CFGraph RTL Design

By Austin York

Table of Contents

File Hierarchy

tb_top_kernel.sv (sim)
- data_results_kernel.sv (sim)
- kernel.sv (top of kernel)
    - control_s_axi (Control of kernel)
    - GPN.v (Interconnect)
        - PE.v (PE wrapper)
            - PE_IPs.bd (FIFOs and BRAM for cache and tracker)
            - MPU
            - VMU
            - MGU
    - MemoryInterfaces.bd
        - AXI Engine Per PE
        - HBM (16 Channels per stack)
        - DDR (1 Channel per stack)

Kernel (Top)

Parameters:

- Default to kernel creation (**do not alter**)

Ports:

- c0_ddr4 ports for memory controller (**do not alter**, duplicate c1 for second ddr4 stack)
- ap, AXI stream, s_axi_control is part of host I/O

Controls:

- ap_start/start_pulse: Kernel start (host), read AXI stream
- ap_idle: High when not started (host can start kernel)
- ap_done: Kernel/Graph finished, includes writing results to buffer
- GPN_start/start_pulse: GPN start, loading vertices and edges write to HBM and DDR4

Arguments:

- GPNControl: **NOT USED**, will be used for the type of graph algorithm BFS,SSSP, etc. (Only BFS Supported Currently), and any other features to be added.
- MemoryOffset0-16: set to # x1024 vertices between channels, MemoryOffset0 is always 0, MemoryOffset16 is the total amount of x1024 vertices in all channels combined
    - Ex: If MemoryOffset4 = 8 and MemoryOffset3 = 4
      Channel3 = MemoryOffset4 - MemoryOffset3 = 4 x1024 vertices in channel
- TotalEdgeSize: # x512 Edges - 1

Clock Generation:

- The FPGA clock is powered by a DDR differential clock **as recommended**.
- Host clock (ap_clk) set to 250MHz
- FPGA clock (AXI_ACLK_IN_0_buf) set to 300MHz
- HBM Reference clock is set to 100MHz
- HBM APB clock set to 100MHz

S_AXI_Control:

- Controls the host to kernel communication
- Provides arguments to kernel/GPN
- **Recommended to not alter, use Kernel Wizard Creation (Builds package files, replace in old project or add files to new example project)**

Wires/Regs:

- Kernel AXI Engine Native Connections
- PE AXI Engine Native Connections
- Host AXI Engine Native Connections

Load Vertices to HBM:

- Includes an independent clock FIFO 256x512 (250MHz in from host / 300MHz out to FPGA)
- All Writes are burst of 128 x 2 vertices (4096 Bytes)
- Host side
    - Begin reading AXI stream after ap_start_pulse
    - Write into FIFO when data ready
    - Finish when last data read
- FPGA side
    - Begin at GPN_start_pulse
    - Read from FIFO when not empty
    - Change channel to write to based on MemoryOffset0-15
    - Write to HBM, continue until burst is complete
    - Check for write completed confirmation then change address
    - Finish when last data is written, writecounter/4==MemoryOffset16

Load Edges to DDR4:

- Includes an independent clock FIFO 512x512 (250MHz in from host / 300MHz out to FPGA)
- All Writes are burst of 64 x 8 edges (4096 Bytes)
- Host side
    - Begin reading AXI stream after ap_start_pulse
    - Write into FIFO when data ready
    - Finish when last data read
- FPGA side
    - Begin at GPN_start_pulse
    - Read from FIFO when not empty
    - Write to DDR, continue until burst is complete
    - Check for write completed confirmation then change address
    - Finish when last data is written, writecounter==TotalEdgeSize

Read Results from HBM:

- Includes an independent clock FIFO 256x512 (250MHz out to host / 300MHz in from FPGA)
- All Reads are burst of 128 x 2 vertices (4096 Bytes)
- Host side
    - Begin reading from FIFO when graph is done
    - Read whenever not empty and send to host through AXI stream
    - Finish if FIFO empty and FPGA side is finished
- FPGA side
    - Begin reading from HBM when graph is done
    - Start read if not full
    - Change channel to read from based on MemoryOffset0-15
    - Read from HBM and write to FIFO, continue until burst is complete
    - Finish when last data is read, readcounter/4==MemoryOffset16

Data Parity:

- Necessary for memory controllers, only 1 ddr4 write parity needed, need 16 Hbm write parity

HBM AXI Mux:

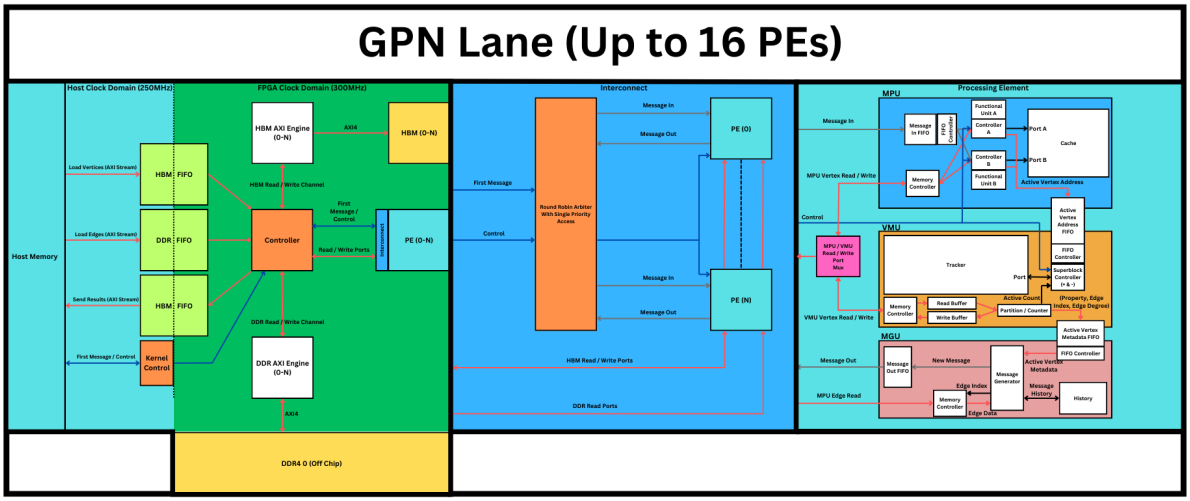
- Give AXI engine read/write access to Host/PEs

Memory Interfaces (VIP)

- Verification of AXI Engine, HBM AXI, and DDR AXI (not true delay)

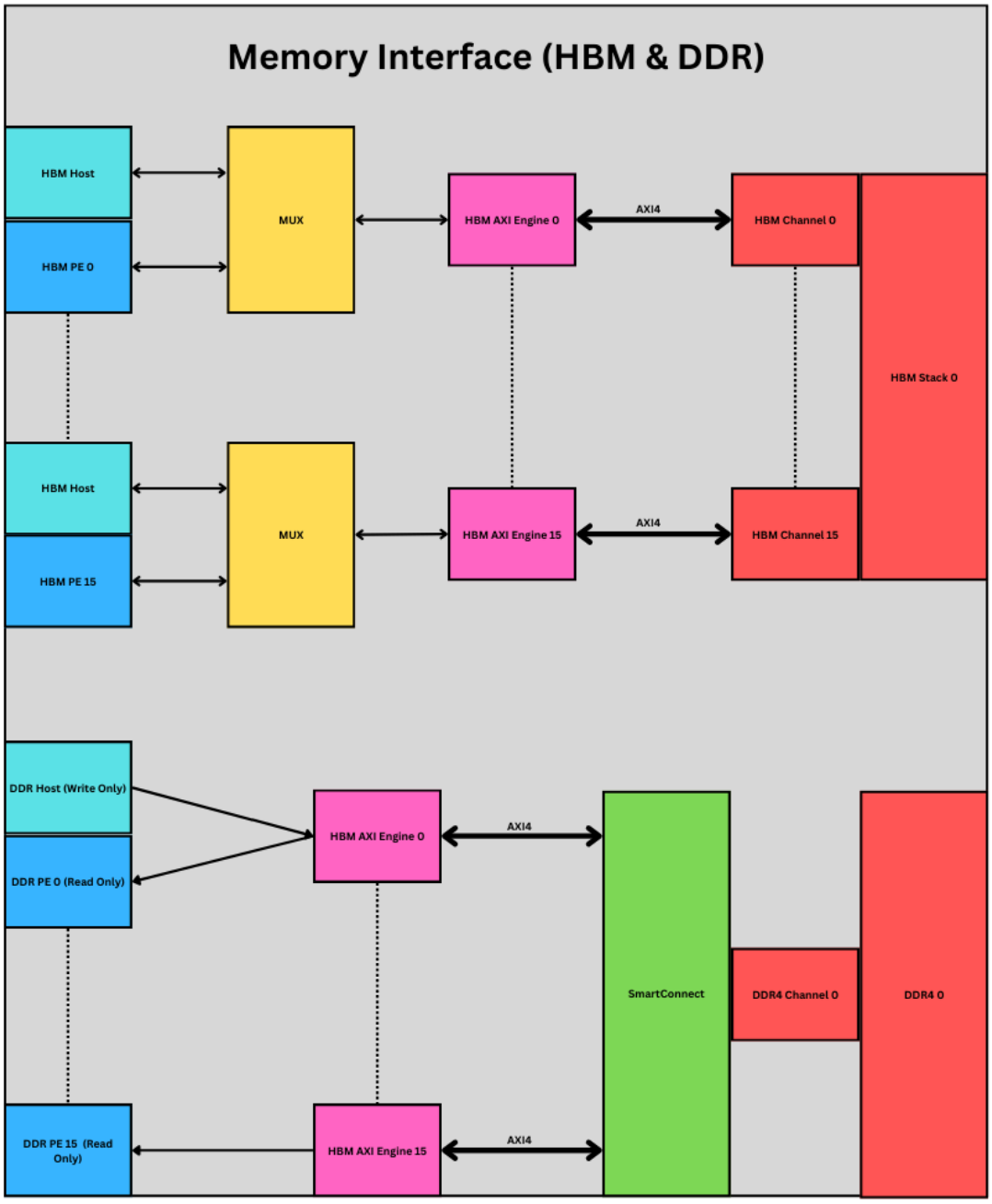
GPN:

- Control and Arguments for GPN/PEs
- AXI Engine native connections for PE0-15



Memory Interface (HBM and DDR)

Facilitate Read/Write from the Host and GPN to the HBM and DDR
HBM: 256 Bits (2 Vertices) Max Burst = 128. Latency ~= 100 cycles
DDR: 512 Bits (8 Edges) Max Burst = 64. Latency ~= 54 cycles



Global Production Network (GPN)

Parameters:

- NUMBER_OF_PEs: # of PEs helps with altering code (**DOES NOT MAKE ANY CONFIGURATION WORK**)
- AXI_CHANNEL_PARTITION: Vertex Memory Offset resolution # Vertices = 2^(x-4)
    - Ex: AXI_CHANNEL_PARTITION = 14, 2^(14-4) = 1024 Vertex resolution
- ADDR_WIDTH: Address width of a message
- UPDATE_WIDTH: Update width of a message
- MESSAGE_WIDTH: Width of a message
- VERTEX_DATAWIDTH: Width of HBM AXI width
- EDGE_DATAWIDTH: Width of DDR AXI width

Ports:

- Start: begin graph computation after vertices and edges loaded
- FirstMessage: Initial vertex you want to compute graph from
- ResetDone: PE reset completed
- GraphDone: Graph computation is completed
- MemoryOffset0-16: Determines where messages get transferred to
- AXI Engine native connections for PE0-15

Wires/Regs:

- regResetDone: Combine ResetDone from each PE
- regResetDone: Combine GraphDone from each PE
- Start_Pulse: Begin graph when ResetDone and start are active
- MaxSuperblockAddress: Caps the tracker reading/checking address to stop reading the whole tracker when not necessary in cases of smaller graphs

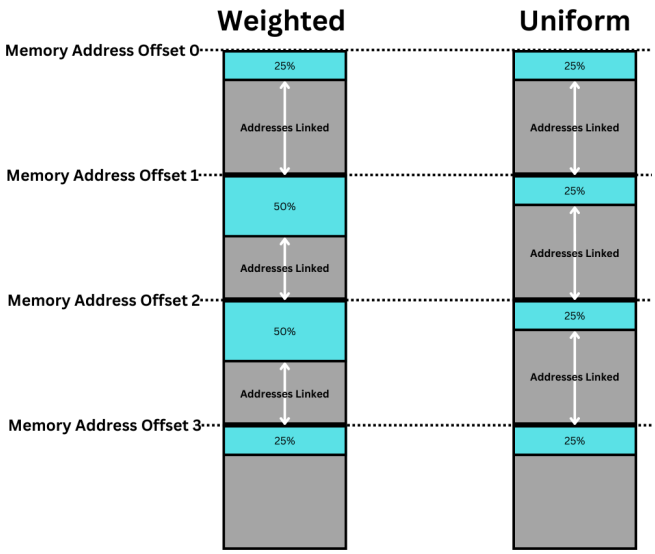
Instantiate PE0-15:

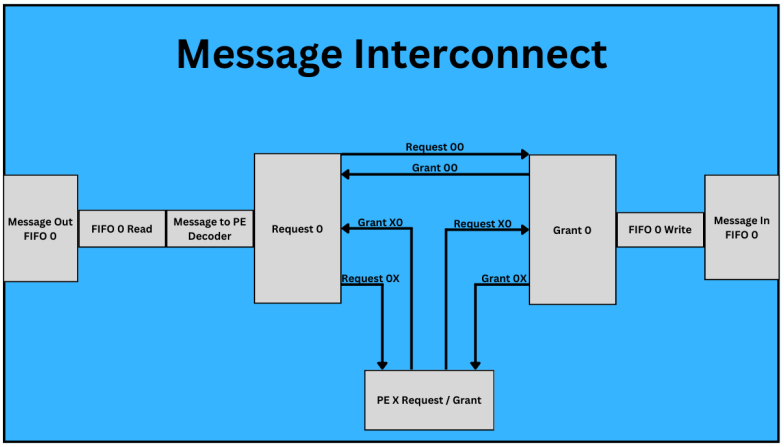
- Corresponding ResetDone, GraphDone, MaxSuperblockAddress, Message I/O FIFO, AXI Engine native connections

