Cube v2 Architecture Specification

Matrix Multiplication Accelerator with MATMUL Block Instruction Support

1. Overview

Cube v2 is a matrix multiplication accelerator that supports large matrix operations through tiled computation. It receives MATMUL block instructions and decomposes them into micro-operations (uops) that can be executed on the 16×16 systolic array.

1.1 Key Features

• **MATMUL Block Instruction**: Supports A[M×K] × B[K×N] = C[M×N] operations

• **Tiled Computation**: Large matrices decomposed into 16×16 uops

• **Out-of-Order Execution**: 64-entry issue queue for uop scheduling

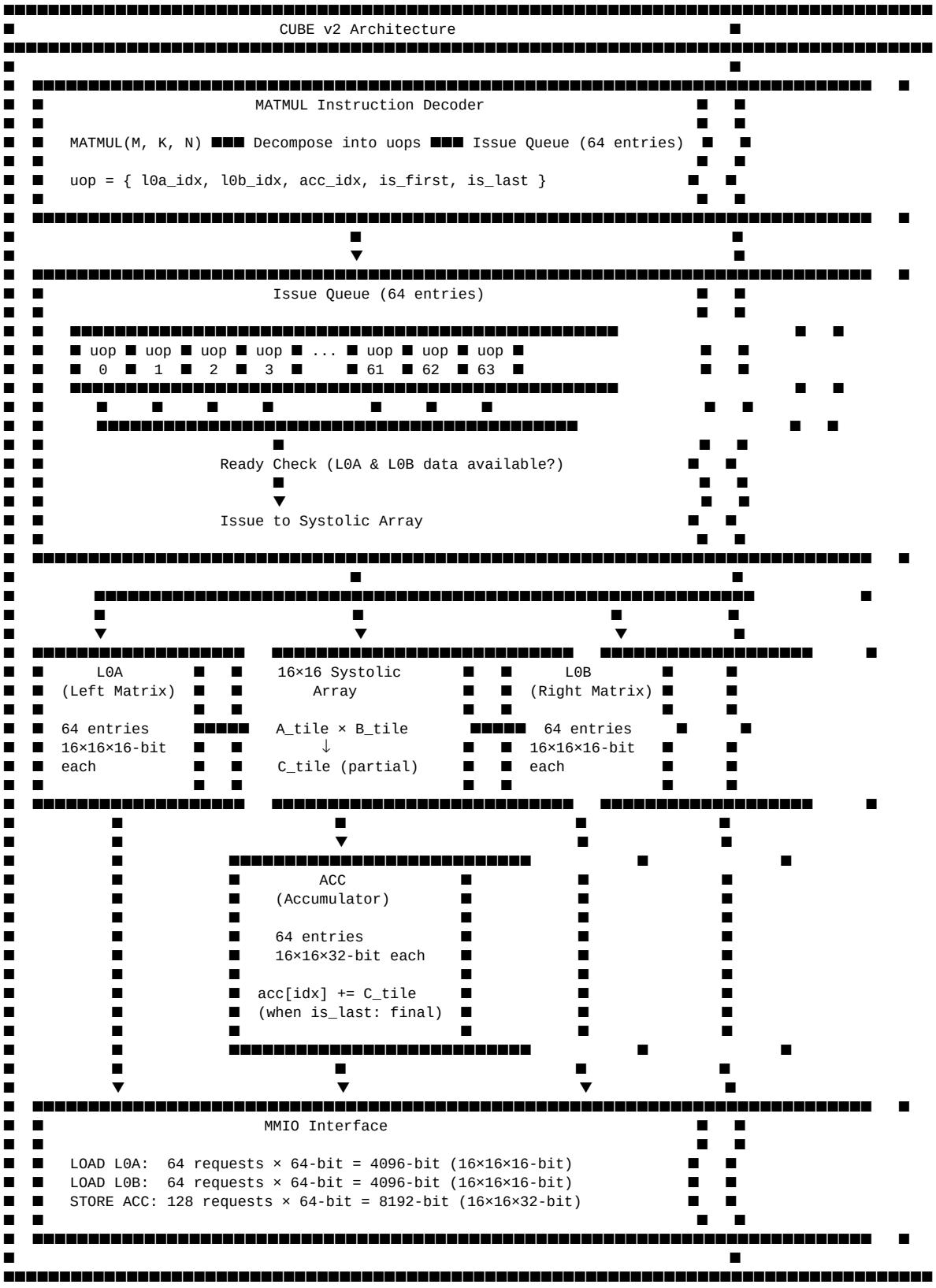
• **Triple Buffering**: L0A (left matrix), L0B (right matrix), ACC (accumulator)

