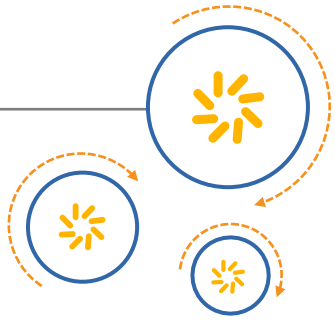
Qualcomm Technologies, Inc.



Qualcomm® Hexagon™ V65 HVX Features for SDM845

80-P9301-84 A

August 3, 2017

Qualcomm Technologies, Inc.
5775 Morehouse Drive
San Diego, CA 92121
U.S.A.

Revision history

Revision	Date	Description
A	August 2017	Initial release

Contents

Figures

Tables

1 Introduction

New V65 HVX features and instructions:

- 256 KB of Vector TCM (VTCM) – For scratch buffers and scatter/gather operations.
- Vector scatter/gather operations – Background movement of sparse elements between vector lanes and VTCM.
- Prefix sum instruction – Helps build address vectors for scatter/gather instructions
- Four-interval piecewise polynomial interpolation – Assists with polynomial approximation of non-linear functions.

V65 HVX also contains enhancements to existing instructions. For details, see the *Hexagon V65 HVX Programmers Reference Manual* (80-N2040-41) in CreatePoint.

2 VTCM

VTCM is used for scratch buffers and scatter/gather operations.

For the SDM845 chipset, the VTCM size is 256 KB. Use VTCM as an intermediate or a temporary buffer. It serves as the input or output of the scatter/gather instructions.

Advantages of using VTCM as the intermediate buffer:

- Guarantees no eviction (vs. L2 if the set is full)
- Faster than L2$ (does not have the overhead of cache management, like association)
- Reduces L2$ pressure
- Lower power than L2$
- Supports continuous read and write for every packet without contention
- Provides up to 50% improvement in the kernel if every packet contains vector read and write (e.g., undistort)

VTCM allocation is managed by the Hexagon Access VTCM Manager (see `incs/HAP_vtcm_mgr.h` in the Hexagon SDK 3.3).

3 Vector scatter/gather operations

3.1 Gather

- Memory-to-memory gather instructions are supported (more precisely, VTCM-to-VTCM operations). Gather operations use slot 0 + slot 1 on the scalar side, and HVX load + store resources.
- Gather is formed by two instructions, one for reading from VTCM and one for storing to VTCM:

```
{ Vtmp.h = vgather(Rt,Mu,Vv.h)
  vmem(Rs+#1) = Vtmp.new
}
```

- If the input data of gather is in DDR, it must first be copied to VTCM and then gathered from there. Gather cannot be done directly on DDR or L2$ contents.
- Vector gather (vgather) operations transfers elemental copies from a large region in VTCM to a smaller vector-sized region in VTCM. Each instruction can gather up to 64 elements. Gather supports halfword and word granularity. Byte gather can be emulated through vector predicate instructions using two packets.
- Gather can be used to do large lookup tables (up to VTCM size).
- Examples that benefit from vgather instructions are timewarp, affine transform, undistortion, bilateral filter, etc.

- `vtmp.h=vgather(Rt,Mu,Vv.h).h`
  - Rt is the input base address in VTCM
  - Mu is the (length-1) of the input region
  - Vv is the byte offset from the base
  - MUST pair with memory write
  - Address (Rt + Vv.h[i]) does **not** need to be aligned by granularity size (halfword)

