SigSort

Specification

Sort Core for Chichibu

Revision 1.3

14 July 2025

English ver.

Table of Contents (English Translation)

1. Overview

1.1. Introduction

- ***SigSort*** is an embedded core that sorts memory-resident data in ascending order using a merge sort algorithm. For descending order, data should be read in reverse (reverse address order).

- It supports the following data formats:
  - 8-bit (Fixed-point)
  - 16-bit (IEEE754 Binary16, Unsigned Short, Signed Short)
  - 32-bit (IEEE754 Float, Unsigned Integer, Signed Integer)

- Before merge sorting, a 16-input pipeline sorting network performs a preliminary sort.

- It uses a 4+1 input merge sort, selecting the smallest and second smallest from 4 blocks of data per cycle.

- The number of sort elements can range from 1 to 1G (2^{30}), including odd values, as long as they are stored consecutively in memory.

- Sorting is executed in multiple stages:
  - The smallest S that satisfies (64 * 4S-1) ≥ (sort data volume N) becomes the number of stages. (The number of stages can also be calculated by rounding up (log4N)-1.)
    For example, if the sort data volume is 16384 words, the number of stages is 5, and the sorted data is finally stored in memory by repeating stages 1 to 5.
    In addition, the sort time is approximately (number of stages S * sort data volume N * 1/2) cycles.You can also store:
    * This is an approximate time for the core, and the time may vary depending on the frequency of the internal clock of this IP and the bandwidth of memory (DDR, etc.) access.
    * Once this IP is started, it will not stop until all stages are completed.The addresses of data elements in sorted order.

- In addition to data, it is possible to store the addresses before sorting in the order after sorting in memory (DDR, etc.).

- It is possible to store the unique data in memory (assuming DDR, etc.) in the order it was sorted after the sort operation (unique mode). It is also possible to store the number of unique data items that existed before the sort operation.

Note1: Memory interfaces (I/Fs) may need customization per system.

Note2: The core can also function as a transformation component without using memory I/F.

1.2. Main Parameters

Memory Bus	Sort Data Read: 64-bit × 1
	Sort Data Write: 64-bit × 1
	Command List Read: 64-bit × 1 |
Throughput	Stage Count (S) × Sort Data Amount (N) × 1/2 cycles
Supported Formats	8-bit (fixed-point)
	16-bit (IEEE754 Binary16, unsigned short, signed short)
	32-bit (IEEE754, unsigned integer, signed integer)
Number of Supported	1,073,741,824(=2^{30}) data
Clock	Undefined (depends on implementation process)

1.3. Implementation Parameters

Parameter Name	Description	Default Value
SRAM_SIZE	• FIFO 用 SRAM word size	64

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[6:0]	O	+	clk	• Gated clock control signal signifying condition of each internal block
gclk[6:0]	I	+	clk	• Gated clock
clk	I	┼	clk	• Clock
reset_n	I	-	-	• Asynchronous reset signal

3. Architecture and Operation Overview

3.1. System Architecture

- A Pipeline Slice Scheduler (***PSS***) retrieves the required context from memory and generates information such as coordinates, which are then used to activate the ***SigSort*** core.
  Refer to the ***PSS*** specification document for details.
  Note: Because the interface is simple, it is not mandatory to use a ***PSS***. You may replace the ***PSS*** with your own custom core.

- The ***SigSort*** core operates as a pipeline, as shown in Figure 1. An Initiator retrieves parameters from the command list in memory and controls the overall process.
  The data is transferred to and from memory, stored in banked SRAM, and processed repeatedly by the merge sort circuit.

- ***SigSort*** does not use the Index input. The iDelta, iAddr, and iVld signals are used as input signals.

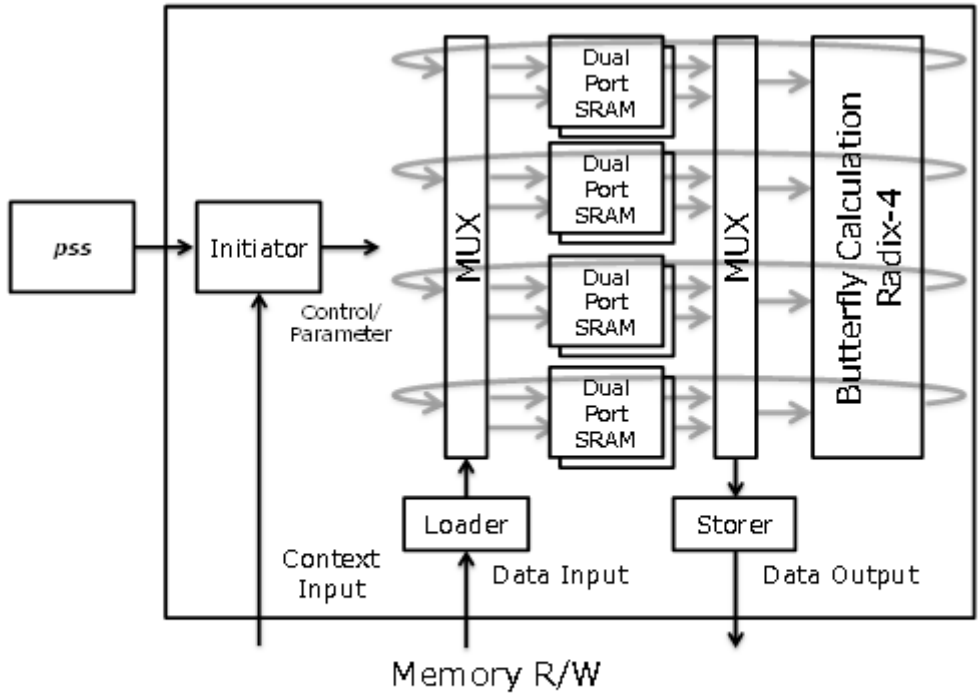


Figure 1 *SigSort* Block Diagram

➢ The circuit block diagram is as follows

The data stored in SRAM is either stored via a 16-input sorting network dedicated to Stage 1 or directly into 4 BUFFs (buffers), and the minimum and second data are selected by 4+1 input merge sorting via shift registers, and stored again in SRAM. In unique mode, data is further compared and only unique data is stored.

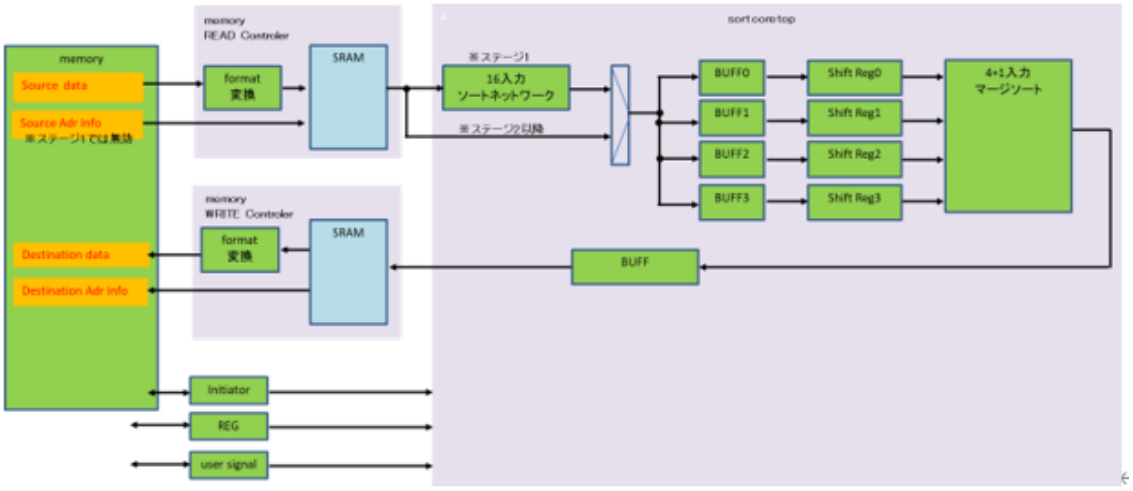


Figure 2 *Sort* Function Block Diagram

階層構造

				備考
SigSort				TOPモジュール
	SigSort_reset			リセット生成ブロック
	SigSort_clock			ゲートクロック生成
	SigSort_reg			frFFT_reg_10　完コピ
	SigSort_init			frFFT_init_10　相当
		gpFIFO_2		gpFIFO_2　相当
		gpSig_2		gpSig_2　相当
		sort_initBlocker		frFFT_initBlocker_10　相当
		sort_initLoader		frFFT_initLoader_10　相当
	SigSort_cntl			sort_initとsort_coreとのブリッジ
	SigSort_mem_rd			メモリ読出し、FIFO書き込み
		SigSort_mem_rd_ch		メモリ読出し、FIFO書き込み（x4ch分）
			SigSort_sram_fifo	FIFO(SRAM)
			sram	
	SigSort_core_top			マージソートブロック
		SigSort_core		マージソートコア
		SigSort_snet		ソートネットワーク
	SigSort_mem_wr			FIFO読出し、メモリ書き込み
		SigSort_sram_fifo		FIFO(SRAM)
			sram	

3.2. Operation Overview (Processing Order)

- The ***PSS*** scans destination coordinates along arbitrary axes and sends results to the Initiator. Settings for the ***PSS*** (e.g., processing unit) are preloaded into memory. ***PSS*** can handle up to 256 context settings (depending on implementation) and schedule them time-divisionally to drive ***SigSort***.