• **64-bit MMIO**: Simplified interface for C++ emitter compatibility

1.2 Specifications Summary

Parameter	Value
Systolic Array Size	16 × 16 (256 PEs)
PE Clusters	4 clusters (4-stage pipeline)
PEs per Cluster	16 × 4 = 64 PEs
Throughput	1 uop/cycle (after pipeline fill)
Pipeline Latency	4 cycles
L0A Buffer	64 entries × 16×16 × 16-bit
L0B Buffer	64 entries × 16×16 × 16-bit
ACC Buffer	64 entries × 16×16 × 32-bit
Issue Queue	64 entries
Data Width (Input)	16-bit
Data Width (Output)	32-bit
MMIO Read Bandwidth	64-bit/cycle
MMIO Write Bandwidth	64-bit/cycle

2. Architecture Block Diagram

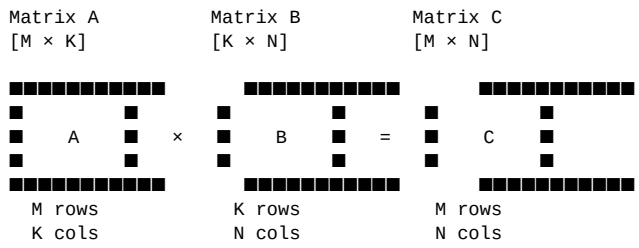


3. MATMUL Instruction Format

3.1 Instruction Fields

```
MATMUL M, K, N, addr_A, addr_B, addr_C

Fields:
  M      : Number of rows in left matrix A (and result C)
  K      : Number of columns in A / rows in B (reduction dimension)
  N      : Number of columns in right matrix B (and result C)
  addr_A : Base address of matrix A in memory
  addr_B : Base address of matrix B in memory
  addr_C : Base address of result matrix C in memory
```

3.2 Matrix Dimensions



4. Uop Decomposition

4.1 Tiling Strategy

Large matrices are decomposed into 16×16 tiles for processing on the systolic array.

```
Number of tiles:
  M_tiles = ceil(M / 16)
  K_tiles = ceil(K / 16)
  N_tiles = ceil(N / 16)

Total uops = M_tiles × K_tiles × N_tiles
```

4.2 Uop Structure

```
struct Uop {
    uint8_t  l0a_idx;     // Index into L0A buffer (0-63)
    uint8_t  l0b_idx;     // Index into L0B buffer (0-63)
    uint8_t  acc_idx;     // Index into ACC buffer (0-63)
    bool     is_first;    // First uop for this ACC entry (clear accumulator)
    bool     is_last;     // Last uop for this ACC entry (result ready)
    uint8_t  m_tile;      // Tile index in M dimension
    uint8_t  k_tile;      // Tile index in K dimension
    uint8_t  n_tile;      // Tile index in N dimension
};
```

