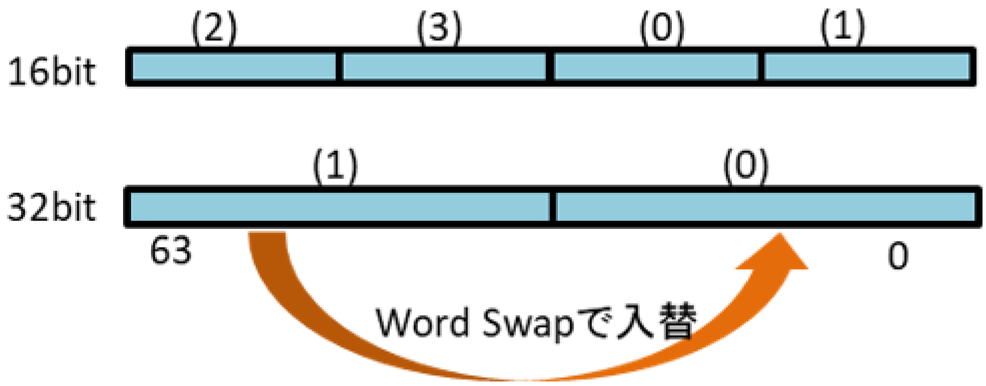


Figure 5 Bit Mapping

➢ The following is an example of memory placement for the following matrix array. When crossing rows, address jumps can be made at address offset Y (column).

◆行列配置
(32bit x 2) or (16bit x 4) = 64bit = 1word
連続した配置を期待 →

列

0-0	0-1	0-2	0-3	0-4	0-5	0-6	0-7	0-8	..	0-14	0-15
1-0	1-1	1-2	1-3						..		
2-0	2-1								..		
3-0	3-1								..		
4-0									..		
5-0									..		
6-0									..		
7-0									..		
8-0									..		
9-0									..		
10-0									..		
11-0									..		
12-0									..		
13-0									..		13-15
14-0	14-1								..	14-14	14-15
15-0	15-1	15-2	15-3						..	15-14	15-15

行

◆メモリ配置

16bit float

address	63			0
0	0-3	0-2	0-1	0-0
8	0-7	0-6	0-5	0-4
16	0-11	0-10	0-9	0-8
24	0-15	0-14	0-13	0-12
32	1-3	1-2	1-1	1-0
40	1-7	1-6	1-5	1-4
48	1-11	1-10	1-9	1-8
56	1-15	1-14	1-13	1-12

offset Y(列)

32bit float

address	63	0
0	0-1	0-0
8	0-3	0-2
16	0-5	0-4
24	0-7	0-6
32	0-9	0-8
40	0-11	0-10
48	0-13	0-12
56	0-15	0-14

3.4. arithmetic section

- ***SigImp*** uses a pivot selection and sweep method to perform iterative operations and output the final inverse matrix. The algorithm of the arithmetic part is described below in C code.

```
#include <stdio.h>
#include <math.h>

int main(){
   double in_mat[5][5]={{1,2,0,-1,3},{-1,1,2,0,4},{2,0,1,1,1},{1,-2,-1,1,-1},{0,1,1,-1,2}}; //入力用の配列
   double inv_a[5][5]; //ここに逆行列が入る
   double mat_a[5][5]; //ここに逆行列が入る
   double inv_a[5][5]; //ここに逆行列が入る
   double mat_c[5][5]; //ここに逆行列が入る
   double buf; //一時的なデータを蓄える
   double tmp;
   int i,j,k; //カウンタ
   int DEGN=5;  //配列の次数

   //初期化
   for(i=0; i<DEGN; i++){
      for(j=0;j<DEGN;j++){
         mat_a[i][j]= in_mat[i][j];
      }
   }

   //単位行列を作る
   for(i=0; i<DEGN; i++){
      for(j=0;j<DEGN;j++){
         inv_a[i][j]= (i==j) ? 1.0 : 0.0;
      }
   }
```

Figure 1 C code（1）

```
//掃き出し法
for(i=0; i<DEGN; i++){
  //ピボット選択
  //最大値検出
  int max = i;
  for( j=i+1; j<DEGN; j++){
    if( fabs(mat_a[j][i]) > fabs(mat_a[max][i]) ){
      max = j;
    }
  }
  printf("switch i=%d, %f <=> max=%d, %f¥n", i, mat_a[i][i], max, mat_a[max][i]);
  // 行の入れ替え
  if( max != i ){
    for( k=0; k<DEGN; k++ ){
      // 入力行列側
      tmp = mat_a[max][k];
      mat_a[max][k] = mat_a[i][k];
      mat_a[i][k] = tmp;
      // 単位行列側
      tmp = inv_a[max][k];
      inv_a[max][k] = inv_a[i][k];
      inv_a[i][k] = tmp;
    }
  }

  //
  buf = 1/mat_a[i][i];
  for(j=0; j<DEGN; j++){
    mat_a[i][j]*= buf;
    inv_a[i][j]*= buf;
    printf("mat=%f  inv=%f buf=%f (%d,%d) =(%d,%d)/(%d,%d)¥n", mat_a[i][j], inv_a[i][j], buf, i,j,i,j,i,i);
  }
  for(j=0; j<DEGN; j++){
    if(i != j){
      buf = mat_a[j][i];
      for(k=0; k<DEGN; k++){
        mat_a[j][k]-= mat_a[i][k]*buf;
        //printf("%f¥n",inv_a[j][k]);
        inv_a[j][k]-= inv_a[i][k]*buf;
        printf("mat=%f  inv=%f buf=%f (%d,%d) -=(%d,%d)*(%d,%d)¥n",mat_a[j][k], inv_a[j][k], buf,  j,k,i,k,j,i);
      }
    }
  }
}

//逆行列を出力
printf("------- Anser ----------¥n");
for(i=0; i<DEGN; i++){
  for(j=0; j<DEGN; j++){
    printf(" %f", inv_a[i][j]);
  }
  printf("¥n");
}
```

