

Vector Processor

Permute Block

Micro-Architecture Specification

|  |  |
| --- | --- |
| Document Number | xxx-xxxxx |
| Date Issued | Apr 10, 2020 |
| Status  Version | Done  1.0 |



**CONFIDENTIAL**

Revision History

| Rev. | Date | Comment |
| --- | --- | --- |
| 0.1 | May 29, 2019 | Initial Draft |
| 0.2 | Jun 07, 2019 | Changes necessary to reflect 64B data-width requirement from FB. |
| 0.3 | Jul 02, 2019 | Added Instruction Latency, throughput information.  Added Instruction issue information for lmul>1 purpose.  Introduced Other & future consideration information. |
| 1.0 | Apr 10, 2020 | Updated to reflect the actual implementation. Introduce instruction latency and throughput summary information. |
|  |  |  |
|  |  |  |

|  |  |  |  |
| --- | --- | --- | --- |
| **Document Sign-off List** | | | |
| **Department** | **Name** | **Date** | **Comment** |
| VLSI |  | 2019-xx-xx |  |
|  |  |  |  |

Table of Contents

[Revision History 2](#_Toc12865988)

[List of Tables 5](#_Toc12865989)

[List of Figures 6](#_Toc12865990)

[1. Overview 7](#_Toc12865991)

[1.1. Challenges 8](#_Toc12865992)

[2. Design Description 9](#_Toc12865993)

[2.1. Switching Mux & Control 11](#_Toc12865994)

[2.2. Zeroing of O/P Data 13](#_Toc12865995)

[2.3. 1 extension of move Data 13](#_Toc12865996)

[2.4. Write Byte Mask Control 13](#_Toc12865997)

[2.5. Instruction Flow 14](#_Toc12865998)

[2.5.1. Integer Extract Instruction – vext 14](#_Toc12865999)

[2.5.2. Integer Scalar Move Instruction - vmv.s.x 14](#_Toc12866000)

[2.5.3. Floating point scalar move from VRF - vmove.f.s 15](#_Toc12866001)

[2.5.4. Floating point scalar move to VRF - vmove.s.f 15](#_Toc12866002)

[2.5.5. Vector Slideup Instruction – vslideup 15](#_Toc12866003)

[2.5.6. Vector Slidedown Instruction – vslidedown 16](#_Toc12866004)

[2.5.7. Slide1up 17](#_Toc12866005)

[2.5.8. Slide1down 18](#_Toc12866006)

[2.5.9. Vector Register Gather Instruction – vrgather 19](#_Toc12866007)

[2.5.10. Vector Compress Instruction 20](#_Toc12866008)

[3. Signal Description 21](#_Toc12866009)

[3.1. Signal Definition 21](#_Toc12866010)

[4. Instruction Summary & Performance 23](#_Toc12866011)

[5. Issues 24](#_Toc12866012)

[6. Other Considerations 25](#_Toc12866013)

List of Tables

Table 1Instruction & Shift Type 7

Table 2 Instruction Overview 13

Table 3 VSEW & Byte Shift Value 14

List of Figures

Figure 1 Permute Block diagram 12

# Overview

This document requires the reader to have an overall knowledge of the permute instructions and the proposed micro-architecture detail.

Permute block handles the following instructions.

* vmv
* vfmv
* vslideup
* vslidedown
* vslide1up
* vslide1down
* vcompress
* vrgather

Table Instruction & Shift Type

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Shift type** | **Shift Amount** | **Comment** |
| vmv scalar <- vector | shift right | Element is specified by a scalar register. | O/P to integer scalar register file (Vicuna pipeline) |
| vmv vector <- Int scalar | No shift | NA | Insert the single integer element to element 0 of the destination. Ignores LMUL. Byte zeroing operation should take care of the 0 extension. |
| vfmv vector <- fp scalar | No shift | NA | Insert a single fp element to element 0 of the destination |
| vfmv fp scalar <- vector | shift right | Element is specified by the integer scalar register. | O/P to fp scalar register file (Vicuna pipeline) |
| vslidedown | shift right | element shift specified by a scalar register/immediate value |  |
| vslide1down | shift right | 1 element. Move 1 element from rs1 scalar register to the most significant position (vl-1) of the vector register. |  |
| vslideup | shift left | element shift specified by a scalar register/immediate value |  |
| vslide1up | shift left | 1 element. Move 1 element from rs1 scalar register to the least significant element position of the vector register. |  |
| vcompress | shift right | Each element experiences different shift amount. Shift amount depends on the “mask” register. |  |
| vrgather | shift left & right (indexed based) | individual element shift (right or left – determined by index value) |  |

## Challenges

* Physical design implementation due to the magnitude of the wire and their routing pose a major convergence issue. This also means frequency scaling will be an issue i.e., scaling to Vicuna pipeline will be difficult.
* Reusing a fixed DP for all permute instruction will be ideal. However, the performance should not be affected.
* Reusing a fixed DP may not be efficient for a given particular operation. A carefully crafted design with appropriate tradeoff is a challenge.
* The micro-architecture should not only be frequency scalable but provide good configurability with minimizing the impact to configure rest of the VPU.

# Design Description

The permute datapath has a set of muxes stitched together to perform various shift operations required by the Vector Extension ISA.

* Permute block is a one block and is independent of the number of lanes configured in VPU.
  + It is not a per lane block since it involves data crossing the lanes.
* The maximum datapath width supported is 512b(64B).
* The switching mux is spread across 2 cycles (ve1, ve2) to accommodate timing.
  + Note: The split implies holding of intermediate data. For 64:1 mux, the number of registers required to hold the pipe data is 16B \* 8b \* 64Element(Bytes)= 8192 (8K).

Current implementation default split is 8:1 mux followed by 8:1 mux. The total size of the pipe register is 8B \* 8b \* 64Elements(Bytes) = 4096(4K) flop.

* Issue unit is expected to issue appropriate uop to account for the difference in the DP width.
* Issue unit is expected not to issue exceptioning instructions.
* Try using the DP that may be required for implementing vrgather instruction for other instructions as well there by minimizing the area.

## Pipe stages:

* The main permute pipeline has 3 stage – ve1, ve2, ve3 in addition to v2.
* Compress pipeline has 2 stages – ve1c & ve2c in addition to v2.
* Compress pipeline’s ve2c overlaps with ve1 of the main pipeline.
* Pipe control involves “run” signal which says advance from the current pipestage to the next pipestage.
* Refer to Figure 1 Permute Block diagram.

### V2 stage

* This stage has very less logic due to timing reasons.
* Data muxes selects appropriate data source for use.
* Minor opcode decoding, shift amount selection according the uop type are among other misc logic in v2 stage.
* For power saving reasons, there are 2 run signals in the v2 stage. One involves compress only run signal and the other involving no compress related information that needs to be clk gated from v2 to ve1 stage.