SigSort uses index Y input or DELTA input, ADDR signal, and VLD signal.

It is possible to repeat a sorting process with the same sorting amount multiple times in a single startup. When using a delta signal (switchable by command), enter a value of "number of repetitions − 1." Entering 0 will result in a single processing. Additionally, the sort data read (or written) during repetition will change according to the Address Offset specified in the Cntl Command described later. When using the IndexY signal, the number of repetitions is left to the PSS, and sorting is performed by adding Index Y * Address Offset to the Base Address.

The Initiator reads image context information sent from ***PSS*** and sets up the pipeline. Parameters extracted from the context are managed using double buffering, so performance degradation will not occur unless the processing unit size specified by ***PSS*** is extremely small.

Once the Initiator starts, mem_read begins transferring data from memory to four internal SRAMs. The data is converted from its specified format to an internal comparison format before being stored in SRAM.

Simultaneously with the data transfer of mem_read, a comparison merge (reordering) process is repeatedly performed on the SRAM in the comparison block.

When even one comparison-merge process is completed, memory_wr starts transferring data from SRAM to memory. memory_wr stores data (address information and data) in memory after converting it to the original format. *Address information can be selected from the command.

memory_wr, memory_rd, and the comparison/merge logic work independently.
This enables overlapping of memory operations and computation, resulting in efficient pipelining.
See: Figure 3 − Pipeline Scheduling Diagram

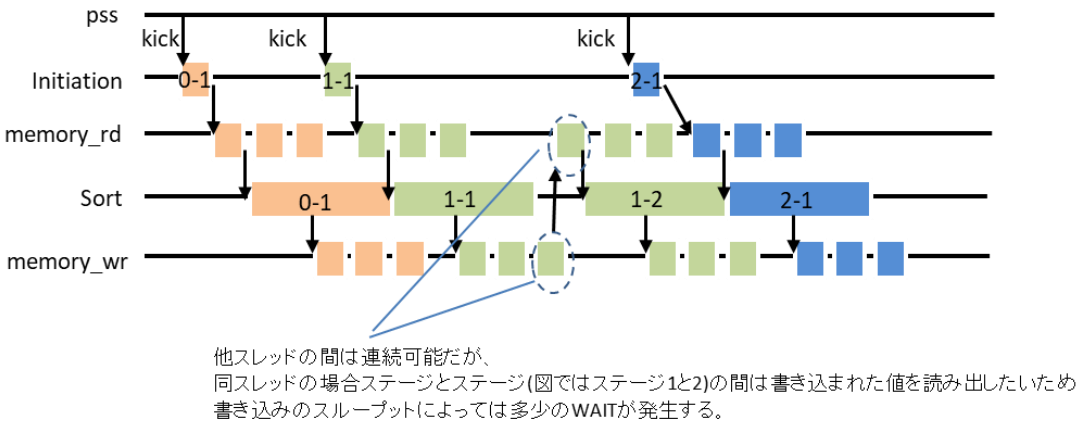


Figure 1 Pipeline scheduling

3.3 Input/Output Format

Data supports the following formats.

This is achieved by a combination of BitFormat[1:0], Sig Format, and ComplementSel in the control command.

Select 8bit(=0)/16bit(=1)/32bit(=3) in BitFormat.

In SigFormat, select with sign bit (=0)/without sign bit (=1).

In Complement Sel, select whether to switch between a mantissa display (=1) or two's complement display (=0).

When SigFormat = 0 (signed) and Complement Sel = 0, the most significant bit is inverted by BitFormat,

When SigFormat = 0 (signed) and Complement Sel = 1, bit inversion is performed when the most significant bit is 1 using BitFormat, and when the most significant bit is 0, the most significant bit is inverted before output to the comparison circuit. The most significant bit is inverted before being output to the comparison circuit.

Format	BitFormat	SigFormat	ComplementSel
8-bit Fixed-point	0x0	0	1
8-bit Unsigned Integer (0-255)	0x0	1	-
16-bit IEEE754 Binary16	0x1	0	1
16-bit Unsigned Short	0x1	1	-

Format	BitFormat	SigFormat	ComplementSel
16-bit Signed Short	0x1	0	0
32-bit IEEE754 Float	0x3	0	1
32-bit Unsigned Integer	0x3	1	–
32-bit Signed Integer	0x3	0	0

In memory, data is generally stored in LSB-first order (little-endian).

If you want to reverse the order, you can use the Byte Swap or Word Swap parameters during memory operations.

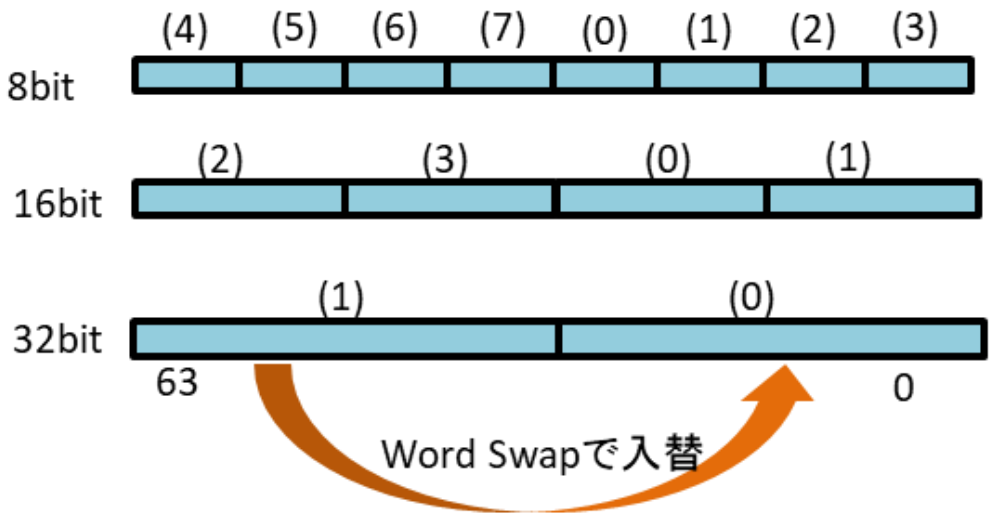


Figure 2 Bit Mapping

3.4 Stages

In ***SigSort***, the final sorted data is produced by repeating the merge sort operation multiple times, depending on the total number of data elements.

The number of stages is automatically calculated from a lookup table based on the data size.

Sub-stages:

Within each stage, there are smaller units called sub-stages, which define the internal sorting granularity (set by internal parameters).