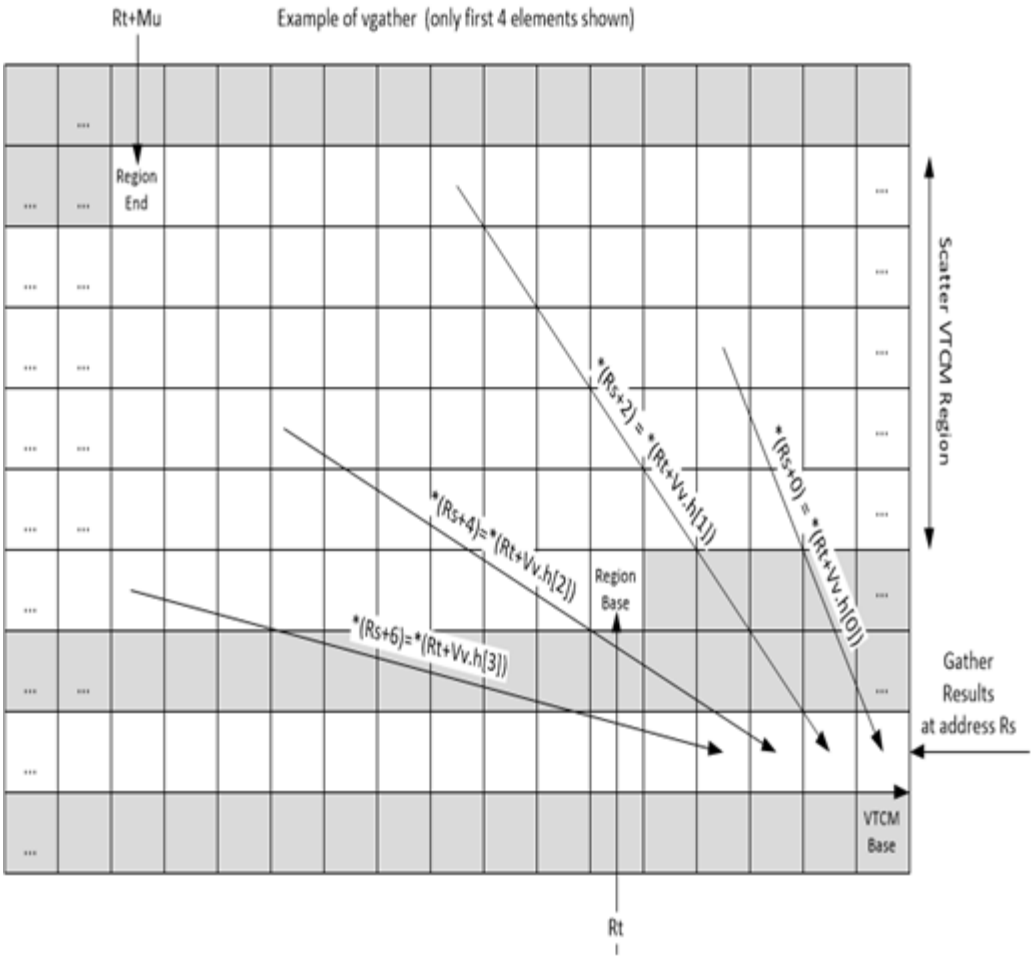


Figure 3-1 vgather example

- The gather region in VTCM must be mapped in a single page (otherwise, an access causes an exception).

The following are scenarios in which a requested element is dropped:

- Address is outside the defined region in VTCM
- Negative offset
- Not accompanied by a store instruction
- Corresponding vector predicate is 0

3.2 Scatter

- Scatter operations copy values from a vector register to a region in VTCM, enabling parallelized data stores to non-contiguous addresses in VTCM.
- Each instruction can scatter up to 64 elements of halfword or word granularity. Byte granularity is emulated through a vector predicate instruction using two packets.
- Post-scatter accumulations are supported. The offset must be aligned by word or halfword (depending on the instruction).
- If multiple values are written to the same memory location, ordering is not guaranteed.
- Vector-to-VTCM scatter instructions are supported:

  ```
  if (Qs4) vscatter(Rt,Mu,Vv.h) = V.h
  ```

- Vector-to-VTCM scatter-and-accumulate instructions are supported, which is useful for histograms:

  ```
  if (Qs4) vscatter(Rt,Mu,Vv.h) += V.h
  ```

- Vector scatter (vscatter) uses slot 0 and HVX store resources,
- Scatters are not L1-coherent, so scalar L1 invalidate operation should be performed for scalar access following scatters.
- Scatter instructions are more efficient than gathers since scatter only accesses VTCM once instead of twice.
- Operations that can be done via scatter or gather usually perform better via scatter.