### VE1 stage

* This is the first of the 2 stage muxing.
* Switching mux control is adjusted to accommodate compress instruction’s.
* Since the DP is byte organized, the selection provided by the VRF register for .vv type instruction, needs following adjustments.
  + The selection control for each byte in an element is repeated with the same selection information provided by the VRF register.
  + The lower bits of this selection bits needs adjustment for the byte index within the element.
* In order to keep the DP and control logic simple, rather than manipulating the mux selection, a zero mask for the VRF is generated according to the instruction need. This zero-mask generation is required for the following
  + vrgather.vv instruction: to zero-out the data when the index is out of range.
  + vrgather.vx instruction: to zero-out the data when the index value is out of range.
  + slidedown instruction: to zero-out the elements vl-1 to maxvlen-slide/shift amount.
* slide1up/down instruction requires that the scalar data from xrf needs to be inserted in the VRF wr data. The shift amount of this scalar data, its conditioning for sign-extension and mux insertion control is done in this ve1 stage.
* Adjustment to the VRF wr mask:
  + For slideup instruction, per the ISA, the elements from 0 to shift\_amount-1 should not be affected. For this reason, a special VRF wr mask is generated which will be “ANDED” with the vrf mask provided by the frontend.
  + vmv instruction related wr mask generation.
  + Compress instruction related.
  + For vrgather instruction different source registers in the register group is iterated for each of the destination register in the register group. This requires special wr mask generation depending on the index value for each element. Another mask to enable zero-ing of data when the index is out of range is also generated.

### VE2 stage

* The data switching mux continues in this stage to get the final data for instructions independent of LMUL or for LMUL=1.
* Zero-ing of the final data, sign-extension of the data is performed in this stage.

### VE3 stage

* This stage serves to accumulate data for slide, gather, compress instructions.
* For LMUL>1, the data source registers are read only once (slide, gather instructions) and presented to permute block. The data from a given source register may serve as data for 1 or 2 destination register within the register group. This stage serves as means to hold on to the data for use with the next uop for slide instruction.
* This stage also accumulates the data for a given destination register in a register group for vrgather instruction in order to avoid multiple writes to VRF.
* For compress instructions with LMUL>1, this stage accumulates merges the data from various source register. Upon assembling a full dataset for a compress instruction, the data is evicted or written to VRF. Since the mask bits are not known earlier, the latency of the instruction cannot be predetermined. The frontend scoreboard keeps the register group busy until the entire sequence of uop for a compress instruction completes execution.
* For compress instruction it is possible that the last uop may trigger 2 writes to VRF. Assume 63 bytes in the accumulator register in VE3 stage. Further assume 2 bytes are to be merged by the last uop to the accumulator. This will result in 65 bytes of data. For this reason, the logic merges one byte to the accumulator, writes the data to VRF. In the subsequent cycle, the left over byte is written back to the VRF. The example is shown for VLEN=64bytes.

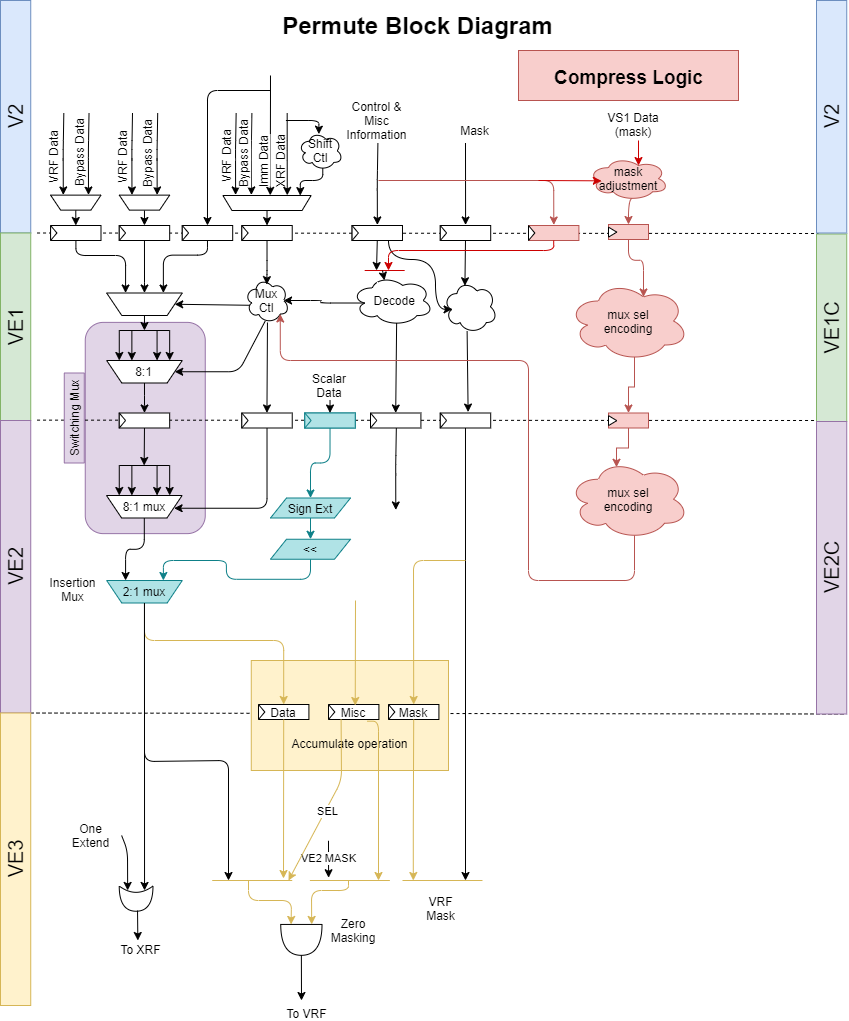


Figure Permute Block diagram

## Switching Mux & Control

A simple routing of the index value for vrgather instruction would make the mux select control a little complicated for other instructions using the same datapath. However, since the shift control information for most of the other instructions are known in v1 & v2, there is sufficient time to derive the control logic for the mux. The following table provides an idea of the mux control logic design. A 512-bit data input is considered and hence there are 64-byte elements that are possible. The value of 64 in the following represents the number of the smallest elements.