データ量		ステージ数
～	64	1
～	256	2
～	1024	3
～	4096	4
～	16384	5
～	65536	6
～	262144	7
～	1048576	8
～	4194304	9
～	16777216	10
～	67108864	11
～	268435456	12
～	1073741824	13
～	4294967296	14

Figure 3 Stage Mapping

※nはデータ量
※sはサブステージ番号

ステージ	INDEX	サブステージ量(小数切上)
1	64	n/64
2	256	n/256
3	1024	n/1024
4	4096	n/4096
5	16384	n/16384
6	65536	n/65536
7	262144	n/262144
8	1048576	n/1048576
9	4194304	n/4194304
10	16777216	n/16777216
11	67108864	n/67108864
12	268435456	n/268435456
13	1073741824	n/1073741824
14	4294967296	n/4294967296

Figure 4 Sub Stage Mapping

Example) When sorting 4096 pieces of data

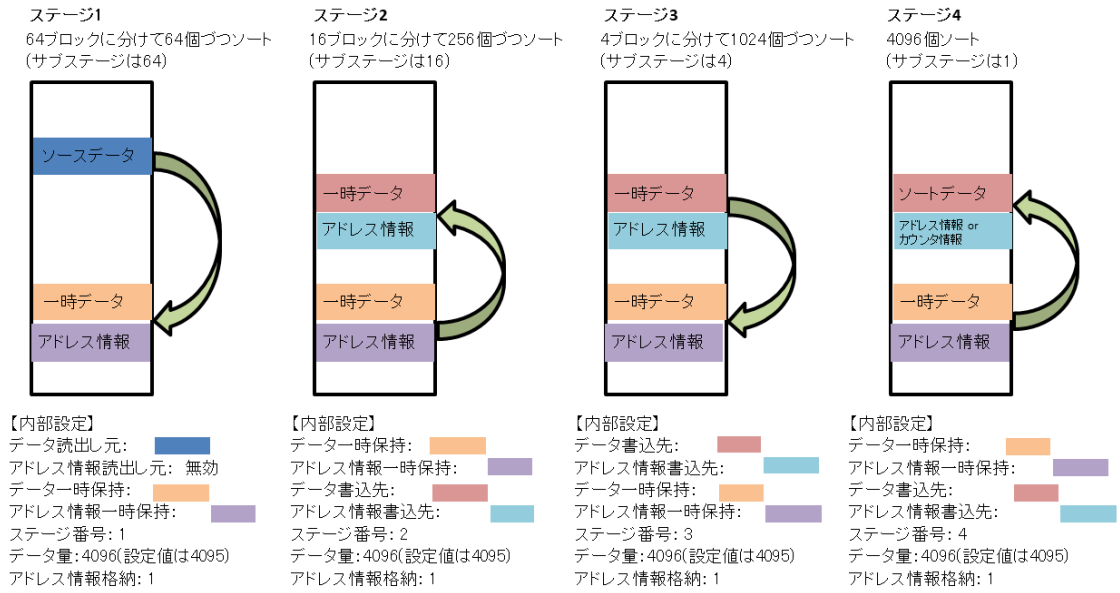


Figure 5 Sort Stage Example

3.5 Core Comparison Block

Stage 1: 16-Input Sorting Network

In Stage 1, a 16-input sorting network is used to perform ascending sort on 16 data elements before they are passed to the downstream 4+1 merge sort.

This 16-sort network is only used in Stage 1. Later stages use a different mechanism.

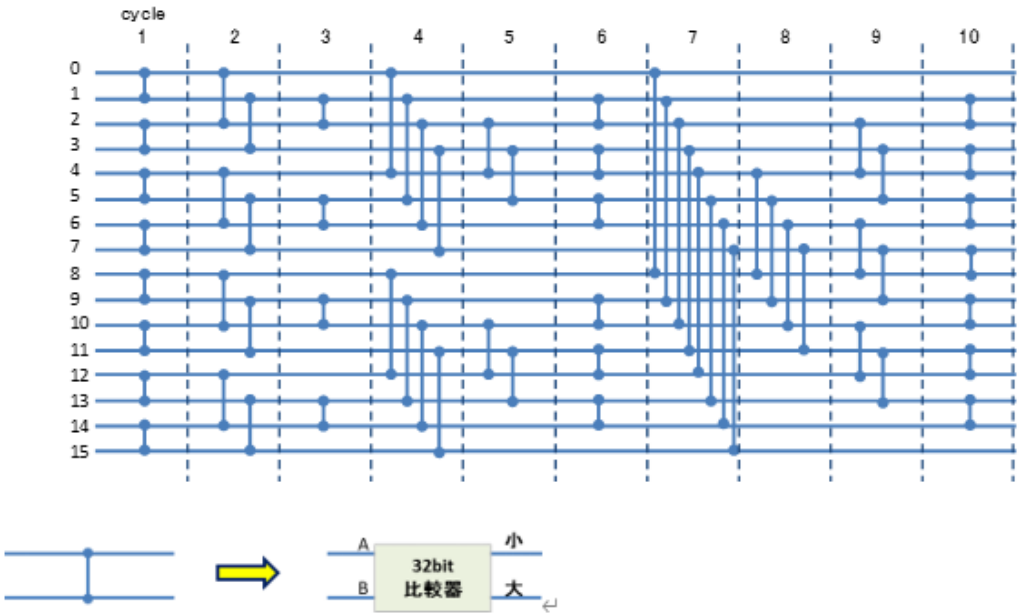


Figure 9 16 Sort Network

入力例	pipeline 1	2	3	4	5	6	7	8	9	10
10	10	1	1	1	1	1	1	1	1	1
13	13	13	10	5	5	2	2	2	2	2
1	1	10	13	7	2	5	5	5	3	3
15	15	15	15	12	10	7	7	7	4	4
5	5	2	2	2	7	10	9	3	5	5
12	12	7	5	10	12	12	10	4	7	6
7	2	5	7	13	13	13	11	6	6	7
2	7	12	12	15	15	15	14	8	8	8
6	3	3	3	3	3	3	3	9	9	9
3	6	6	4	4	4	4	4	10	10	10
14	4	4	6	6	6	6	6	11	10	10
4	14	14	14	11	9	8	8	14	12	11
10	10	8	8	8	8	9	10	10	11	12
11	11	9	9	9	11	10	12	12	14	13
9	8	10	10	10	10	11	13	13	13	14
8	9	11	11	14	14	14	15	15	15	15

Figure 10 16 Sort Network Example

The 4+1 input merge sort compares the values of the 4 input shift registers to find the minimum and second data. Each BUFF and shift register contains the already sorted. The minimum value is detected from the table by comparing the top of the shift registers in a round-robin fashion; in the case of the second data, the top of the shift registers other than the minimum value and the second data in the shift register that contained the minimum value are candidates. For this reason, the second data of the shift register is compared with the first data of each register in a round-robin manner.

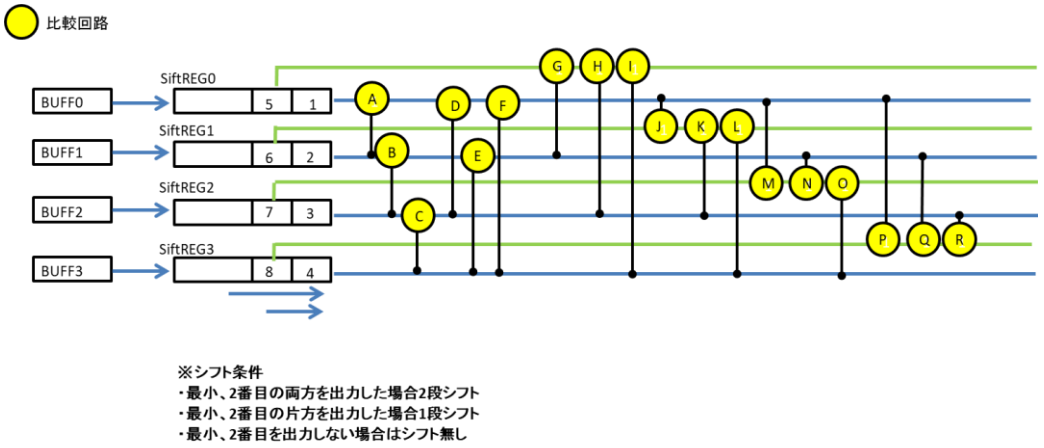
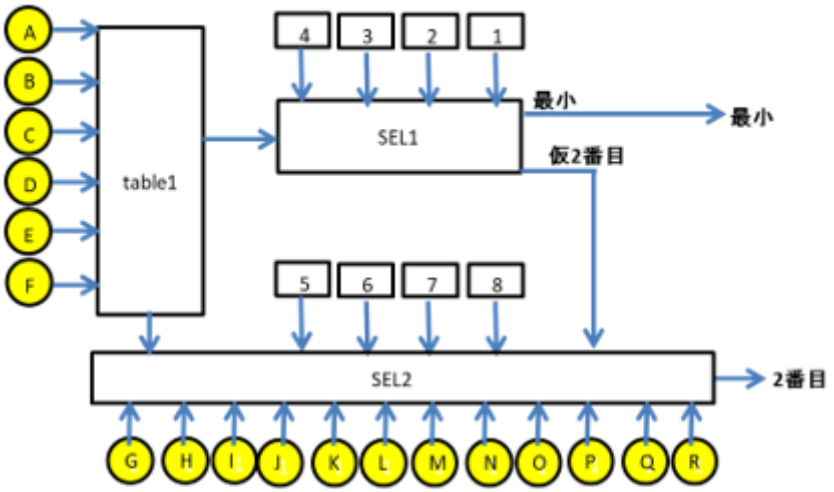