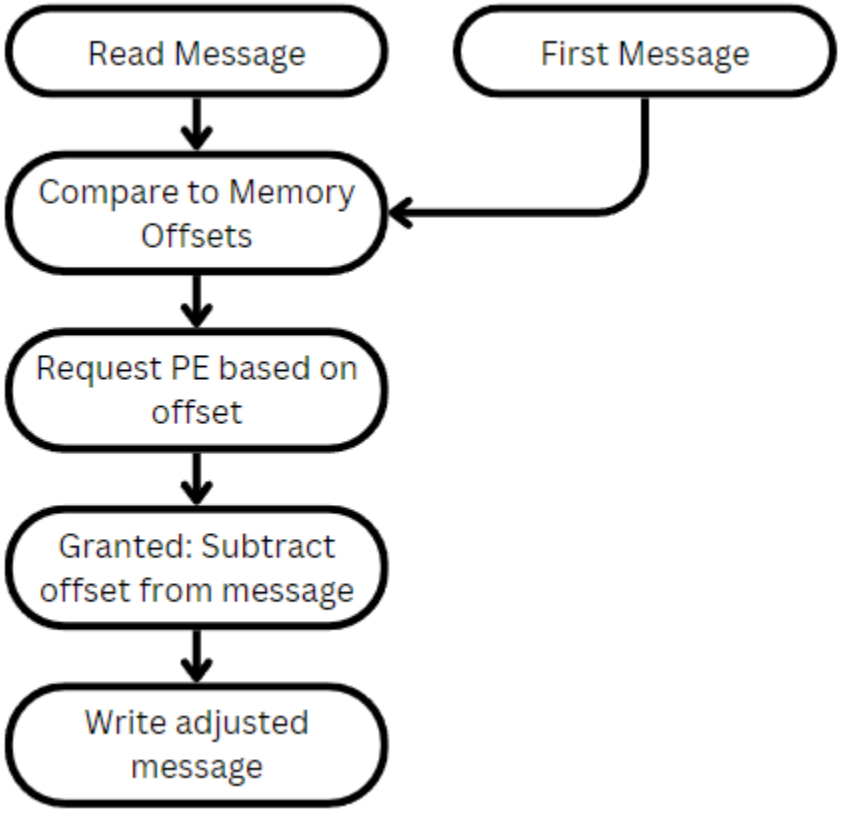
Interconnect

The interconnect connects all Message I/O FIFOs between the PEs. Initially, the first message is sent into the network from the host, which is directed to the corresponding PE based on its address and memory offsets. Once a message is available on the Message Output FIFO, it is read by the PE interconnect controller, and a request is sent to the grant arbiter of the PE it is intended for. The PE to which it needs to hand off the message is determined by the memory offsets loaded at kernel start. If only one request is active on the PEs' grant arbiter, the requesting PE is granted access to the Message In FIFO. It writes an adjusted message based on the address offset of the PE it is sending to. In the case where two or more requests are active, a round-robin protocol is used to provide access. Implementing single-priority access in the grant arbiter is necessary to reduce network latency. Round-robin is utilized to prevent PE starvation and the congestion of Message Output FIFOs.

Documentation Codes
Memory Offsets: 1A
First Message: 2A & 2B
Grant Arbiter: 3A & 3B
Request: 4A
Interconnect State: 5A & 5B







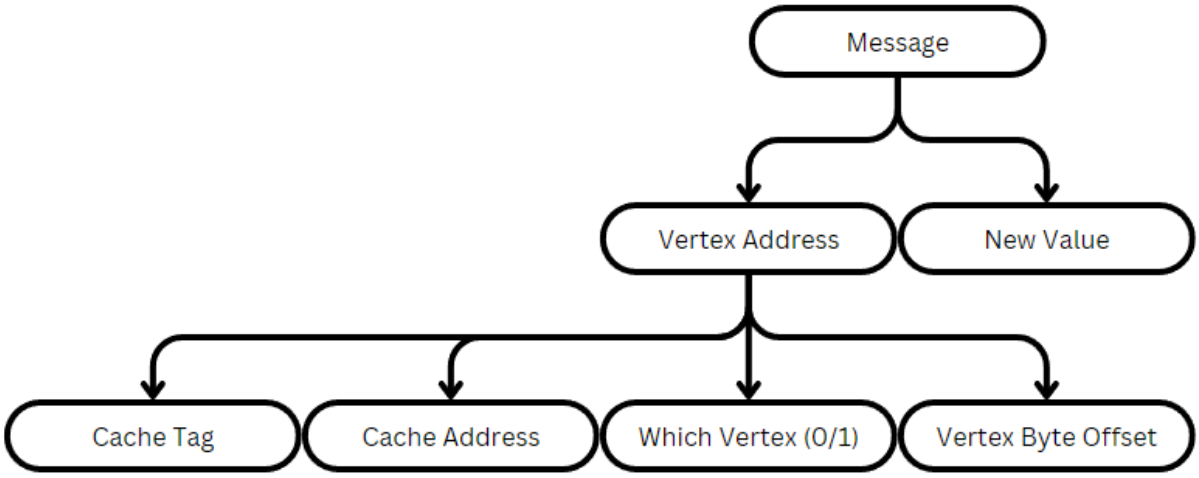
Processing Element (PE)

The PE connects the MPU, VMU, and MGU to their IPs. It facilitates the resetting of the cache, tracker, and work history. When all three units are idle, the GraphDone delay begins (30 cycles). A mux connects the MPU and VMU to the HBM AXI Engine.