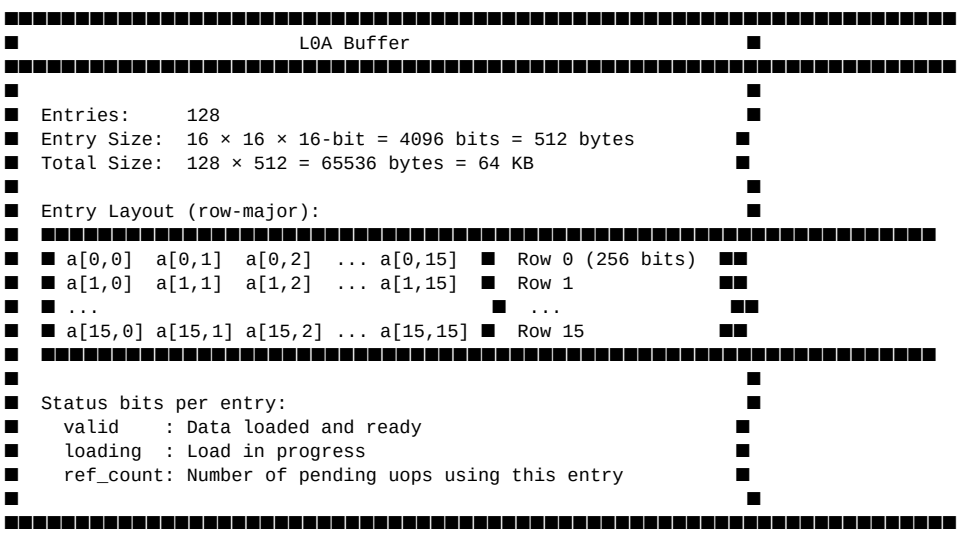
4.3 Uop Generation Example

For MATMUL with M=32, K=48, N=32:

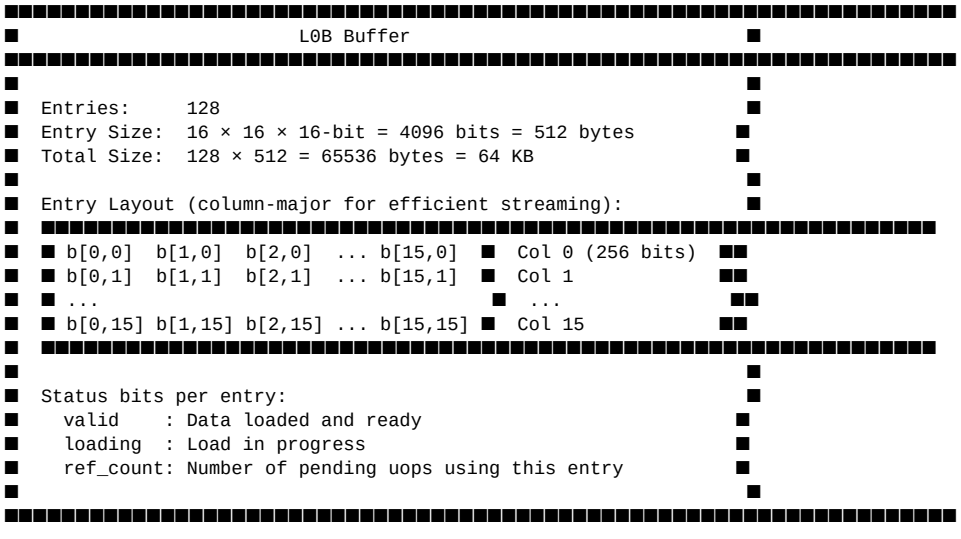
```
M_tiles = ceil(32/16) = 2
K_tiles = ceil(48/16) = 3
N_tiles = ceil(32/16) = 2

Total uops = 2 × 3 × 2 = 12 uops

Uop sequence (for one output tile C[0,0]):
  uop0: A[0,0] × B[0,0] → ACC[0] (is_first=1, is_last=0)
  uop1: A[0,1] × B[1,0] → ACC[0] (is_first=0, is_last=0)
  uop2: A[0,2] × B[2,0] → ACC[0] (is_first=0, is_last=1)

  C[0,0] = A[0,0]×B[0,0] + A[0,1]×B[1,0] + A[0,2]×B[2,0]
```