Figure 2 C code（2）

3.5. connection with pss

- The command list is retrieved from memory based on the address output by the ***PSS*** (iAddr). For details on the command list, please refer to the Command List Description. If ***PSS*** does not exist, access the ***PSS*** interface directly.

- Normally, data R/W is performed while the address is incremented in memory word units.

3.6. Performance

- From startup to termination, the added value of the cycle overhead required for memory access is required, but the arithmetic operations are about 380 cycles for an 8x8 square matrix and about 820 cycles for a 16x16 matrix.

Matrix Size	Compute Cycles (Imp Block)	Total Instruction Count (Approx.)
8×8	~380 cycles	~4,608 instructions
16×16	~820 cycles	~34,560 instructions
32×32	~3,268 cycles	~268,800 instructions

4. Register Description

4.1. Overview

- All registers are accessed via the Control Bus.

- Be cautious when setting certain registers, as some may affect pipeline operation or performance depending on timing.

- The following symbols indicate access types:

Symbol	Meaning
R	Read Only (write has no effect)
R/W	Read / Write
R/WC	Read / Write Clear

- Do not access reserved registers, and for reserved fields, always write '0'.

- In address and data fields, 'x' indicates a Don't Care value.

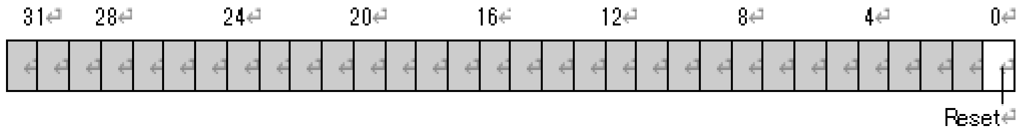
4.2. Definition (Register List)

Address	**Register Name**	**Description**
0x0000_0000	Reset	Reset control
0x0000_0004	System	System control

4.3. Details

4.3.1.1. Reset Register

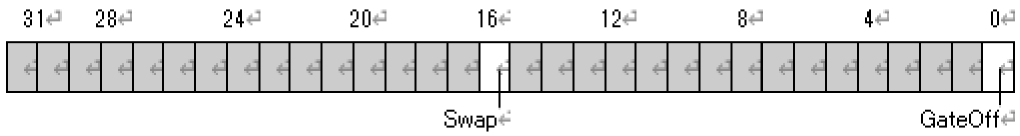
[Address: 0x0000_0000]



Name	Type	Default	Description
Reset	R/W	0	Synchronous reset. After setting '1', it is necessary to clear '0'. Unlike the reset_n signal, the contents of the register are retained. After setting '1', immediately assert the rstReq signal. This signal notifies the external system that ***SigImp*** is in reset state and requests a response. Once the response is complete, the rstAck signal must be asserted (if no response is required, always assert '1'). After these procedures are complete, the Reset automatically returns to '0'.

4.3.1.2. System Register

[Address: 0x0000_0004]



Name	Type	Default	Description
Swap	R/W	0	Enables Word Swap. If set to 1, swaps the upper and lower 32-bit halves of the 64-bit bus. (Swaps within a 32-bit word are configured via the command list.)
GateOff	R/W	0	Gate clock off mode. When set to 1, all bits in the gate signal are forced to '1', disabling gated clocks.

5. Command List Description

5.1. Overview

- The starting address of the command list is specified by the address output from the ***PSS***. After ***SigImp*** is activated, it retrieves the command list from memory and stores it in internal registers.

- Each stage of the pipeline manages its own command list independently, so different stages can operate with different commands concurrently during pipeline execution. Therefore, synchronization commands are not required.

- For reserved registers or fields, always set them to '0'.

- All addresses listed here are relative to the base address provided by ***PSS***.