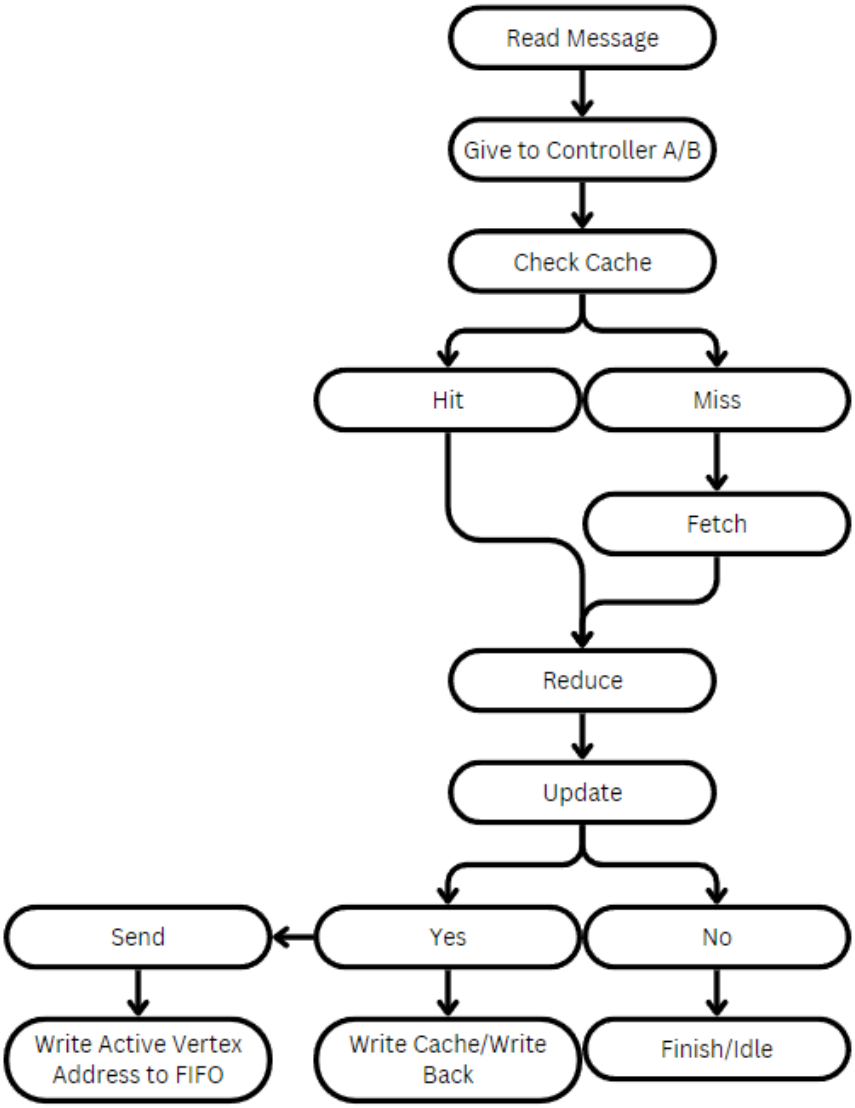
Message Processing Unit (MPU)

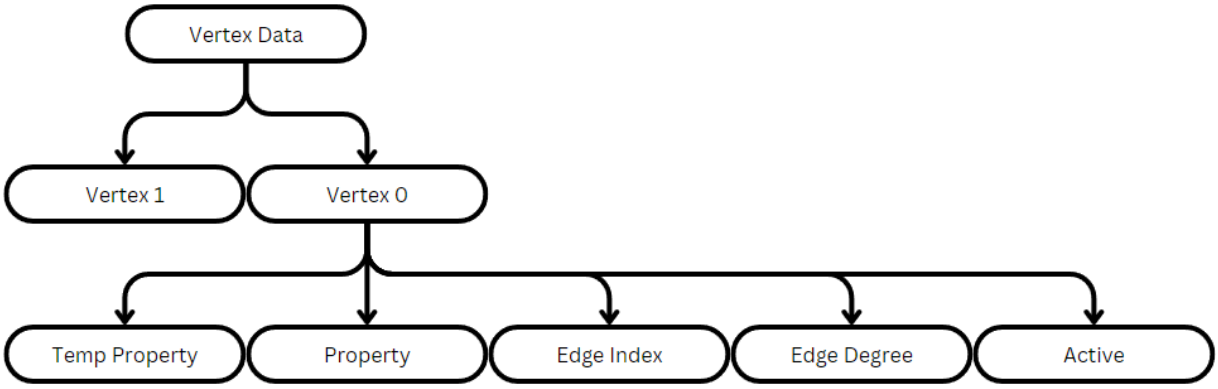
The MPU receives a message, and the FIFO controller retrieves it along with the addresses of cache controllers A and B. It then hands off the message to whichever controller is idle, giving priority to controller A if there's a matching address. Having the additional check performed by the FIFO controller eliminates the need for read/write collision safety within the cache, which would otherwise increase power usage. Once the message handoff is completed, all corresponding values are calculated based on the message. There's a one-cycle delay for the cache data line to become ready for reading. The cache data line is then read, and it's determined whether a vertex hit or miss has occurred. In case of a cache miss, a request is sent

to the memory controller to fetch the vertex. The memory controller prioritizes writebacks A > B and fetches A > B.