Table Instruction Overview

|  |  |  |  |
| --- | --- | --- | --- |
| **Instruction** | **Operation** | **Mux select/control/index value** | **Comment** |
| **vslidedown** | Shift Right | Element index + offset |  |
| **vslide1down** | Shift Right | Element index + 1 |  |
| **vslideup** | Shift Left | Element index - offset |  |
| **vslide1up** | Shift Left | Element index - 1 |  |
| **vcompress** | Shift Right | Each mux’s select signal may vary.  Dedicated logic generates the switching mux control based on the mask register. | explained in later section. |
| **vrgather** | Indexing | Element index as in the vector register |  |
| **vextract** | Variable Shift Right | Element index (w/ byte offset adjustment) | explained in later section. |

* Translate the element shift/index to byte shift.
  + In the above table the offset/idx is actually **(instruction expressed offset << vsew)**.

Table VSEW & Byte Shift Value

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| vsew setting | | | | Byte shift for element index for various vsew setting. Converting element shift to byte shift. | | | | | | | | |
|
|
| vsew(b) | SEW | vsew(d) | SEW/8 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 10 |
| 0 | 8 | 0 | 1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 10 |
| 1 | 16 | 1 | 2 | 0 | 2 | 4 | 6 | 8 | 10 | 12 | 20 |
| 10 | 32 | 2 | 4 | 0 | 4 | 8 | 12 | 16 | 20 | 24 | 40 |
| 11 | 64 | 3 | 8 | 0 | 8 | 16 | 24 | 32 | 40 | 48 | 80 |
| 100 | 128 | 4 | 16 | 0 | 16 | 32 | 48 | 64 | 80 | 96 | 160 |
| 101 | 256 | 5 | 32 | 0 | 32 | 64 | 96 | 128 | 160 | 192 | 320 |
| 110 | 512 | 6 | 64 | 0 | 64 | 128 | 192 | 256 | 320 | 384 | 640 |
| 111 | 1024 | 7 | 128 | 0 | 128 | 256 | 384 | 512 | 640 | 768 | 1280 |

* vcompress related control signal.
  + Logic to detect the 1/0 and appropriately generate the mux control/index values.



## Zeroing of O/P Data

Zeroing of the data is required for the following

* Extract: If SEW < XLEN, the value is zero-extended to XLEN.
* Extract: x[rs1] ≥ VLEN/SEW (index is out of range), zero is returned.
* Move to vector: Zero out other element - 0 < index < VLEN/SEW.
* Move to vector: Zero extension for SEW>XLEN.
* Gather: Index out of bound results in 0-ing of the data.
* Gather: EDIV to be accounted for 0-ing of sub-element. Not required for FB. So it is not implemented.
* Slidedown: vl-offset to vl elements will be 0-ed out.

## 1 extension of move Data

The following instructions requires 1/sign extension for the data that is being moved from XRF/FRF to VRF.

* vmv.s.x: sign extend for SEW < FLEN
* vfmv.s.f: one extend for SEW > FLEN

## Write Byte Mask Control

The shifted data through switching mux may require further conditioning before being written to the VRF or sent to scalar register file.

* All the bytes in the Inactive element will result in write byte mask value of 0.
* Slideup: 0 < i < max(vstart, OFFSET) Unchanged
* Compress instruction needs to generate the appropriate wr mask. This depends on the valid input mask bits.

## Instruction Flow

* If vl=0 then vslide[1]up/down instructions are not issued to the block since the destination registers are unchanged.
* Overlapping register condition is not accounted in the permute block. Illegal instruction exception is raised by the issue unit and the instruction is not seen by the vector functional blocks.
* No exceptioning instruction is issued by the issue block to permute block.
* There is no pipe-stalls in the permute block. Neither it is a good idea to manipulate the permute ready signal (to the frontend) for this purpose. A 3 cycle latency uop cannot be followed immediately by a 2 cycle latency uop. The issue unit is expected to take of this avoidance. Note: Similarly, a 2 cycle latency uop cannot be immediately followed by a one cycle latency uop.

### Integer Extract Instruction – vmv.x.s

* Element 0 of the source vector register is returned to integer register file.
* Ignores LMUL value.
* The switching mux select control signals is set to extract the element 0 (byte/hword/word)
* Frontend is taking care of the sign extension as may be required.
* Though there is no shift/slide operation associated with this instruction, the Datapath used for the vrgather/slide is reused to avoid unnecessary hardware addition.
* Though this instruction may be made to complete in 1 cycle, in order to avoid various conflicts, the instruction is forced to have 2 cycle latency.

### Integer Scalar Move Instruction - vmv.s.x

* Data is inserted to the element [0] of the destination register.
* The main switching mux used for the vrgather instruction is not used for this instruction. However, the o/p of the switching mux goes through another level of muxing for scalar register data insertion. This level of mux is used for this instruction.
* The scalar data insertion mux involves, sign-extension and shift operation. However, there is no shift associated with this instruction. The shift operation is required for slidedown instruction.
* Zero mask is appropriately set. Only the lower most element is enabled.
* Wr Byte Mask enables all the elements.
* For vl=0, issue unit is not expected to issue the instruction to permute block.

### Floating point scalar move from VRF – vmv.f.s

* Element 0 of vs2 data is extracted and provided to the frontend.
* Ignores LMUL and vector register group.
* The operation is similar to that of vmv.x.s.
* Data 1-ext is performed on the extracted data when SEW < FLEN.

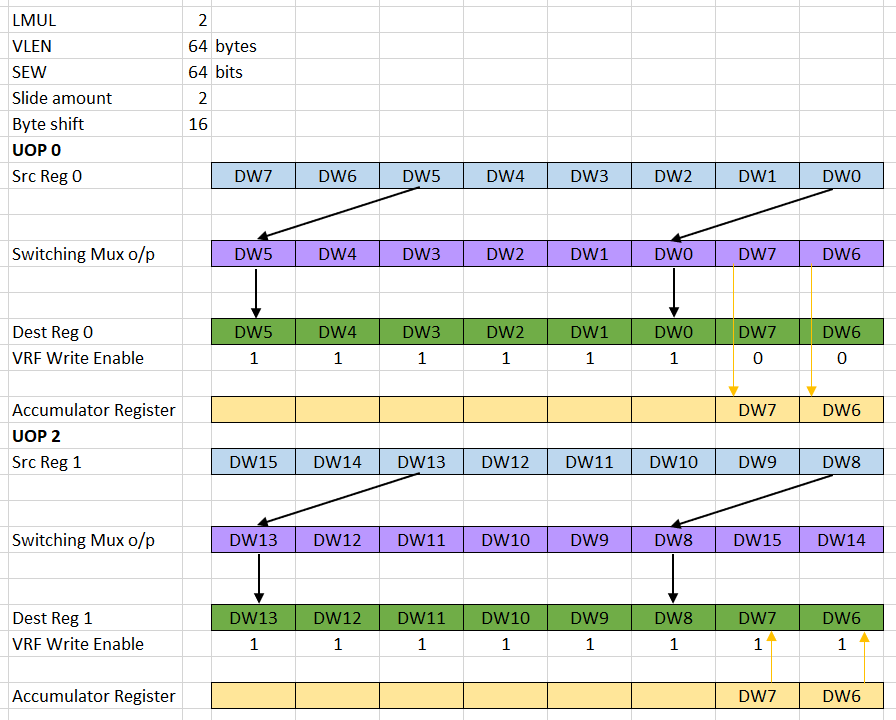
### Floating point scalar move to VRF - vmv.s.f

* The input data is conditioned for 1 extension (NaN-boxing).
* Data is inserted to the element [0] of the destination register.
* Switching mux is not used for this instruction. Instead the insertion mux is used for inserting the scalar floating-point data to element 0 of the vector data.
* Zero mask is appropriately set. Only the lower most element is enabled.
* Wr mask enables only the element 0
* For vl=0, issue unit is not expected to issue the instruction to permute block.