5. Buffer Specifications

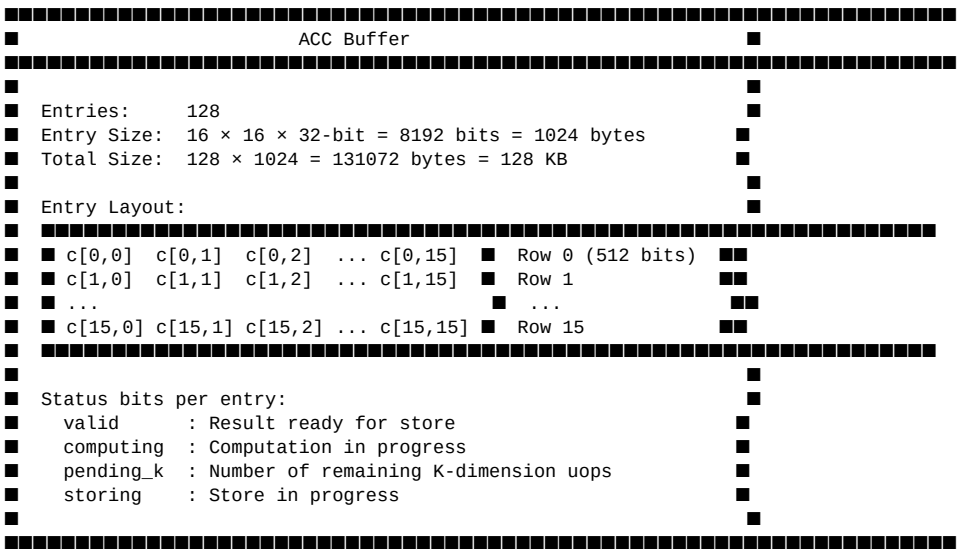
5.1 L0A Buffer (Left Matrix Input)