Figure 6 4+1 Merge Sort Diagram



※最小のデータがあったShiftREGの2番目のデータとなる可能性も
あるため、tableから検出した2番目のデータと比較する。

※1cycleで実現するために、予めShiftREGの2番目のデータと、1番目のデータを比較
しておく。

Figure 12 4+1 Merge Sort Function Diagram

	A	B	C	D	E	F
結果	2>1	3>2	4>3	3>1	4>2	4>1
4321	1	1	1	1	1	1
3421	1	1	0	1	1	1
4231	1	0	1	1	1	1
2431	1	0	1	1	0	1
3241	1	1	0	1	0	1
2341	1	0	0	1	0	1
4312	0	1	1	1	1	1
3412	0	1	0	1	1	1
4132	0	1	1	0	1	1
1432	0	1	1	0	1	0
3142	0	1	0	1	1	0
1342	0	1	0	0	1	0
4213	1	0	1	0	1	1
2413	1	0	1	0	0	1
4123	0	0	1	0	1	1
1423	0	0	1	0	1	0
2143	1	0	1	0	0	0
1243	0	0	1	0	0	0
3214	1	1	0	1	0	0
2314	1	0	0	1	0	0
3124	0	1	0	1	0	0
1324	0	1	0	0	0	0
2134	1	0	0	0	0	0
1234	0	0	0	0	0	0
解無し	その他					

※解無しの場合、組合せとしてはあり得ないが、ハード上は4321とする。

Figure 13 4+1 Merge Sort Table

Buffer and Shift Register Format

Each BUFF and shift register entry is 66 bits, structured as:

- 1 bit — Valid/Invalid flag
- 1 bit — Data Present/Absent flag
- 32 bits — Address info
- 32 bits — Data
- Total: 66 bits × 16 words × 4 channels

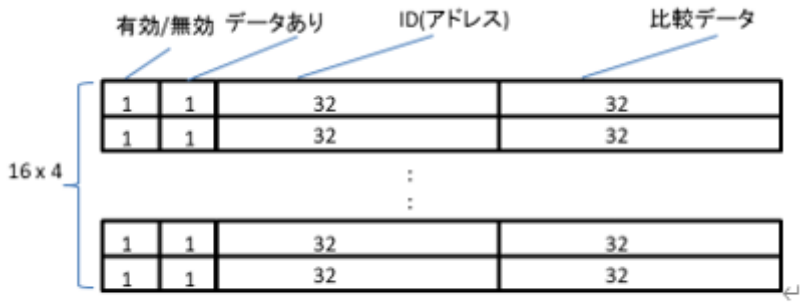


Figure 14 Buffer & Shift Register Bit Assignment

Valid/Invalid Flag:

- Set when data is loaded from SRAM into BUFF.
- If all 8 comparison candidates (4 heads + 4 second entries) are invalid, the system enters WAIT state until new data is loaded.

Data Present/Absent Flag:

- If a BUFF has already finished outputting all its data in a sub-stage but others still have data: → Mark remaining slots as "data absent" to avoid unnecessary comparisons.
- If both compared values are marked absent, the lower BUFF number is considered smaller.

These flags are added when writing from SRAM to BUFF, based on remaining data count in the sub-stage.

If there are fewer than 16 words left, the remaining unused slots are marked as "data absent".

Shift Register Actions:

- 1-stage shift occurs if only the minimum value is found.
- 2-stage shift occurs if both minimum and second smallest are in the same shift register:
  - If two or more data entries exist: shift two steps.
  - If only one entry exists: pop the first value from BUFF and push the second directly.

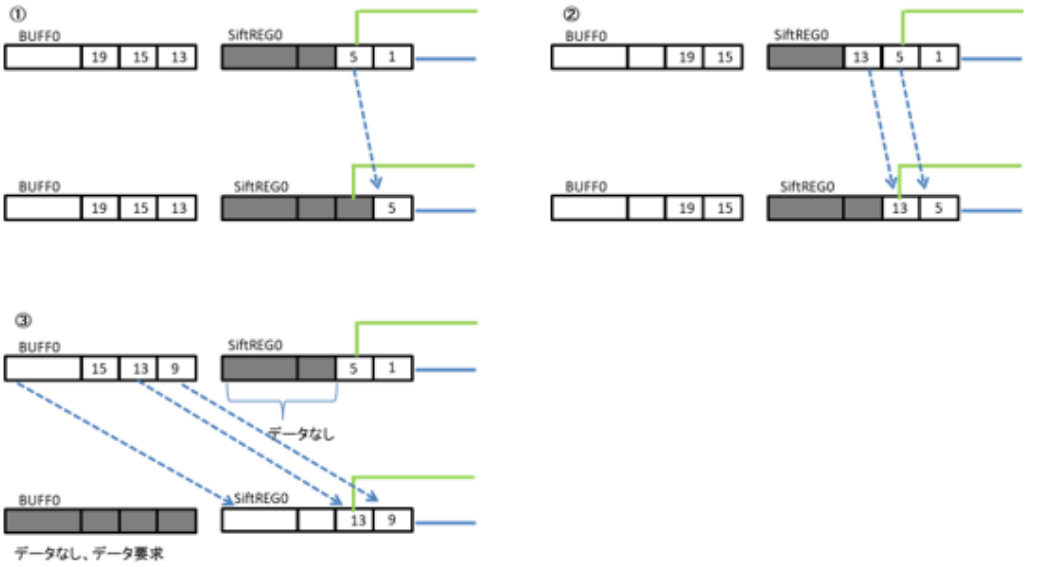


Figure 15 Buffer & Shift Register Action

If there is only one data at the beginning of the shift register, the beginning of BUFF0 is the second evaluated value. If the minimum value and the second are detected in the same shift register, SiftREG is a two-stage shift condition, and when PUSHing data from BUFF to the shift register, data is PUSHed into the shift register with the top of BUFF removed④.

If only the minimum value is detected, the shift register is a one-stage shift condition and simply PUSHs the data from BUFF to the shift register⑤.

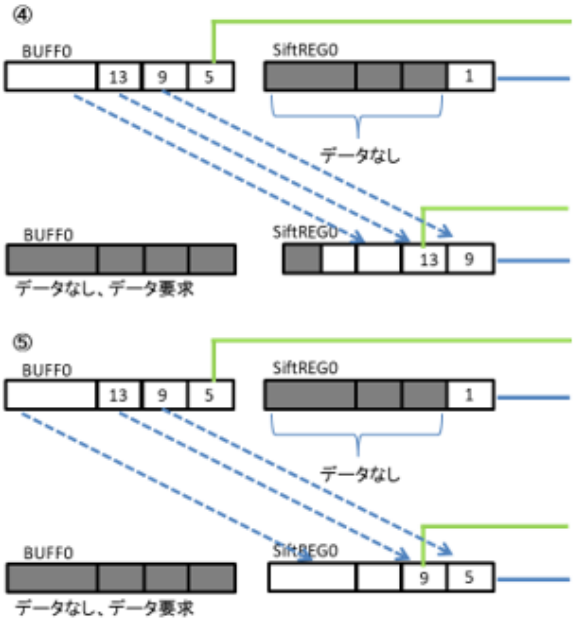


Figure 16 Buffer & Shift Register Special Action

Unique Mode (Final Stage Only)

In Unique Mode, two unique buffers (buff0, buff1) are attached downstream of the shift register in the final stage only.