### Vector Slideup Instruction – vslideup

* Left shift the data by offset amount.
* Convert the element mask to equivalent byte mask.
* Wr Byte Mask is adjusted so that elements from 0 to max(vstart, offset) is unchanged.
* Shift value of each byte is determined by destination ByteID - Offset.
  + For e.g, assume VLEN=64B, lmul=1, slide amount is 2. The o/p byte 4 will receive data from i/p byte 2. Likewise, destination byte 63 will receive the data from input bye 61.
* Insert mux is not used for slideup operation since there is no scalar data insertion involved.
* When LMUL>1 or VLEN > (SWPL & #Lanes), the shift control for the switching mux needs some additional check.
* Take an example of VLEN=64B, lmul=2 and slide amount is 2. Byte 62 for register 0 of the source register group should find itself in Byte 0 of register 1 of the destination group. The source registers are read only one time. So it is important to register the data to be written from one source register offset to a different destination register offset. The accumulator register in ve3 stage is used for this purpose.
* When slide/shift amount is greater than the size of VLEN, the frontend determines the initial destination register that needs to be written in the lmul register group.
  + For e.g, for lmul=2, sew=8b, shift/slide amount=64, the source register 0 in the lmul register group is written to destination register offset 1 in the lmul register group.
  + The number of uop varies from 0 to a maximum of LMUL which depends on the slide/shift amount.
  + When shift amount > (LMUL \* VLEN), there is no uop issued to permute block.
  + When shift amount < (LMUL \* VLEN), the number of uop issued to permute block is from 1 to LMUL depending on the shift/slide amount.

**Slideup Illustration**



### Vector Slidedown Instruction – vslidedown

* Right shift the data by offset amount. Similar to slideup instruction the switching mux performs the rotate operation.
* Wr Byte Mask needs to account for not changing the elements from 0 to vstart. For now, vstart=0.
* Shift value of each byte is determined by ByteID + Offset.
  + For e.g, assume lmul=1, sew=8, VLEN=64B, and a slide/shift amount (offset) = 2. Source byte 63 is moved to destination byte 61. Likewise byte 0 of the source register is moved to destination register byte 0. Dest byte[x] = Source byte[x + offset]

### Slide1up

* Same as slideup except that offset is 1 and scalar element input data is inserted in element 0 of the destination.

### Slide1down

* Same as slide1down except that offset is 1 and scalar element input data is inserted in element (vl-1) of the destination.
* The scalar data is inserted to the VRF data after the switching mux only for destination register offset of 0 in the lmul register group.

### Vector Register Gather Instruction – vrgather

* Data for Out of bound index is zero. The zeroing logic will account for this. No special attention is required for the switching mux control for this reason.
* Since the DP is primarily set for byte shifting, for SEW > 8, byte id within an element will serve as the byte offset for the switching mux control. This change to the incoming index value may be performed in ve1, since the byte shifting within an element takes place in ve3 stage.
* In case of LMUL >1, each of the destination register (in the group) will be iterated through all of the input registers (in the group).
* Non-zero EDIV implies the indexing is within the sub-elements of a given element. Index adjustment is required and will depend on SEW value. Since FB does not require this support for permute block, the plan is not to implement this feature.
* Since Non-zero EDIV value implies indexing of sub-elements within an element, issue unit need not iterate all source register slices for one destination register slice. This is a performance optimization.

### Vector Compress Instruction

* The permute block treats this like a vrgather instruction with vcompress block supplying the required index values for the switching mux.
* The pipe flow for vcompress instruction is as follows.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| lmul=1 |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |
| compress\_0 | preshift | v2 | ve1c | ve2c |  |
| (PSEUDO)gather\_0 |  |  |  | ve1 | ve2 |



Pipe stages ve1c and ve2c is associated with compress instruction only & also the compress mask operand goes through pre-shift operation for mask conditioning before the data is made valid to the compress instruction uop.

* The pipestages v2, ve1c & ve2c accepts this preconditioned mask register value and provides the switching mux logic with the mux select values. Ve2c overlaps with ve1 pipestage of the permute block.
* The input operand is switched according to the mux control value received from the compress module.
* For LMUL>1, the compressed data must be accumulated until VLEN bytes of data and is written back to VRF. Any left-over accumulated data in ve3 stage is written back to VRF though it may not contain entire VLEN bytes of data.
* During the end of instruction accumulator writeback when LMUL>1, there is a possibility that there are no valid bytes. In this case, the VRF wr mask is set to 0.