5.2 L0B Buffer (Right Matrix Input)

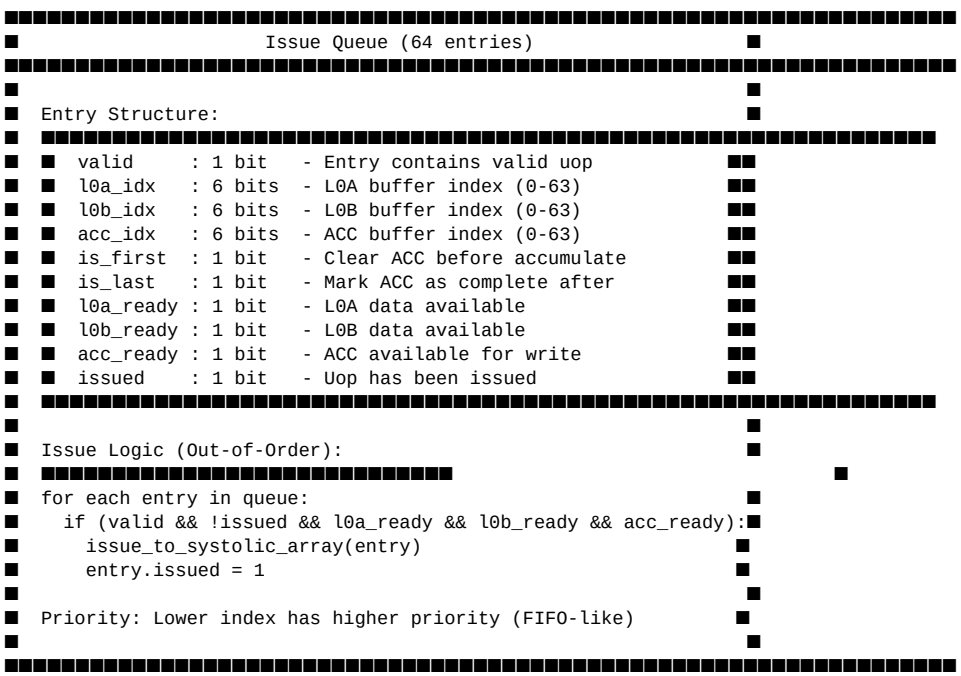


5.3 ACC Buffer (Accumulator Output)



6. Issue Queue

6.1 Structure



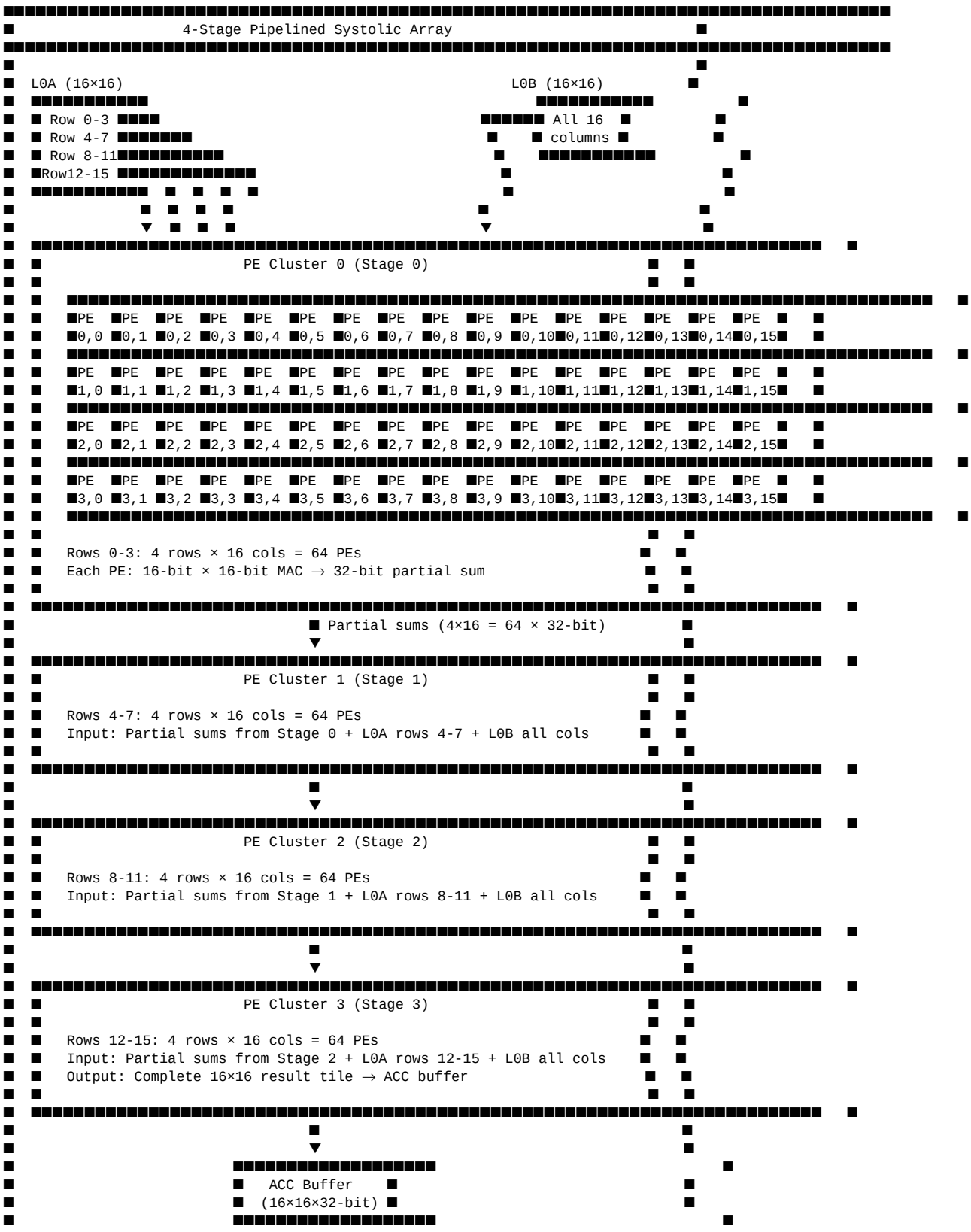
6.2 Ready Check Logic

```
l0a_ready = L0A[uop.l0a_idx].valid
l0b_ready = L0B[uop.l0b_idx].valid
acc_ready = !ACC[uop.acc_idx].computing ||
            (ACC[uop.acc_idx].computing && current_uop_for_acc_done)
```