- These store the previous comparison results.
- If the newly pushed values (dat0 and dat1) match the previous value, the unique counter is incremented.
- If the values are different, the value and its count are written to memory.
-

buff0	buff1	条件	out0 act	out1 act	buff0 status	buff1 status	cnt0 act	cnt1 act	uniq cnt
–	–	最終ステージ以外	dat0	dat1	–	–	–	–	–
empty	empty	last action	–	–	empty	empty	0	0	–
full	empty	last action	buff0, cnt0	–	empty	empty	0	0	–
full	full	last action	buff0, cnt0	buff1, cnt1	empty	empty	0	0	–
empty	empty	dat1 is none	–	–	dat0	–	1	0	–
empty	empty	dat0 ＝ dat1	–	–	dat0	–	2	–	+1
empty	empty	dat0 ≠ dat1	–	–	dat0	dat1	1	1	–
full	empty	buff0＝dat0, dat1 is none	–	–	–	–	+1	0	+1
full	empty	buff0＝dat0, dat0＝dat1	–	–	–	–	+2	–	+2
full	empty	buff0＝dat0, dat0≠dat1	–	–	–	dat1	+1	1	+1
full	empty	buff0≠dat0, dat1 is none	buff0, cnt0	dat0, 1	empty	–	0	0	–
full	empty	buff0≠dat0, dat0＝dat1	–	–	–	dat1	–	2	+1
full	empty	buff0≠dat0, dat0≠dat1	buff0, cnt0	dat0, 1	dat1	empty	1	0	–
full	full	buff1＝dat0, dat1 is none	buff0, cnt0	buff1, (cnt1+1)	empty	empty	0	0	+1
full	full	buff1＝dat0, dat0＝dat1	–	–	–	–	–	+2	+2
full	full	buff1＝dat0, dat0≠dat1	buff0, cnt0	buff1, (cnt1+1)	dat1	empty	1	0	+1
full	full	buff1≠dat0, dat1 is none	buff0, cnt0	buff1, cnt1	dat0	empty	1	0	–
full	full	buff1≠dat0, dat0＝dat1	buff0, cnt0	buff1, cnt1	dat0	empty	2	0	+1
full	full	buff1≠dat0, dat0≠dat1	buff0, cnt0	buff1, cnt1	dat0	dat1	1	1	

Figure 7 Unique mode buffer condition matrix

3.6 SRAM FIFO

SRAM is used as a temporary storage buffer for:

- Reading data from memory, and
- Writing data back to memory.

Read Side

- There are four FIFOs, each implemented using SRAM.
- The SRAM is dual-banked: one bank for sort data, and another for address info.
- Each of these banks has two planes to enable double-buffering, improving throughput by allowing the next stage to start reading while the current stage is still processing merge operations.

FIFO Operation:

- Each of the 4 FIFOs operates independently and manages its own data count.
- Memory read requests are issued per FIFO.
- Data is read using the data source address and address info source address specified in the command list, incremented with each read.

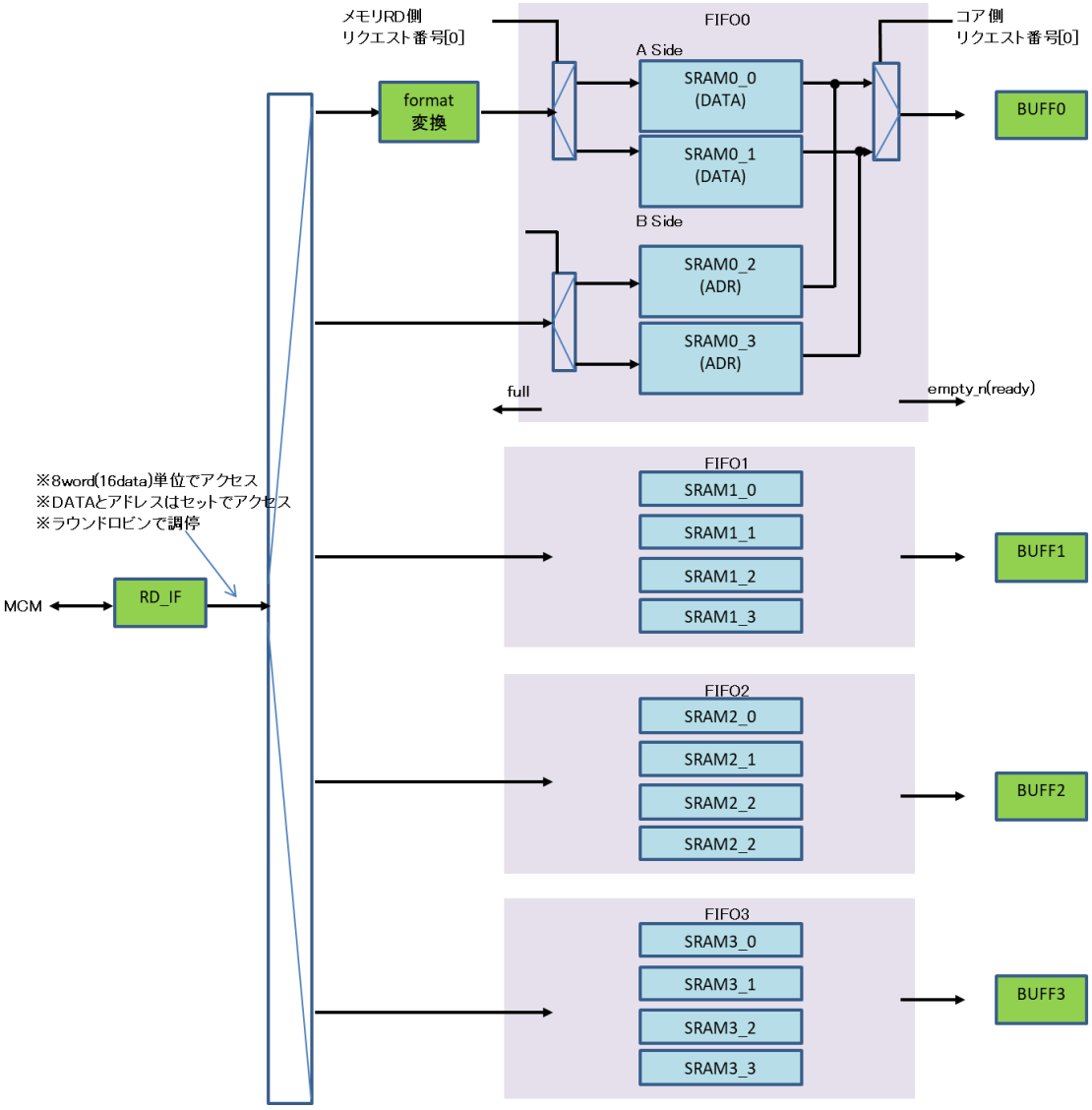


Figure 17: SRAM Write FIFO Diagram

Each of the four FIFOs has an internal arbitration circuit implemented to issue independent memory read requests. The arbitration circuit is implemented in a round-robin fashion.

ラウンドロビン型調停
アクセスを行ったFIFOは、優先度最低へ
その他のFIFO、優先度が高いほうへシフトする。

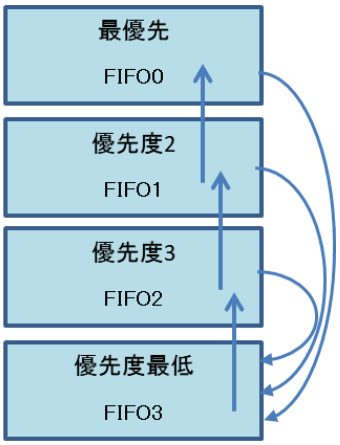


Figure 18: Read Request Arbitration

The following table shows the allocation table of data and address information allocated to the four FIFOs.

Also shown is the addressing table to realize the sorting.