# Signal Description

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| input |  | clk; | // | clock |
| input |  | rstn; | // | active low reset signal |
| input |  | viper\_valid; | // | permute instruction packet valid |
| input | [`EX\_CTRL\_RANGE] | viper\_ex\_ctrl; | // | opcode |
| input | [1:0] | viper\_vtype\_lmul; | // | combination of vlen & lmul |
| input | [`CSR\_VL\_RANGE] | viper\_vl; | // | vector length |
| input | [`VP\_DATA\_RANGE] | v2\_vrf\_data0; | // | read port0 data |
| input | [`VP\_DATA\_RANGE] | v2\_vrf\_data1; | // | read port1 data |
| input | [`VP\_DATA\_RANGE] | v2\_vrf\_data2; | // | read port2 data |
| input | [`VP\_DATA\_RANGE] | v2\_vrf\_data3; | // | read port3 data |
| input | [1:0] | viper\_vs1\_select; | // | vs1 data |
| input |  | viper\_vs1\_valid; | // |  |
| input | [1:0] | viper\_vs2\_select; | // | vs2 data |
| input |  | viper\_vs2\_valid; | // |  |
| input | [1:0] | viper\_vs3\_select; | // | vs3 data (not used) |
| input |  | viper\_vs3\_valid; | // |  |
| input | [`VP\_MASK\_RANGE] | viper\_msk\_bits; | // | vmask data |
| input | [XLEN-1:0] | viper\_rs1\_data; | // | scalar register data (xrf/frf) |
| input | [XLEN-1:0] | viper\_rs2\_data; | // | imm data |
| input |  | viper\_rs1\_valid; | // | rs1 valid info - determines data input to be used |
| input |  | viper\_imm\_valid; | // | imm data in rs2 is valid |
| input | [`CSR\_VL\_RANGE] | viper\_vstart; | // | vector element start register |
| input | [`SEW\_RANGE] | viper\_vtype\_sew; | // | single element width |
| input | [`MOP\_CNT\_RANGE] | viper\_mop\_cnt; | // | micro-op count for shift adjustment |
| input |  | viper\_mop\_first; | // | first uop |
| input |  | viper\_mop\_last; | // | last uop |
| input |  | viper\_vrgather; |  |  |
| output | [XLEN-1:0] | viper\_rd\_data; | // | permute -> xrf data |
| output | [`VP\_DATA\_RANGE] | viper\_data; | // | permute data o/p |
| output | [`VP\_MASK\_RANGE] | viper\_mask; | // | permute data wr mask |
| output |  | viper\_data\_valid; | // | permute data valid |
| output | [`MOP\_CNT\_RANGE] | viper\_wr\_offset; | // | compress related dest reg offset |

# Instruction Summary & Performance

The following table considers the effective instruction latency & throughput and not the uops. The compress instruction being complex, requires additional attention and is subject to change since it is dependent on many units and assumptions.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Item | **vrgather** | **vcompress** | **slideup/ down** | **slide1up/ down** | **move (v->x)** | **move  (f->v)** | **move (v->f)** | **move (x->v)** | **Comment** |
| vs2 | x | x | x | x | x |  | x |  |  |
| vs1 | x | x |  |  |  |  |  |  |  |
| v0.t | x | x | x | x |  |  |  |  |  |
| Xreg/Imm | x |  | x | x |  |  |  |  | vrgather.vi requires imm/int scalar reg |
| Destination Type | v | v | v | v | x | f | v | v | v - vector reg x - int scalar f - fp scalar |
| #uops |  |  |  |  |  |  |  |  |  |
| LMUL dependency | x | x | x | x |  |  |  |  |  |
| vstart | trap check |  | x | x |  |  |  |  | currently vstart=0 |
| Trap | vstart!=0 |  |  |  |  |  |  |  | frontend implementation |
| Comment | vs1/Xreg/Imm are mutually exclusive |  |  |  |  |  |  |  |  |

vcompress

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| lmul | 1 | 2 | 4 | 8 |
| #uop issued by frontend | 1 | 2 | 4 | 8 |
| effective latency with preshift | 5 | 7 | 9 | 13 |
| effective latency with preshift + noforwarding accounting | 6 | 8 | 10 | 14 |
| compress uop throughput | 1 | 1 | 1 | 1 |
| compress instruction throughput | 2 | 2 | 2 | 2 |

Vslidedown illustration

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| slidedown |  |  |  |  |  |  |  |  |  |  |  |
| lmul | 8 |  |  |  |  |  |  |  |  |  |  |
| shift amount | 65 | (bytes) |  |  |  |  |  |  |  |  |  |
| src seq | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 0 |  |  |  |
| dst seq |  |  |  | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| uop 0 | v2 | ve1 | ve2 | ve3 |  |  |  |  |  |  |  |
| uop 1 |  | v2 | ve1 | ve2 | ve3 |  |  |  |  |  |  |
| uop 2 |  |  | v2 | ve1 | ve2 | ve3 |  |  |  |  |  |
| uop 3 |  |  |  | v2 | ve1 | ve2 | ve3 |  |  |  |  |
| uop 4 |  |  |  |  | v2 | ve1 | ve2 | ve3 |  |  |  |
| uop 5 |  |  |  |  |  | v2 | ve1 | ve2 | ve3 |  |  |
| uop 6 |  |  |  |  |  |  | v2 | ve1 | ve2 | ve3 |  |
| uop 7 |  |  |  |  |  |  |  | v2 | ve1 | ve2 | ve3 |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| lmul | 4 |  |  |  |  |  |  |  |  |  |  |
| shift amount | 128 | (bytes) |  |  |  |  |  |  |  |  |  |
| src seq | 2 | 3 | 0 | 1 |  |  |  |  |  |  |  |
| dst seq |  |  |  | 0 | 1 | 2 | 3 |  |  |  |  |
| uop 0 | v2 | ve1 | ve2 | ve3 |  |  |  |  |  |  |  |
| uop 1 |  | v2 | ve1 | ve2 | ve3 |  |  |  |  |  |  |
| uop 2 |  |  | v2 | ve1 | ve2 | ve3 |  |  |  |  |  |
| uop 3 |  |  |  | v2 | ve1 | ve2 | ve3 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| lmul | 2 |  |  |  |  |  |  |  |  |  |  |
| shift amount | 64 | (bytes) |  |  |  |  |  |  |  |  |  |
| src seq | 1 | 0 |  |  |  |  |  |  |  |  |  |
| dst seq |  |  |  | 0 | 1 |  |  |  |  |  |  |
| uop 0 | v2 | ve1 | ve2 | ve3 |  |  |  |  |  |  |  |
| uop 1 |  | v2 | ve1 | ve2 | ve3 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |  |  |
| lmul | 1 |  |  |  |  |  |  |  |  |  |  |
| shift amount | 34 | (bytes) |  |  |  |  |  |  |  |  |  |
| src seq | 0 |  |  |  |  |  |  |  |  |  |  |
| dst seq |  |  | 0 |  |  |  |  |  |  |  |  |
| uop 0 | v2 | ve1 | ve2 |  |  |  |  |  |  |  |  |

## Instruction Issue

|  |  |  |
| --- | --- | --- |
| **Number of uops calculation** | | |
| **Instruction** | **# uops** | **Comment** |
| vext | 1 |  |
| vmv | 1 |  |
| vfmv | 1 |  |
| vfmv | 1 |  |
| vslideup | 0 to LMUL(max) | The number of uop depends on the shift amount. |
| vslidedown | LMUL |  |
| vslide1up | LMUL |  |
| vslide1down | LMUL |  |
| vrgather | LMUL ^ 2 |  |
| vcompress | LMUL |  |