7. Pipelined Systolic Array Architecture

7.1 4-Stage PE Cluster Pipeline

The systolic array is organized as 4 PE Clusters in a pipeline configuration, enabling 1 uop/cycle throughput after initial pipeline fill.



```
■                                                                               ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

7.2 Pipeline Timing

```
Cycle:    0     1     2     3     4     5     6     7     8    ...
          ■     ■     ■     ■     ■     ■     ■     ■     ■
          ▼     ▼     ▼     ▼     ▼     ▼     ▼     ▼     ▼

uop0:   [C0]■■[C1]■■[C2]■■[C3]■■■ACC
uop1:         [C0]■■[C1]■■[C2]■■[C3]■■■ACC
uop2:               [C0]■■[C1]■■[C2]■■[C3]■■■ACC
uop3:                     [C0]■■[C1]■■[C2]■■[C3]■■■ACC
uop4:                           [C0]■■[C1]■■[C2]■■[C3]■■■ACC
...

Legend:
  [C0] = PE Cluster 0 processing
  [C1] = PE Cluster 1 processing
  [C2] = PE Cluster 2 processing
  [C3] = PE Cluster 3 processing
  ■ACC = Write to ACC buffer

Pipeline characteristics:
  - Latency: 4 cycles (from uop issue to ACC write)
  - Throughput: 1 uop/cycle (after pipeline fill)
  - In-flight uops: Up to 4 (one per stage)
```

7.3 PE Cluster Data Flow

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                          PE Cluster Internal Structure                         ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                                                                               ■
■  Inputs per cycle:                                                            ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■  ■  From L0A: 4 rows × 16 elements × 16-bit = 1024 bits                ■   ■
■  ■  From L0B: 16 cols × 16 elements × 16-bit = 4096 bits               ■   ■
■  ■  From prev cluster: 4 rows × 16 cols × 32-bit = 2048 bits (partial)■   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■                                                                               ■
■  Computation per cycle (64 PEs):                                              ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■  ■  For each PE[row][col] (row in 0..3, col in 0..15):                  ■   ■
■  ■                                                                       ■   ■
■  ■    // Each PE computes dot product of A row segment × B column       ■   ■
■  ■    partial_sum = 0                                                    ■   ■
■  ■    for k in 0..15:                                                    ■   ■
■  ■      partial_sum += A[row][k] × B[k][col]                            ■   ■
■  ■                                                                       ■   ■
■  ■    // Add incoming partial sum from previous cluster                 ■   ■
■  ■    result[row][col] = partial_sum + prev_partial[row][col]           ■   ■
■  ■                                                                       ■   ■
■  ■  Total MACs per cycle: 64 PEs × 16 MACs = 1024 MACs                  ■   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■                                                                               ■
■  Output per cycle:                                                            ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■  ■  To next cluster: 4 rows × 16 cols × 32-bit = 2048 bits (partial)  ■   ■
■  ■  (Cluster 3 outputs to ACC buffer instead)                          ■   ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■                                                                               ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

7.4 Pipeline Registers

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                          Pipeline Register Structure                           ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                                                                               ■
■  Each pipeline stage has registers to hold:                                   ■
■                                                                               ■
```

```
■  struct PipelineReg {                                                         ■
■      valid: 1 bit           // Stage has valid data                         ■
■      uop_id: 6 bits         // Which uop is in this stage                   ■
■      acc_idx: 7 bits        // Target ACC buffer index                      ■
■      is_first: 1 bit        // Clear ACC before write                       ■
■      is_last: 1 bit         // Mark ACC complete after write                ■
■      partial_sums: 64×32 bits  // 4 rows × 16 cols partial results         ■
■  };                                                                          ■
■                                                                              ■
■  Pipeline advancement (every cycle):                                         ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■  ■  pipe_reg[3] → ACC buffer (if valid)                                ■    ■
■  ■  pipe_reg[2] → pipe_reg[3]                                          ■    ■
■  ■  pipe_reg[1] → pipe_reg[2]                                          ■    ■
■  ■  pipe_reg[0] → pipe_reg[1]                                          ■    ■
■  ■  new_uop    → pipe_reg[0] (if issue_valid)                          ■    ■
■  ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■   ■
■                                                                              ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

7.5 Throughput Analysis