Upon completion of a fetch or a hit, the vertex is updated based on the graph algorithm, currently supporting BFS and SSSP. Each cache controller has its own functional units, enabling both to operate in parallel. If a vertex is updated, it's written to the cache and subsequently to the HBM. To track the vertex, the edge degree must be greater than 0, indicating that the vertex has edges needing propagation with the new value. The active bit value is then switched from 0 to 1, informing the VMU which vertices need to be sent for generation. If controller B needs to write to the active vertex FIFO simultaneously, it's delayed by one cycle to allow controller A to write its value first. Finally, a writeback is scheduled with the memory controller to ensure vertex consistency with the cache. The MPU determines when the VMU starts checking its tracker once all available messages and active vertices have been processed. Maintaining consistency between the cache and HBM is crucial. If the VMU encounters an active vertex, the controllers read the cache based on the superblock address and deactivate any active vertices matching the cache tag. The controller addresses are interleaved to enable parallel cache probing. The cache utilizes URAM (Block SRAM), reducing latency to one cycle after setting the address. The sizes of URAM are 72x512, with two vertices including cache tags, valid, and dirty bits using 288x512 (4 URAM) for 512 lines of cache. The current design employs 8 URAM (36KB) for a 1024 cache depth.

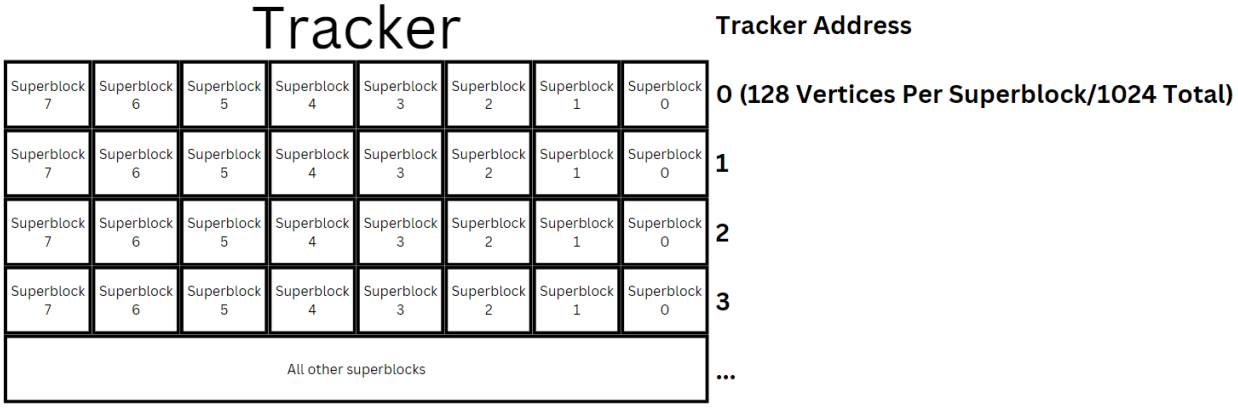

Read Message
Give to Controller A/B
Check Cache
Hit
Miss
Fetch
Reduce
Update
Send
Yes
No
Write Active Vertex Address to FIFO
Write Cache/Write Back
Finish/Idle


Vertex Data
Vertex 1
Vertex 0
Temp Property
Property
Edge Index
Edge Degree
Active

Documentation Codes
FIFO Controller: 1A & 1B
Cache Controller A: 2A, 2B, & 2C
Cache Controller B: 3A, 3B, & 3C
Fetch/Writeback: 4A
Active Vertex FIFO: 5A
Functional Units/Reduction Engine: 6A