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **LMUL independent instructions** | | | | | | |
| **Instruction** | **uop** | **vs2** | **vs1** | **Latency to vd** | **Latency to xrf result** | **Comment** |
| vext | 1 | vs2 |  |  | 2 |  |
| vmv | 1 |  |  | 2 |  |  |
| vfmv | 1 | vs2 |  |  | 2 |  |
| vfmv | 1 |  |  | 2 |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **LMUL = 1** | | | | | | |
| **Instruction** | **uop** | **vs2** | **vs1** | **vd** | **Latency to vd** | **Comment** |
| vslideup | 1 | vs2 |  | vd | 2 |  |
| vslidedown | 1 | vs2 |  | vd | 2 |  |
| vslide1up | 1 | vs2 |  | vd | 2 |  |
| vslide1down | 1 | vs2 |  | vd | 2 |  |
| vrgather | 1 | vs2 | vs1 | vd | 2 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **LMUL = 2** | | | | | | |
| **Instruction** | **uop** | **vs2** | **vs1** | **vd** | **Latency to vd** | **Comment** |
| **vslideup** | 1 | vs2[0] |  | vd[start] | 2 |  |
|  | 2 | vs2[1] |  | vd[start+1] | 2 |  |
|  |  |  |  |  |  |  |
| **vslidedown** | 1 | vs2[start] |  | vd[0] | 3 |  |
|  | 2 | vs2[start+1] |  | vd[1] | 3 |  |
|  |  |  |  |  |  |  |
| **vslide1up** | 1 | vs2[0] |  | vd[0] | 2 |  |
|  | 2 | vs2[1] |  | vd[1] | 2 |  |
|  |  |  |  |  |  |  |
| **vslide1down** | 1 | vs2[0] |  | vd[0] | 3 |  |
|  | 2 | vs2[1] |  | vd[1] | 3 |  |
|  |  |  |  |  |  |  |
| **vrgather** | 1 | vs2[0] | vs1[0] | vd[0] | 2 |  |
|  | 2 | vs2[1] | vs1[1] | vd[0] | 2 |  |
|  | 3 | vs2[0] | vs1[0] | vd[1] | 2 |  |
|  | 4 | vs2[1] | vs1[1] | vd[1] | 2 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **LMUL = 4** | | | | | | |
| **Instruction** | **uop** | **vs2** | **vs1** | **vd** | **Latency to vd** | **Comment** |
| **vslideup** | 1 | vs2[0] |  | vd[start] | 2 |  |
|  | 2 | vs2[1] |  | vd[start+1] | 2 |  |
|  | 3 | vs2[2] |  | vd[start+2] | 2 |  |
|  | 4 | vs2[3] |  | vd[start+3] | 2 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **vslidedown** | 1 | vs2[start] |  | vd[0] | 3 |  |
|  | 2 | vs2[start+1] |  | vd[1] | 3 |  |
|  | 3 | vs2[start+2] |  | vd[2] | 3 |  |
|  | 4 | vs2[start+3] |  | vd[3] | 3 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **vslide1up** | 1 | vs2[0] |  | vd[0] | 2 |  |
|  | 2 | vs2[1] |  | vd[1] | 2 |  |
|  | 3 | vs2[2] |  | vd[2] | 2 |  |
|  | 4 | vs2[3] |  | vd[3] | 2 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **vslide1down** | 1 | vs2[0] |  | vd[0] | 3 |  |
|  | 2 | vs2[1] |  | vd[1] | 3 |  |
|  | 3 | vs2[2] |  | vd[2] | 3 |  |
|  | 4 | vs2[3] |  | vd[3] | 3 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **vrgather** | 1 | vs2[0] | vs1[0] | vd | 2 |  |
|  | 2 | vs2[1] | vs1[1] | vd | 2 |  |
|  | 3 | vs2[2] | vs1[2] | vd | 2 |  |
|  | 4 | vs2[3] | vs1[3] | vd | 2 |  |
|  | 5 | vs2[0] | vs1[0] | vd[1] | 2 |  |
|  | 6 | vs2[1] | vs1[1] | vd[1] | 2 |  |
|  | 7 | vs2[2] | vs1[2] | vd[1] | 2 |  |
|  | 8 | vs2[3] | vs1[3] | vd[1] | 2 |  |
|  | 9 | vs2[0] | vs1[0] | vd[2] | 2 |  |
|  | 10 | vs2[1] | vs1[1] | vd[2] | 2 |  |
|  | 11 | vs2[2] | vs1[2] | vd[2] | 2 |  |
|  | 12 | vs2[3] | vs1[3] | vd[2] | 2 |  |
|  | 13 | vs2[0] | vs1[0] | vd[3] | 2 |  |
|  | 14 | vs2[1] | vs1[1] | vd[3] | 2 |  |
|  | 15 | vs2[2] | vs1[2] | vd[3] | 2 |  |
|  | 16 | vs2[3] | vs1[3] | vd[3] | 2 |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |
| **LMUL = 8** | | | | | | |
| **Instruction** | **uop** | **vs2** | **vs1** | **vd** | **Latency to vd** | **Comment** |
| **vslideup** | 1 | vs2[0] |  | vd[start] | 2 |  |
|  | 2 | vs2[1] |  | vd[start+1] | 2 |  |
|  | 3 | vs2[2] |  | vd[start+2] | 2 |  |
|  | 4 | vs2[3] |  | vd[start+3] | 2 |  |
|  | 5 | vs2[4] |  | vd[start+4] | 2 |  |
|  | 6 | vs2[5] |  | vd[start+5] | 2 |  |
|  | 7 | vs2[6] |  | vd[start+6] | 2 |  |
|  | 8 | vs2[7] |  | vd[start+7] | 2 |  |
|  |  |  |  |  |  |  |