※sはサブステージ番号
※BUFFメモリ0～3は、SRAMに配置されたBUFFです。

	アドレス							
ステージ	オフセット	FIFOメモリ0		FIFOメモリ1		FIFOメモリ2		FIFOメモリ3	
1	64*(s-1)+	0 ～	15	16 ～	31	32 ～	47	48 ～	63
2	256*(s-1)+	0 ～	63	64 ～	127	128 ～	191	192 ～	255
3	1024*(s-1)+	0 ～	255	256 ～	511	512 ～	767	768 ～	1023
4	4096*(s-1)+	0 ～	1023	1024 ～	2047	2048 ～	3071	3072 ～	4095
5	16384*(s-1)+	0 ～	4095	4096 ～	8191	8192 ～	12287	12288 ～	16383
6	65536*(s-1)+	0 ～	16383	16384 ～	32767	32768 ～	49151	49152 ～	65535
7	262144*(s-1)+	0 ～	65535	65536 ～	131071	131072 ～	196607	196608 ～	262143
8	1048576*(s-1)+	0 ～	262143	262144 ～	524287	524288 ～	786431	786432 ～	1048575
9	4194304*(s-1)+	0 ～	1048575	1048576 ～	2097151	2097152 ～	3145727	3145728 ～	4194303
10	16777216*(s-1)+	0 ～	4194300	4194301 ～	8388601	8388602 ～	12582902	12582903 ～	16777203
11	67108864*(s-1)+	0 ～	16777200	16777201 ～	33554401	33554402 ～	50331602	50331603 ～	67108803
12	268435456*(s-1)+	0 ～	67108800	67108801 ～	134217601	134217602 ～	201326402	201326403 ～	268435203
13	1073741824*(s-1)+	0 ～	268435200	268435201 ～	536870401	536870402 ～	805305602	805305603 ～	1073740803
14	4294967296*(s-1)+	0 ～	1073740800	1073740801 ～	2147481601	2147481602 ～	3221222402	3221222403 ～	4294963203

※各BUFFメモリが空になりそうな時には、メモリからデータを取得する。

Figure 8 SRAM Write FIFO Address Assignment

BUFF0

ステージ	残りINDEX数-1		残りINDEX数-1			
1	[31:4] == 0	の時	残りINDEX数[4:0]	others	0x0000_0010	
2	[31:6] == 0	の時	残りINDEX数[6:0]	others	0x0000_0040	
3	[31:8] == 0	の時	残りINDEX数[8:0]	others	0x0000_0100	
4	[31:10] == 0	の時	残りINDEX数[10:0]	others	0x0000_0400	
5	[31:12] == 0	の時	残りINDEX数[12:0]	others	0x0000_1000	
6	[31:14] == 0	の時	残りINDEX数[14:0]	others	0x0000_4000	
7	[31:16] == 0	の時	残りINDEX数[16:0]	others	0x0001_0000	
8	[31:18] == 0	の時	残りINDEX数[18:0]	others	0x0004_0000	
9	[31:20] == 0	の時	残りINDEX数[20:0]	others	0x0010_0000	
10	[31:22] == 0	の時	残りINDEX数[22:0]	others	0x0040_0000	
11	[31:24] == 0	の時	残りINDEX数[24:0]	others	0x0100_0000	
12	[31:26] == 0	の時	残りINDEX数[26:0]	others	0x0400_0000	
13	[31:28] == 0	の時	残りINDEX数[28:0]	others	0x1000_0000	
14	[31:30] == 0	の時	残りINDEX数[30:0]	others	0x4000_0000	

BUFF1

ステージ	残りINDEX数-1			残りINDEX数-1			
1	[31:4] == 1	の時	残りINDEX数[5] ,残りINDEX数[3:0]	[31:4] != 0	0x0000_0010	ohters([3:4]=0)	0x0000_0000
2	[31:6] == 1	の時	残りINDEX数[7] ,残りINDEX数[5:0]	[31:6] != 0	0x0000_0040	ohters	0x0000_0000
3	[31:8] == 1	の時	残りINDEX数[9] ,残りINDEX数[7:0]	[31:8] != 0	0x0000_0100	ohters	0x0000_0000
4	[31:10] == 1	の時	残りINDEX数[11] ,残りINDEX数[9:0]	[31:10] != 0	0x0000_0400	ohters	0x0000_0000
5	[31:12] == 1	の時	残りINDEX数[13] ,残りINDEX数[11:0]	[31:12] != 0	0x0000_1000	ohters	0x0000_0000
6	[31:14] == 1	の時	残りINDEX数[15] ,残りINDEX数[13:0]	[31:14] != 0	0x0000_4000	ohters	0x0000_0000
7	[31:16] == 1	の時	残りINDEX数[17] ,残りINDEX数[15:0]	[31:16] != 0	0x0001_0000	ohters	0x0000_0000
8	[31:18] == 1	の時	残りINDEX数[19] ,残りINDEX数[17:0]	[31:18] != 0	0x0004_0000	ohters	0x0000_0000
9	[31:20] == 1	の時	残りINDEX数[21] ,残りINDEX数[19:0]	[31:20] != 0	0x0010_0000	ohters	0x0000_0000
10	[31:22] == 1	の時	残りINDEX数[23] ,残りINDEX数[21:0]	[31:22] != 0	0x0040_0000	ohters	0x0000_0000
11	[31:24] == 1	の時	残りINDEX数[25] ,残りINDEX数[23:0]	[31:24] != 0	0x0100_0000	ohters	0x0000_0000
12	[31:26] == 1	の時	残りINDEX数[27] ,残りINDEX数[25:0]	[31:26] != 0	0x0400_0000	ohters	0x0000_0000
13	[31:28] == 1	の時	残りINDEX数[29] ,残りINDEX数[27:0]	[31:28] != 0	0x1000_0000	ohters	0x0000_0000
14	[31:30] == 1	の時	残りINDEX数[31] ,残りINDEX数[29:0]	[31:30] != 0	0x4000_0000	ohters	0x0000_0000

※0x10や0x40などの値を作るため上位の値を利用

BUFF2 ※3

ステージ	残りINDEX数-1			残りINDEX数-1			
1	[31:4] == 2	の時	残りINDEX数[4:0]	[31:5] != 0 ([3:4]=2,3)	0x0000_0010	ohters([3:4]=0,1)	0x0000_0000
2	[31:6] == 2	の時	残りINDEX数[6:0]	[31:7] != 0	0x0000_0040	ohters	0x0000_0000
3	[31:8] == 2	の時	残りINDEX数[8:0]	[31:9] != 0	0x0000_0100	ohters	0x0000_0000
4	[31:10] == 2	の時	残りINDEX数[10:0]	[31:11] != 0	0x0000_0400	ohters	0x0000_0000
5	[31:12] == 2	の時	残りINDEX数[12:0]	[31:13] != 0	0x0000_1000	ohters	0x0000_0000
6	[31:14] == 2	の時	残りINDEX数[14:0]	[31:15] != 0	0x0000_4000	ohters	0x0000_0000
7	[31:16] == 2	の時	残りINDEX数[16:0]	[31:17] != 0	0x0001_0000	ohters	0x0000_0000
8	[31:18] == 2	の時	残りINDEX数[18:0]	[31:19] != 0	0x0004_0000	ohters	0x0000_0000
9	[31:20] == 2	の時	残りINDEX数[20:0]	[31:21] != 0	0x0010_0000	ohters	0x0000_0000
10	[31:22] == 2	の時	残りINDEX数[22:0]	[31:23] != 0	0x0040_0000	ohters	0x0000_0000
11	[31:24] == 2	の時	残りINDEX数[24:0]	[31:25] != 0	0x0100_0000	ohters	0x0000_0000
12	[31:26] == 2	の時	残りINDEX数[26:0]	[31:27] != 0	0x0400_0000	ohters	0x0000_0000
13	[31:28] == 2	の時	残りINDEX数[28:0]	[31:29] != 0	0x1000_0000	ohters	0x0000_0000
14	[31:30] == 2	の時	残りINDEX数[30:0]	[31] != 0	0x4000_0000	ohters	0x0000_0000