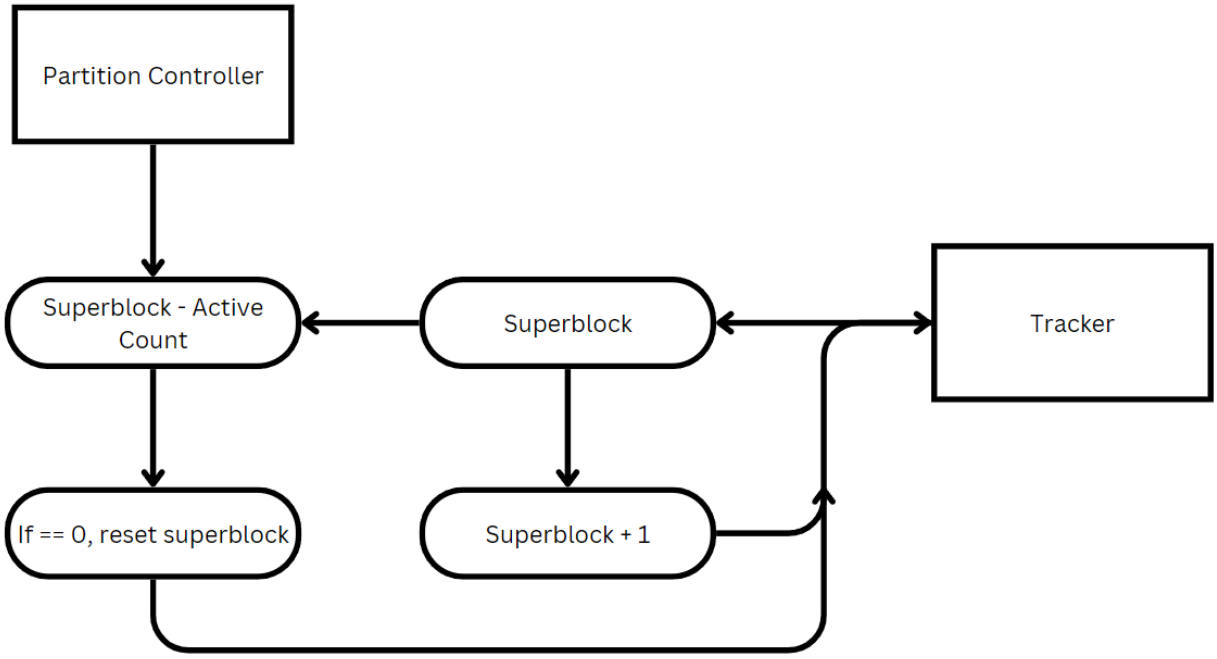
Vertex Management Unit (VMU)

The VMU is used to track the number of active vertices in the HBM by combining the memory blocks into superblocks to reduce the necessary tracking size. Depending on the amount of SRAM available, you can scale the resolution up by combining more blocks into a superblock. To track 256MB of vertex data with a superblock resolution of 128 blocks, 32 URAM is needed, which is 72x16384 (148KB). Not all graphs use the full 256MB channel, so a Max Superblock Address is calculated from the memory offsets given at kernel start. For example, when only 128MB is used, the Max Superblock Address (with a max depth of 16384) is halved to 8192, which saves 3 cycles per line.



When an active vertex is available, the tracker will read the superblock dedicated to the vertex address, increment it by 1, and label the tracker as active. If the MPU is inactive and no more vertices need to be tracked, the VMU starts checking the superblocks one by one. This process is bulk synchronous, ensuring consistency between the MPU cache, HBM, and VMU tracker. The tracker will read 8 superblocks and determine if any active vertices need to be read, starting at superblock 0. If none are active, it moves onto the next superblock until all 8 are checked. Once a superblock is non-zero, a read to HBM is initiated to read 32 vertices. While the vertex data is being read, it is placed into a read buffer, where the partition controller will record the number of active vertices and send the active vertex metadata to the MGU for message generation. Vertices are placed into the write buffer with any active vertices

deactivated. At the same time, a probe checks the active bits to see if any active vertices are read and notifies the MPU if it needs to resolve any conflicts in its cache. If no active vertices are received, the buffers are reset, the write back is skipped, and the next 32 vertices are read. Once partitioning is completed and at least one active vertex was hit, the superblock value is decremented by the number of active blocks detected by the partition counter, then read from the write buffer, and the now deactivated vertices are written back.

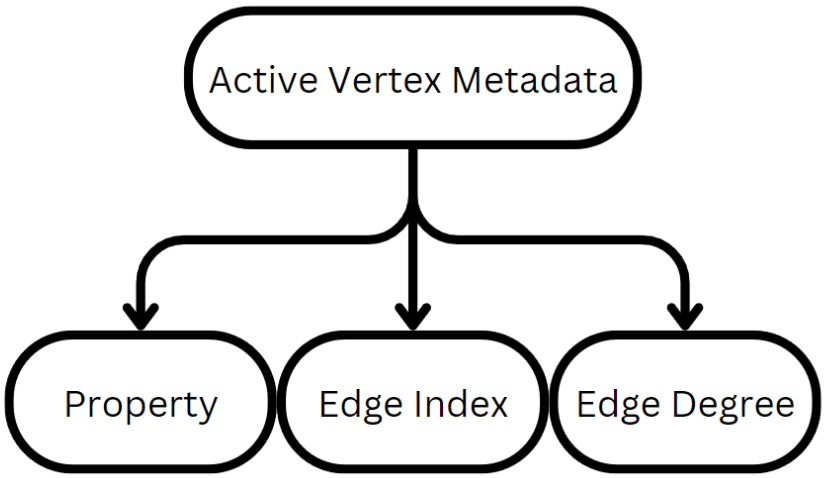


The process continues reading the next burst until no more active vertices are left in the superblock, then it moves onto the next superblock. After every 8 superblocks, the next tracker address is read to get the next 8. At this step, it is necessary to check if the max superblock address has been reached and deactivate the tracker once the tracker has been cleared of all active superblocks. In case the active vertex metadata FIFO is full, checking stops and resumes only when there are no new active vertices and the active vertex metadata FIFO is not full anymore.The read and write buffers are sleeping until a memory transaction which reduces power usage.