| **vslidedown** | 1 | vs2[start] |  | vd[0] | 3 |  |
|  | 2 | vs2[start+1] |  | vd[1] | 3 |  |
|  | 3 | vs2[start+2] |  | vd[2] | 3 |  |
|  | 4 | vs2[start+3] |  | vd[3] | 3 |  |
|  | 5 | vs2[start+4] |  | vd[4] | 3 |  |
|  | 6 | vs2[start+5] |  | vd[5] | 3 |  |
|  | 7 | vs2[start+6] |  | vd[6] | 3 |  |
|  | 8 | vs2[start+7] |  | vd[7] | 3 |  |
|  |  |  |  |  |  |  |
| **vslide1up** | 1 | vs2[0] |  | vd[0] | 2 |  |
|  | 2 | vs2[1] |  | vd[1] | 2 |  |
|  | 3 | vs2[2] |  | vd[2] | 2 |  |
|  | 4 | vs2[3] |  | vd[3] | 2 |  |
|  | 5 | vs2[4] |  | vd[4] | 2 |  |
|  | 6 | vs2[5] |  | vd[5] | 2 |  |
|  | 7 | vs2[6] |  | vd[6] | 2 |  |
|  | 8 | vs2[7] |  | vd[7] | 2 |  |
|  |  |  |  |  |  |  |
| **vslide1down** | 1 | vs2[0] |  | vd[0] | 3 |  |
|  | 2 | vs2[1] |  | vd[1] | 3 |  |
|  | 3 | vs2[2] |  | vd[2] | 3 |  |
|  | 4 | vs2[3] |  | vd[3] | 3 |  |
|  | 5 | vs2[4] |  | vd[4] | 3 |  |
|  | 6 | vs2[5] |  | vd[5] | 3 |  |
|  | 7 | vs2[6] |  | vd[6] | 3 |  |
|  | 8 | vs2[7] |  | vd[7] | 3 |  |
|  |  |  |  |  |  |  |
| **vrgather** | 1 | vs2[0] | vs1[0] | vd[0] | 2 |  |
|  | 2 | vs2[1] | vs1[1] | vd[0] | 2 |  |
|  | 3 | vs2[2] | vs1[2] | vd[0] | 2 |  |
|  | 4 | vs2[3] | vs1[3] | vd[0] | 2 |  |
|  | 5 | vs2[4] | vs1[4] | vd[0] | 2 |  |
|  | 6 | vs2[5] | vs1[5] | vd[0] | 2 |  |
|  | 7 | vs2[6] | vs1[6] | vd[0] | 2 |  |
|  | 8 | vs2[7] | vs1[7] | vd[0] | 2 |  |
|  | 9 | vs2[0] | vs1[0] | vd[1] | 2 |  |
|  | 10 | vs2[1] | vs1[1] | vd[1] | 2 |  |
|  | 11 | vs2[2] | vs1[2] | vd[1] | 2 |  |
|  | 12 | vs2[3] | vs1[3] | vd[1] | 2 |  |
|  | 13 | vs2[4] | vs1[4] | vd[1] | 2 |  |
|  | 14 | vs2[5] | vs1[5] | vd[1] | 2 |  |
|  | 15 | vs2[6] | vs1[6] | vd[1] | 2 |  |
|  | 16 | vs2[7] | vs1[7] | vd[1] | 2 |  |
|  | 17 | vs2[0] | vs1[0] | vd[2] | 2 |  |
|  | 18 | vs2[1] | vs1[1] | vd[2] | 2 |  |
|  | 19 | vs2[2] | vs1[2] | vd[2] | 2 |  |
|  | 20 | vs2[3] | vs1[3] | vd[2] | 2 |  |
|  | 21 | vs2[4] | vs1[4] | vd[2] | 2 |  |
|  | 22 | vs2[5] | vs1[5] | vd[2] | 2 |  |
|  | 23 | vs2[6] | vs1[6] | vd[2] | 2 |  |
|  | 24 | vs2[7] | vs1[7] | vd[2] | 2 |  |
|  | 25 | vs2[0] | vs1[0] | vd[3] | 2 |  |
|  | 26 | vs2[1] | vs1[1] | vd[3] | 2 |  |
|  | 27 | vs2[2] | vs1[2] | vd[3] | 2 |  |
|  | 28 | vs2[3] | vs1[3] | vd[3] | 2 |  |
|  | 29 | vs2[4] | vs1[4] | vd[3] | 2 |  |
|  | 30 | vs2[5] | vs1[5] | vd[3] | 2 |  |
|  | 31 | vs2[6] | vs1[6] | vd[3] | 2 |  |
|  | 32 | vs2[7] | vs1[7] | vd[3] | 2 |  |
|  | 33 | vs2[0] | vs1[0] | vd[4] | 2 |  |
|  | 34 | vs2[1] | vs1[1] | vd[4] | 2 |  |
|  | 35 | vs2[2] | vs1[2] | vd[4] | 2 |  |
|  | 36 | vs2[3] | vs1[3] | vd[4] | 2 |  |
|  | 37 | vs2[4] | vs1[4] | vd[4] | 2 |  |
|  | 38 | vs2[5] | vs1[5] | vd[4] | 2 |  |
|  | 39 | vs2[6] | vs1[6] | vd[4] | 2 |  |
|  | 40 | vs2[7] | vs1[7] | vd[4] | 2 |  |
|  | 41 | vs2[0] | vs1[0] | vd[5] | 2 |  |
|  | 42 | vs2[1] | vs1[1] | vd[5] | 2 |  |
|  | 43 | vs2[2] | vs1[2] | vd[5] | 2 |  |
|  | 44 | vs2[3] | vs1[3] | vd[5] | 2 |  |
|  | 45 | vs2[4] | vs1[4] | vd[5] | 2 |  |
|  | 46 | vs2[5] | vs1[5] | vd[5] | 2 |  |
|  | 47 | vs2[6] | vs1[6] | vd[5] | 2 |  |
|  | 48 | vs2[7] | vs1[7] | vd[5] | 2 |  |
|  | 49 | vs2[0] | vs1[0] | vd[6] | 2 |  |
|  | 50 | vs2[1] | vs1[1] | vd[6] | 2 |  |
|  | 51 | vs2[2] | vs1[2] | vd[6] | 2 |  |
|  | 52 | vs2[3] | vs1[3] | vd[6] | 2 |  |
|  | 53 | vs2[4] | vs1[4] | vd[6] | 2 |  |
|  | 54 | vs2[5] | vs1[5] | vd[6] | 2 |  |
|  | 55 | vs2[6] | vs1[6] | vd[6] | 2 |  |
|  | 56 | vs2[7] | vs1[7] | vd[6] | 2 |  |
|  | 57 | vs2[0] | vs1[0] | vd[7] | 2 |  |
|  | 58 | vs2[1] | vs1[1] | vd[7] | 2 |  |
|  | 59 | vs2[2] | vs1[2] | vd[7] | 2 |  |
|  | 60 | vs2[3] | vs1[3] | vd[7] | 2 |  |
|  | 61 | vs2[4] | vs1[4] | vd[7] | 2 |  |
|  | 62 | vs2[5] | vs1[5] | vd[7] | 2 |  |
|  | 63 | vs2[6] | vs1[6] | vd[7] | 2 |  |
|  | 64 | vs2[7] | vs1[7] | vd[7] | 2 |  |