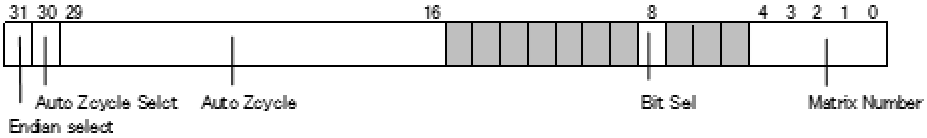
5.2. Definition (Command List Table)

Address	Command Name	Description
00	Cntl	Control command
04	Data Source Base Address	Source base address for reading data
08	Data Distination Base Address	Base address for writing output data
0C	Source Address Offset Y	Offset per column during read
10	Source Address Offset Z	Offset per repetition during read
14	Destination Address Offset Y	Offset per column during write
18	Destination Address Offset Z	Offset per repetition during write
1C	Error Distination Address	Memory address for writing error results

5.3. Details

5.3.1.1. Cntl Command

○ Addres:0x00
制御コマンド



Name	Size	Description
Matrix Number	4	行列演算数N−1を設定 0:禁止 1〜15:2〜16正方行列
Bit Sel	1	32bit/16bit選択 0 : 32bit floatを使用する 1 : 16bit floatを使用する
Auto Zcycle	14	自動で連続的に繰り返す(Zcycle)回数−1を設定　ex) 16回繰り返し⇒15
Auto Zcycle Select	1	0:シングル動作(非連続動作)　　1:本IP内で連続的にZcycleを行う
Endian select	1	16bit選択時有効　32bit内で、上位16bitと下位16bitを入替

5.3.1.2. Data Source Base Address Command

○ Addres:0x04
データ読出し元アドレス

Name	Size	Description
Data Source Base Address	32	入力行列データが格納してあるメモリの先頭アドレスを指定 ※[31:3]がアドレス有効ビットです。[1:0]は、バスのコマンドビットとなります。

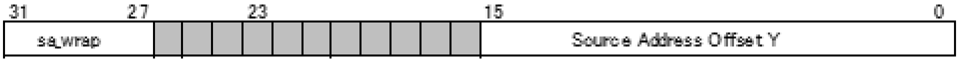
5.3.1.3. Data Distination Base Address Command

○ Addres:0x08
データ出力先アドレス

Name	Size	Description
Data Destination Base Address	32	逆行列出力データを格納するメモリの先頭アドレスを指定 ※[31:3]がアドレス有効ビットです。[1:0]は、バスのコマンドビットとなります。

5.3.1.4. Source Address Offset Y(Column) Command

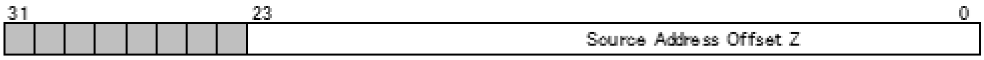
○ Addres:0x0C
読出しアドレス オフセットY



Name	Size	Description
Source Address Offset Y	24	データを読み出すメモリのアドレスオフセットを指定
sa_wrap	5	入力行列データの格納アドレスをマスクします。 アドレスへのマスク値は0x0_ffff_ffff >> wrap マスクされたアドレスビットは、ベースアドレスの値を保持します。 例えば、wrapの値が8の場合は、アドレスの上位8bitはBase Addressの上位8bitのまま保存され、残り下位24bitだけが変化いたします。

5.3.1.5. Source Address Offset Z(Loop) Command

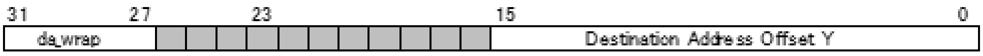
○ Addres:0x10
読出アドレス オフセットZ



Name	Size	Description
Source Address Offset Z	24	データを読み出すメモリのアドレスオフセットを指定

5.3.1.6. Destination Address Offset Y(Column) Command

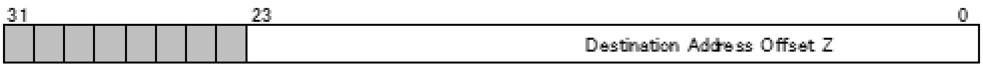
○ Addres:0x14
出力先アドレス オフセットY



Name	Size	Description
Destination Address Offset Y	24	データを書き込むメモリのアドレスオフセットを指定
da_wrap	5	逆行列出力データの格納アドレスをマスクします。 アドレスへのマスク値は0x0_ffff_ffff >> wrap マスクされたアドレスビットは、ベースアドレスの値を保持します。 例えば、wrapの値が8の場合は、アドレスの上位8bitはBase Addressの上位8bitのまま保存され、残り下位24bitだけが変化いたします。

5.3.1.7. Destination Address Offset Z(Loop) Command

○ Addres:0x18
出力先アドレス オフセットZ



Name	Size	Description
Destination Address Offset Z	24	データを書き込むメモリのアドレスオフセットを指定

5.3.1.8. Error Destination Address Command

○ Addres:0x1C
エラー出力先アドレス

31	23	0
Error Distination Address		

Name	Size	Description
Error Destination Address	32	エラーを格納するメモリの先頭アドレスを指定 ※[31:3]がアドレス有効ビットです。[1:0]は、バスのコマンドビットとなります。

6. Application Notes

6.1. Additional Information

6.1.1. SRAM Usage

- No SRAM to use.

  All temporary buffers are in FF configuration.