BUFF3

ステージ	残りINDEX数-1			残りINDEX数-1			
1	[31:4] == 3	の時	残りINDEX数[6] ,残りINDEX数[3:0]	[31:6] != 0	0x0000_0010	ohters([3:4]=0,1,2)	0x0000_0000
2	[31:6] == 3	の時	残りINDEX数[8] ,残りINDEX数[5:0]	[31:8] != 0	0x0000_0040	ohters	0x0000_0000
3	[31:8] == 3	の時	残りINDEX数[10] ,残りINDEX数[7:0]	[31:10] != 0	0x0000_0100	ohters	0x0000_0000
4	[31:10] == 3	の時	残りINDEX数[12] ,残りINDEX数[9:0]	[31:12] != 0	0x0000_0400	ohters	0x0000_0000
5	[31:12] == 3	の時	残りINDEX数[14] ,残りINDEX数[11:0]	[31:14] != 0	0x0000_1000	ohters	0x0000_0000
6	[31:14] == 3	の時	残りINDEX数[16] ,残りINDEX数[13:0]	[31:16] != 0	0x0000_4000	ohters	0x0000_0000
7	[31:16] == 3	の時	残りINDEX数[18] ,残りINDEX数[15:0]	[31:18] != 0	0x0001_0000	ohters	0x0000_0000
8	[31:18] == 3	の時	残りINDEX数[20] ,残りINDEX数[17:0]	[31:20] != 0	0x0004_0000	ohters	0x0000_0000
9	[31:20] == 3	の時	残りINDEX数[22] ,残りINDEX数[19:0]	[31:22] != 0	0x0010_0000	ohters	0x0000_0000
10	[31:22] == 3	の時	残りINDEX数[24] ,残りINDEX数[21:0]	[31:24] != 0	0x0040_0000	ohters	0x0000_0000
11	[31:24] == 3	の時	残りINDEX数[26] ,残りINDEX数[23:0]	[31:26] != 0	0x0100_0000	ohters	0x0000_0000
12	[31:26] == 3	の時	残りINDEX数[28] ,残りINDEX数[25:0]	[31:28] != 0	0x0400_0000	ohters	0x0000_0000
13	[31:28] == 3	の時	残りINDEX数[30] ,残りINDEX数[27:0]	[31:30] != 0	0x1000_0000	ohters	0x0000_0000
14	[31:30] == 3	の時	残りINDEX数[32] ,残りINDEX数[29:0]	others	0x0000_0000		

※0x10や0x40などの値を作るため上位の値を利用

Figure 9 SRAM Write FIFO Address Assignment Detail

Write Side

- Similar to the read side, the write side also has dual-plane SRAM for both data and address info to enable double-buffering.
- This allows the next stage's sorting operation to begin while the current stage is still writing data to memory.
- Memory addressing is done using:

- Data destination address
- Address info destination address

These addresses are incremented linearly during writing.

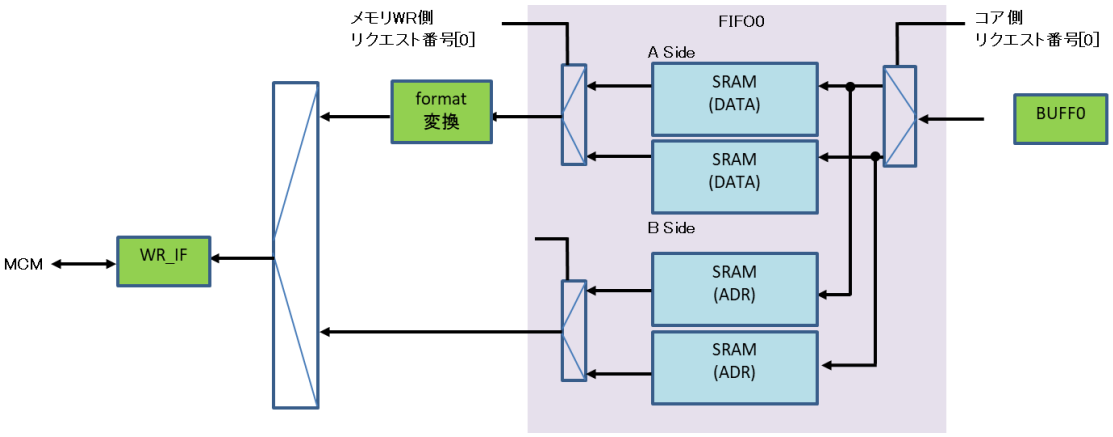


Figure 21: SRAM Read FIFO Diagram

3.7 Connection with *PSS*

- The iAddr signal output by the ***PSS*** is used to fetch the command list from memory.
- For more information on the command list, see the command list section of the specification.
- If a ***PSS*** is not used, you can directly access the ***PSS*** interface signals.

Normally, data is read/written to memory by incrementing the address per word unit.

3.8 Performance

From start to finish, the overhead of cycles required for memory access and the amount of data required for merge sort N/2 cycles must be added. Also multiply by the number of stages until fully sorted. The highest rate of memory access approaches the maximum throughput of (number of stages S * amount of data N/2). That is, 'N/2 * (log4N − 1) * decimal point rounded up.

Since a typical bubble sort takes N2 cycles and a typical merge sort takes N * log2N cycles, this IP has a small sort time advantage over other sorting performance.

Algorithm	Cycle Count Formula	Example (N = 4,096)
Bubble Sort	N^2	16,777,216 cycles
Standard Merge Sort	$N \times \log_2(N)$	49,152 cycles
SigSort (This IP)	$N / 2 \times (\log_4(N) - 1)$	10,240 cycles

Figure 22: Compare Sort Algorithm

4. Register Description

4.1 Overview

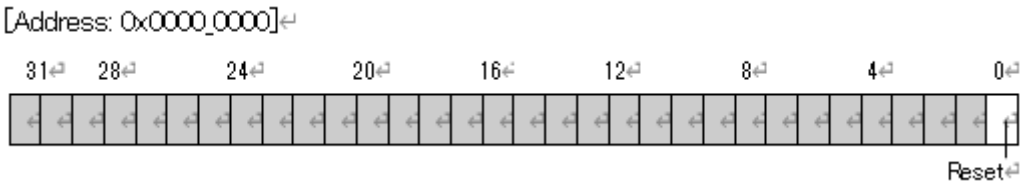
- All registers are accessed via the Control Bus.
- Some registers may affect pipeline behavior and performance, so timing of configuration is important.
- The following symbols are used to describe access types:
    - R: Read-only (write has no effect)
    - R/W: Read/Write
    - R/WC: Read/Write Clear
- Do not access reserved registers.
  For reserved fields, always write '0'.
- In register and data tables, 'x' means Don't Care.

4.2 Definition (Register List)

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

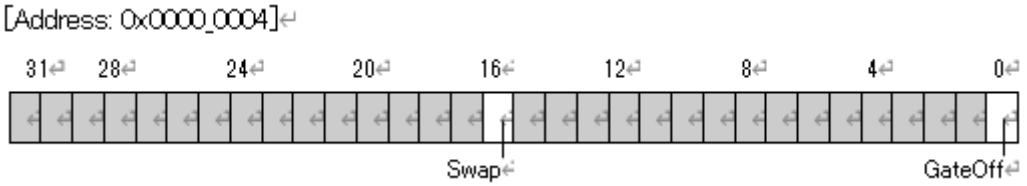
4.3 Details

4.3.1.1 Reset Register



Name	Type	Default	Description
Reset	R/W	0	Synchronous reset. Setting to 1 asserts the reset, and you must then clear it by writing 0. Unlike reset_n, this does not clear register contents. Upon setting to 1, the core asserts the rstReq signal to notify the external system of a reset condition. The external system must respond by asserting rstAck (or keep it asserted if no action is needed). After the sequence completes, this bit automatically returns to 0.

4.3.1.2 System Register



Name	Type	Default	Description
Swap	R/W	0	Word Swap setting. When set to 1, the upper and lower 32-bit halves of the 64-bit bus are swapped.. Swapping within 32-bit words is controlled via the command list.
GateOff	R/W	0	Gated Clock Off Mode. When set to 1, all bits of the gate signal are fixed to 1, effectively disabling clock gating.

5. Command List Description

5.1 Overview

- The command list is located in memory at an address specified by the value output from ***PSS*** (iAddr).
- Once ***SigSort*** is activated, it retrieves the command list and stores its content into internal registers.
- Each stage in the pipeline manages its own copy of the command list independently. → This allows concurrent operation of multiple stages even during pipeline execution, with different command configurations. → Therefore, no synchronization commands are needed.
- For reserved registers or fields, set the value to '0'.
- Addresses listed are relative to the address provided by ***PSS***.