- `vscatter(Rt,Mu,Vv.h) = V.h`
  - Rt is the input base address in VTCM
  - Mu is the (length-1) of the input region
  - Vv is byte offset from base
  - Address (Rt + Vv.h[i]) does NOT need to be aligned by granularity size (half word)
- Accumulate version: address must be aligned by granularity size.
- Region addresses cannot cross a page boundary (otherwise, an exception occurs)

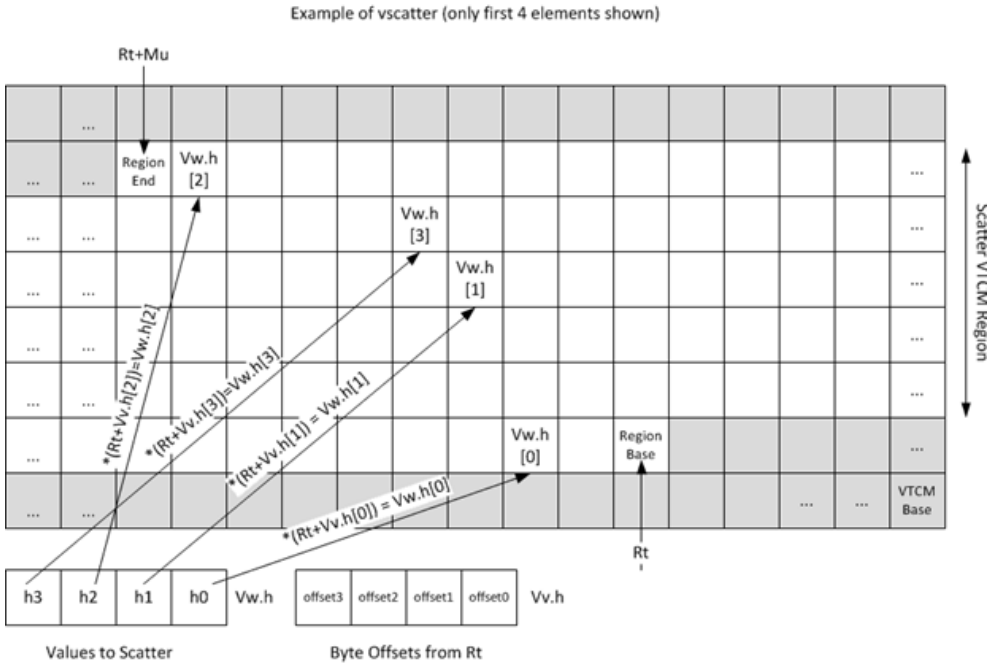


Figure 3-2 vscatter example

Scenarios in which a requested element is dropped:

- Address is outside the defined region in VTCM
- Negative offset
- Corresponding vector predicate is 0

3.3 Scatter/gather memory ordering

Memory is ordered for the following cases:

- A series of VMEM (load/store) instructions
- VMEM (load/store) followed by scatter/gather instruction
- Gather followed by load instruction

For other sequences, the software must follow the store-release and load acquire memory model. Unordered cases include:

- Scatter followed by load/store
- Scatter/gather followed by store

For unordered cases, fence instruction must be inserted to ensure ordering.

- vmem(Rt):scatter_release
- The scatter-release address Rt indicates the address from which a subsequent load must block until the scatter-release completes (it should match the first address that is loaded from the VTCM scatter/gather destination).
- Load after the scatter-release instruction to complete the barrier operation. Everything before the scatter-release is ordered and visible to the loaded data.

3.4 Scatter/gather performance recommendations

- Avoid bank conflicts
  - Repeated addresses cause conflicts (there is no hardware coalescing access)
  - It is best to spread out the distribution of the lower bits of the requested address within a vector scatter/gather, and across subsequent vector scatter/gathers
  - If address bits [7:1] are unique across elements within a vector, the operation will be conflict-free
  - Use a vector predicate to mask out any “don’t care” values
- To maximize bandwidth, avoid concurrent in-flight scatther/gather operations
- The region in VTCM accessed by scatter/gather must be in a single page (a single translation).
- For intra (within a vector) and inter (in-flight) scatter/gather, distributing out these accesses address bits [7:1] can help absorb conflicts within a vector instruction.
- Minimize the density of scatter and gather instructions.
  - Spread these instructions out in a larger loop rather than concentrating them in a tight loop.
  - If multiple scatter/gather in-flight cannot be avoided, limit the bursts to about four for a given thread.
- Defer loading from a gather result or a scatter-release.
  - If the in-flight scatters and gathers (including from other threads) avoid conflicts, a distance of 12 or more packets should be sufficient.
  - Double that distance might be needed if the addresses of in-flight accesses are random.

3.5 Scatter/gather performance

Table 3-1 Peak scatter/gather performance for V65

Operation	Addressing	Vector bandwidth (per packet)	Latency (packets)
Scatter	Conflict-free	2/3	8
Gather	Conflict-free	1/2	12
Scatter	Uniform Random	1/3	16
Gather	Uniform Random	1/6	32

NOTE: A scatter instruction is more efficient than a gather instruction.