Vcompress instruction related information

The vcompress instruction’s latency depends on the mask bits and the lmul value.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Legend | | | | | | | | | | | | | | | | | |
|  | | | | | | | | | | | | | | | | | |
| Potential VRF wr for data corresponding to the uop | | | | | | | | | | | | | | | | | |
| PSUEDO gather operation not issued by frontend | | | | | | | | | | | | | | | | | |
| Pipestage | | | | | | | | | | | | | | | | | |
| ve1c - compressed pipeline ve1 stage | | | | | | | | | | | | | | | | | |
| ve2c - compressed pipeline ve2 stage | | | | | | | | | | | | | | | | | |
| ve1 - permute block main pipeline's ve1 stage (gather op) | | | | | | | | | | | | | | | | | |
| ve2 - permute block main pipeline's ve2 stage (gather op) | | | | | | | | | | | | | | | | | |

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| lmul=1 |  |  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 |  |  |  |  |
| compress\_0 | preshift | v2 | ve1c | ve2c |  |  |  |  |  |
| (PSEUDO)gather\_0 |  |  |  | ve1 | ve2 |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
| lmul=2 |  |  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 |  |  |
| compress\_0 | preshift | v2 | ve1c | ve2c |  |  |  |  |  |
| (PSEUDO)gather\_0 |  |  |  | ve1 | ve2 | ve3 |  |  |  |
| compress\_1 |  | preshift | v2 | ve1c | ve2c |  |  |  |  |
| (PSEUDO)gather\_1 |  |  |  |  | ve1 | ve2 | ve3 |  |  |
|  |  |  |  |  |  |  |  |  |  |
| lmul=4 |  |  |  |  |  |  |  |  |  |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| compress\_0 | preshift | v2 | ve1c | ve2c |  |  |  |  |  |
| (PSEUDO)gather\_0 |  |  |  | ve1 | ve2 | ve3 | ve3 | ve3 |  |
| compress\_1 |  | preshift | v2 | ve1c | ve2c |  |  |  |  |
| (PSEUDO)gather\_1 |  |  |  |  | ve1 | ve2 | ve3 | ve3 |  |
| compress\_2 |  |  | preshift | v2 | ve1c | ve2c |  |  |  |
| (PSEUDO)gather\_2 |  |  |  |  |  | ve1 | ve2 | ve3 |  |
| compress\_3 |  |  |  | preshift | v2 | ve1c | ve2c |  |  |
| (PSEUDO)gather\_3 |  |  |  |  |  |  | ve1 | ve2 | ve3 |

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 |
| compress\_0 | preshift | v2 | ve1c | ve2c |  |  |  |  |  |  |  |  |  |
| (PSEUDO)gather\_0 |  |  |  | ve1 | ve2 | ve3 | ve3 | ve3 | ve3 | ve3 | ve3 | ve3 |  |
| compress\_1 |  | preshift | v2 | ve1c | ve2c |  |  |  |  |  |  |  |  |
| (PSEUDO)gather\_1 |  |  |  |  | ve1 | ve2 | ve3 | ve3 | ve3 | ve3 | ve3 | ve3 |  |
| compress\_2 |  |  | preshift | v2 | ve1c | ve2c |  |  |  |  |  |  |  |
| (PSEUDO)gather\_2 |  |  |  |  |  | ve1 | ve2 | ve3 | ve3 | ve3 | ve3 | ve3 |  |
| compress\_3 |  |  |  | preshift | v2 | ve1c | ve2c |  |  |  |  |  |  |
| (PSEUDO)gather\_3 |  |  |  |  |  |  | ve1 | ve2 | ve3 | ve3 | ve3 | ve3 |  |
| compress\_4 |  |  |  |  | preshift | v2 | ve1c | ve2c |  |  |  |  |  |
| (PSEUDO)gather\_4 |  |  |  |  |  |  |  | ve1 | ve2 | ve3 | ve3 | ve3 |  |
| compress\_5 |  |  |  |  |  | preshift | v2 | ve1c | ve2c |  |  |  |  |
| (PSEUDO)gather\_5 |  |  |  |  |  |  |  |  | ve1 | ve2 | ve3 | ve3 |  |
| compress\_6 |  |  |  |  |  |  | preshift | v2 | ve1c | ve2c |  |  |  |
| (PSEUDO)gather\_6 |  |  |  |  |  |  |  |  |  | ve1 | ve2 | ve3 |  |
| compress\_7 |  |  |  |  |  |  |  | preshift | v2 | ve1c | ve2c |  |  |
| (PSEUDO)gather\_7 |  |  |  |  |  |  |  |  |  |  | ve1 | ve2 | ve3 |

## Instruction latency and throughput summary

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Instruction** | **#uop** | **uop  Throughput** | **Instruction Latency LMUL 1/2/4/8** | **Latency considering no forwarding** | **Comment** |
| Register Move | 1 | 1 | 2 | 3 | Independent of LMUL.  Does not include v2 stage. |
| Gather | LMUL^2 | 1 | 2/5/17/65 | 3/6/18/66 | Does not include v2 stage. |
| Compress | LMUL | 1 | 4/6/8/12 | 5/7/9/13 | Does not include v2 stage.  But includes pre-shift stage. |
| Compress | LMUL | 1 | 5/7/9/13 | 6/8/10/14 | Includes pre-shift and hence v2 cycle as well. |
| Slidedown/ Slide1down | LMUL | 1 | 2/4/6/10 | 3/5/7/11 | Does not include v2 stage. |
| Slideup/ Slide1up | LMUL (max) | 1 | 2/3/5/9 | 3/4/6/10 | Does not include v2 stage. The number of uops for slideup may vary from 0 to LMUL(max). This depends on the shift/slide/offset amount/value. The latency numbers are for slide1up and maximum number expected for slideup instruction. |

# Issues

Timing path:

* The compress instruction related switching mux control signal logic is timing critical.
* The switching mux control adjustment for vrgather instruction (which depends on the vsew) when used in ve1 stage will create a timing issue. It is important to adjust the levels of muxing stages across VE1/2.

# Other Considerations

# Future consideration

* Vrgather instruction which is .vx, .vi (not .vv) type may be optimized for the number of uop issued. The number of uop may be restricted to LMUL since there is only one data element to be sourced from the source register. The frontend is expected to read appropriate source register (vs2[x]) for each of the vd[offset] to be updated.
* Make the levels of switching mux spread across ve1/2 stage fully configurable. It should be noted that ve2 stage may not be fully utilized. This does not necessarily do away with ve3 stage. This may possibly enable the forwarding of data from ve2 stage if required.
* EDIV support – affects only the vrgather instruction.
* If for some reason the ISA spec changes to support vd=vs for slideup instruction for lmul>1, the current design will experience a WAR hazard due to the source/destination sequence followed currently. This is not a major change to the permute block. A fix would probably involve an additional cycle of latency for slide[1]up operation from 2 to 3.
* Probably vmv.x.s & vmv.f.s can be converted to 1 cycle latency instruction since the o/p wr port for xrf/frf does not conflict with the vrf write port from permute block. However, vmv.s.x & vmv.s.f can still be a 2 cycle latency instruction since it involves writing to vrf and it may not be worth, arbitrating between the ve1 & ve2 & ve3 stages within the permute block.