5.2 Definition

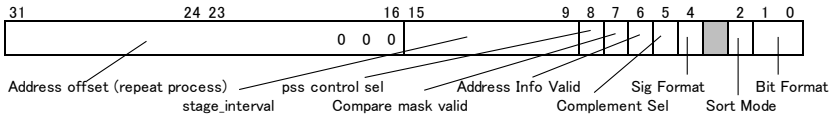
Address Offset	Command Name	Description
0x00	Cntl	Control command
0x04	IndexInfo	Amount of data to sort
0x08	Data Source Address	Address from which to read data
0x0C	Data Mask Bit	Data masking

Address Offset	Command Name	Description
0x10	Data Template Address	Temporary storage address for data
0x14	Address Info Template	Temporary storage address for address info
0x18	Data Destination Address	Destination address for sorted data
0x1C	Address Info Destination	Destination address for address/counter info

5.3 Details

5.3.1.1 Cntl Command

○ Addres:0x00
制御コマンド



Name	Size	Description
Bit Format	2	8/16/32bitより比較データを選択 0:8bit 1:16bit 2:設定禁止 3:32bit
Sort Mode	1	0:すべてのデータを昇順ソートします。 1:ユニークモード、ユニークなデータで昇順ソートします。 また、データのカウント値をアドレス情報の代わりに格納します。
Sig Format	1	符号ビットありなしを選択 0:符号ビットあり 1:符号ビットなし

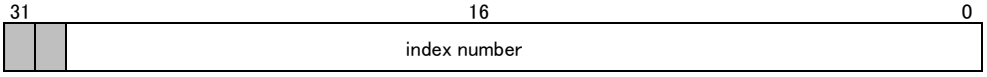
※各フォーマットによる[5:0]の設定値は以下

8bit固定小数点	010001
16bit IEEE754BIn16	000010
16bit Unsigned Short	110010
16bit Signed Short	100010
32bit IEEE754	000011
32bit Unsigned Integer	110011
32bit Signed Integer	100011

Address Info Valid	1	ソート後にアドレス情報を格納するかどうかを選択 ※ユニークモード時は、カウント情報 0:無効(格納しない) 1:有効(格納する)
Compare mask valid	1	比較時のデータマスクをするかしないかを選択 ※本機能が有効の場合は、Sig Formatの値は無効となり、自動的に 符号ビットなしとなります。
pss control sel	1	連続起動時の回数ソースをindex YかDeltaのどちらかを選択する [0]： Deltaを使用する(Delta分繰り返す) [1]： index Yを使用(繰り返しはPSSに委ね、index Yはあくまでも現在の繰り返し回数)
stage interval	7	ステージ間のインターバルを64cycle単位で設定します。 ※本ビットは、一時データの読出し時に前ステージの書き込みが間に合わない場合に有効です。 推奨値は、システムによって変わります。 0:インターバルなし 1: 64 cycle (= 1 x 64) ～ 127: 8128 cycle (= 127 x 64)
Address offset (repeat process)	16	deltaによってプロセスが繰り返されるときの、 データ読出し元アドレス、データ書き込み先アドレス、データー時保持アドレス のオフセットを設定します。 なお、アドレス情報書き込み先アドレス、アドレス情報一時保持アドレス に関しては、Bit Format 8bitの場合はx4、16bitの場合はx2、32bitの場合はx1して オフセットとして使用します。 ※下位3bitは、0固定です。

5.3.1.2. Indexinfo Command

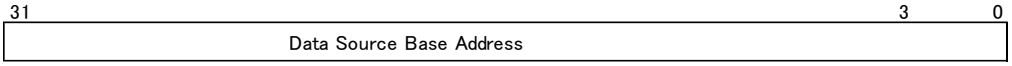
○ Addres:0x04
INDEX情報



Name	Size	Description
index number	30	Sortするデータ量を設定 設定する値は、実際のデータ数-1とする。 奇数設定制限はなし 0～1073741823(データ量1～1073741824)

5.3.1.3.Data Source Address Command

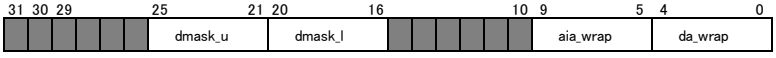
○ Addres:0x08
データ読出し元アドレス



Name	Size	Description
Data Source Base Address	32	ソートするデータが格納してあるメモリの先頭アドレスを指定 ※ソートステージ1の場合は不要のため無視します。 ※[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. Data Mask Bit

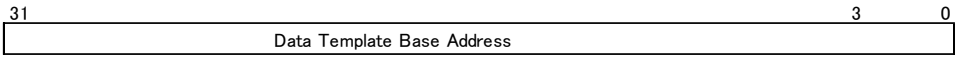
○ Addres:0x0C
データマスク



Name	Size	Description
da_wrap	5	データ格納メモリのアドレス(source/distination共通)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。
aia_wrap	5	アドレス情報/カウンタ情報格納メモリのアドレス(source/distination共通)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。
dmask_l	5	比較時にデータをマスクする場合にマスクするビットを設定します。 本ビットでのマスクは、データの下位側をマスクする際に使用します。 下位側から何ビット目以下をマスクして比較するかを設定可能です。 例) dmask_l = 8 0xFFFF_FF??として比較します。 ※dmask_uとは、&して適用します。 ※データが8bitの場合は、[31:24]が有効ビットです。 例えば、8bitの下位1bitを比較しない場合は、25を指定します。 ※データが16bitの場合は、[31:16]が有効ビットです。
dmask_u	5	比較時にデータをマスクする場合にマスクするビットを設定します。 本ビットでのマスクは、データの上位側をマスクする際に使用します。 上位側から何ビット目以上をマスクして比較するかを設定可能です。 例) dmask_u = 8 0x??FF_FFFFとして比較します。 ※dmask_lとは、&して適用します。 ※データが8bitの場合は、[31:24]が有効ビットです。 例えば、8bitの上位1bitを比較しない場合は、1を指定します。 ※データが16bitの場合は、[31:16]が有効ビットです。

5.3.1.5. Data Template Address Command

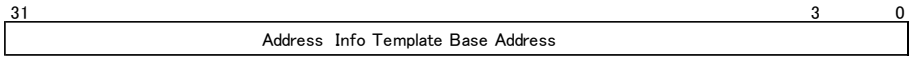
○ Addres:0x10
データー時保持アドレス



Name	Size	Description
Data Template Base Address	32	ソートしたデータを一時格納するメモリの 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。

5.3.1.6. Address Info Template Address Command

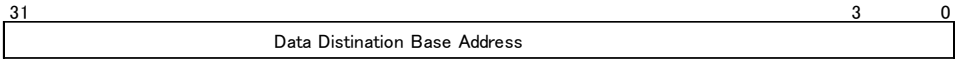
○ Addres:0x14
アドレス情報一時保持アドレス



Name	Size	Description
Address Info Template Base Address	32	ソートしたアドレス情報を一時格納するメモリの 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。 ※ユニークモード時も設定してください。

5.3.1.7. Data Destination Address Command

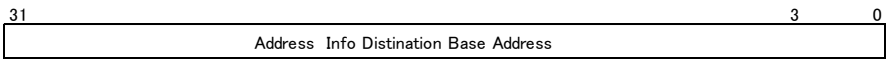
○ Addres:0x18
データ書き込み先アドレス



Name	Size	Description
Data Destination Base Address	32	ソートしたデータを格納するメモリの先頭アドレスを指定 ※ただし、最終ステージ以外は、一時データとなる。 ※[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。

5.3.1.8. Address Info Destination Address Command

○ Addres:0x1C
アドレス/カウンタ情報書き込み先アドレス



Name	Size	Description
Address Info Destination Base Address	32	ソートしたアドレス情報を格納するメモリの 先頭アドレスを指定 ※ただし、最終ステージ以外は、一時データとなる。 ※[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。 ※ユニークモード時は、カウント情報の格納先となります。

6. Application Notes

6.1 Additional Notes

6.1.1 SRAM Used

The following table describes the SRAM used in this core.

In total, 10 KBytes of SRAM are used. All SRAM blocks operate with the base clock (clk).

※ The actual required memory capacity can be reduced depending on the memory access

Block	Purpose	Type	Bit	Words	Mask	Size(Byte)	Count	Clock
FIFO	Temporary data storage	Dual	64	64	-	512	20	Clk signal