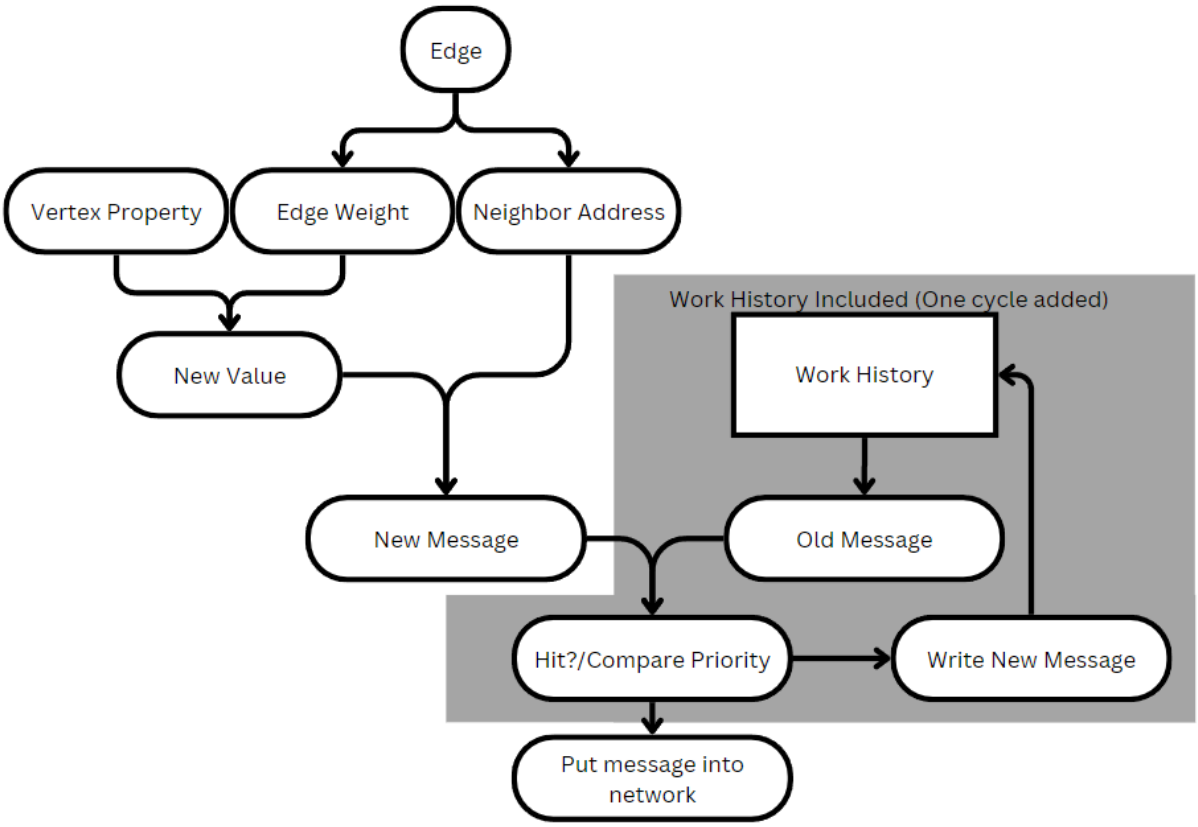
Documentation Codes
Tracker: 1A, 1B, 1C, 1D
Increment: 2A, 2B, 2C
Check Tracker: 3A, 3B
Read: 4A, 4B, 4C
Decrement: 5A, 5B
Write: 6A
Partition Controller: 7A

Message Generation Unit (MGU)

The MGU takes the active vertex metadata and breaks it down into the property, edge index, and edge degree. The edge index represents the memory address in the DDR, while the edge degree indicates how many edges need to be read and propagated. Additionally, the edge index determines the initial offset of the edge within the read, thus establishing the starting offset of the edge within the set of 8 edges. The new value for the message is computed from the vertex property and the edge weight, with the vertex address being the neighboring address from the edge data. If the number of computed edges isn't equal to the edge degree, the process continues, generating new messages for each edge until all edges have been processed and new messages generated.



A work history is employed for previously sent messages to determine whether to dispatch the generated message. This check adds one cycle to the generation process. However, when a message with higher priority (smaller value in BFS) has already been sent, the generated message isn't transmitted into the network. This significantly reduces the number of messages injected into the system, freeing up the interconnect and reducing processing requirements in the MPU. If the message in the work history doesn't match the tag, has lower priority, or is invalid, the new message takes its place. The work history occupies only two URAM (72x1024), which is ¼ the size of the MPU cache. It can be expanded for additional algorithms and its size can be adjusted based on available SRAM.



Message Generation w/ Work History

Documentation Codes
Setup/Read: 1A, 1B, & 1C
Message Generation: 2A, 2B
Work History: 3A