3.5.1 9x9 bilateral using gather for 256 entries lookup table

Gather/load operations are 15 packets apart. There are 8 gathers in-flight/20 packets.

Bank conflicts must be mitigated or removed by replicating the 256-entry table across vector lanes.

Table 3-2 9x9 bilateral using gather for 256 entries lookup table

Bilateral	Instruction	Conflict address	Table size	Cycle/pixel
v60	vlut		256	6.5
v65	vgather	0	32768	4.0
v65	vgather	2	16384	4.1
v65	vgather	4	8192	6.3
v65	vgather	8	4096	11.0
v65	vgather	16	2048	20.6
v65	vgather	32	1024	40.3
v65	vgather	64	512	80.4

4 Prefix sum instruction

Use the prefix sum instruction (prefixsum) to compute contiguous offsets for valid positions when processing and consolidating sparse data. This is faster than dynamically computing the permute codeword for vdelta.

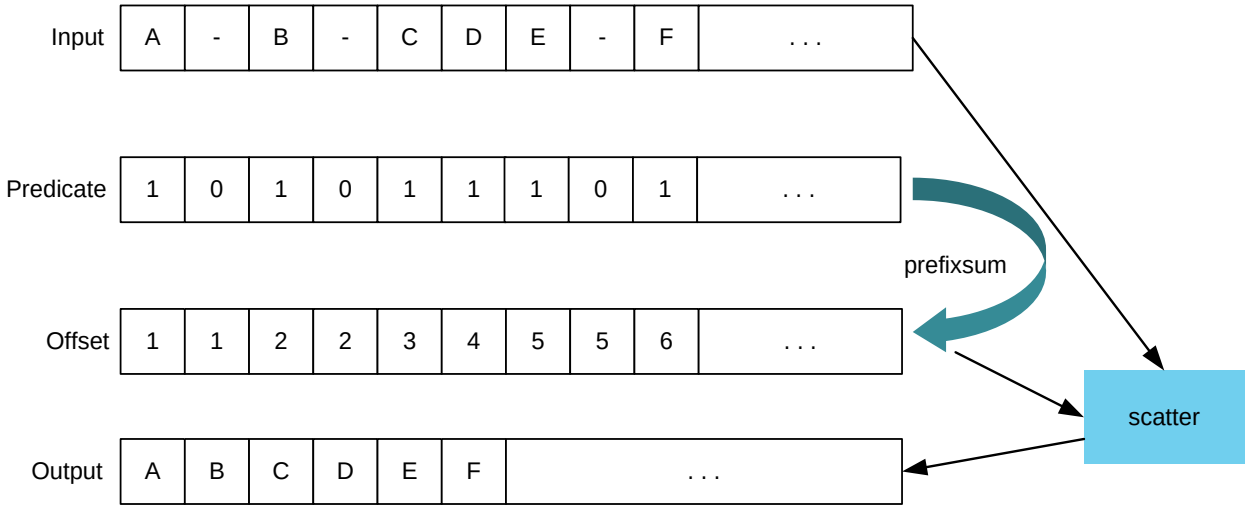


Figure 4-1 prefix sum instruction

5 Four-interval piecewise interpolation instruction

Two new instructions are used to accelerate four-interval piecewise polynomial evaluation for non-linear function estimation.

Instruction VLUT4 is a four-entry lookup table in a scalar register pair (Rtt) used to enable polynomial approximation with four intervals. A new variant of the VMPA instruction is used to compute output value by looking up the base value and computing the offset through the multiplication.

Following is an example of cubic polynomial interpolation with four intervals:

$c_0+c_1*x+c_2*x^2+c_3*x^3 = c_0+(c_1+(c_2+c_3*x)*x)*x$

Where:

c_3 => **VLUT4**

$A = c_2+c_3*x$ => **VMPA**

$B = c_1+A*x$ => **VMPA**

c_0+B*x => **VMPA**

A References

A.1 Related documents

Title	Number
Qualcomm Technologies, Inc.	
Hexagon V65 HVX Programmer's Reference Manual	80-N2040-41

A.2 Acronyms and terms

Acronym or term	Definition
DDR	Double data rate
HVX	Hexagon Vector eXtensions
TCM	Tightly coupled memory
VTCM	Vector TCM