```
Performance metrics:

  Single uop computation:
    - 16×16 tile = 256 output elements
    - Each element requires 16 MACs (K=16)
    - Total: 256 × 16 = 4096 MACs per uop

  Pipeline throughput:
    - 4 clusters × 64 PEs × 16 MACs = 4096 MACs per cycle
    - 1 uop completes per cycle (after 4-cycle fill)

  Example: MATMUL(64, 64, 64)
    - Tiles: 4 × 4 × 4 = 64 uops
    - Pipeline fill: 4 cycles
    - Steady state: 64 uops × 1 cycle = 64 cycles
    - Total: 4 + 64 = 68 cycles
    - Throughput: 64 × 4096 MACs / 68 cycles = 3855 MACs/cycle

  Peak throughput:
    - 4096 MACs/cycle (after pipeline fill)
    - At 1 GHz: 4.096 TMAC/s (INT16)
```

8. MMIO Interface

8.1 Bandwidth Calculations

```
MMIO Bandwidth: 64 bits per cycle (read and write)
(Simplified from 2048-bit for C++ emitter compatibility)

L0A/L0B Entry Size: 16 × 16 × 16-bit = 4096 bits
  → LOAD requires 64 cycles (64 × 64 bits)

ACC Entry Size: 16 × 16 × 32-bit = 8192 bits
  → STORE requires 128 cycles (128 × 64 bits)
```

8.2 Memory Map

```
Base Address: 0x80000000
```

Offset	Size	Description
0x0000	8B	Control Register
0x0008	8B	Status Register
0x0010	8B	MATMUL Instruction Register (M,K,N)

```
■ 0x0018      ■ 8B      ■ Address A Register                    ■
■ 0x0020      ■ 8B      ■ Address B Register                    ■
■ 0x0028      ■ 8B      ■ Address C Register                    ■
■ 0x0030      ■ 8B      ■ Load L0A Command (entry_idx, addr)    ■
■ 0x0038      ■ 8B      ■ Load L0B Command (entry_idx, addr)    ■
■ 0x0040      ■ 8B      ■ Store ACC Command (entry_idx, addr)   ■
■ 0x0048      ■ 8B      ■ Queue Status (entries used/free)      ■
■ 0x0050      ■ 8B      ■ L0A Status (bitmap of valid entries)  ■
■ 0x0058      ■ 8B      ■ L0B Status (bitmap of valid entries)  ■
■ 0x0060      ■ 8B      ■ ACC Status (bitmap of ready entries)  ■
■             ■         ■                                       ■
■ 0x1000      ■ 8B      ■ L0A Data Port (64-bit)                ■
■ 0x2000      ■ 8B      ■ L0B Data Port (64-bit)                ■
■ 0x3000      ■ 8B      ■ ACC Data Port (64-bit)                ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

8.3 Control Register Bits

```
Control Register (0x0000):
  Bit 0    : START      - Start MATMUL execution
  Bit 1    : RESET      - Reset accelerator
  Bit 2    : LOAD_L0A   - Trigger L0A load
  Bit 3    : LOAD_L0B   - Trigger L0B load
  Bit 4    : STORE_ACC  - Trigger ACC store
  Bit 7:5  : Reserved
  Bit 15:8 : Entry index for LOAD/STORE operations
  Bit 63:16: Reserved
```

8.4 Status Register Bits

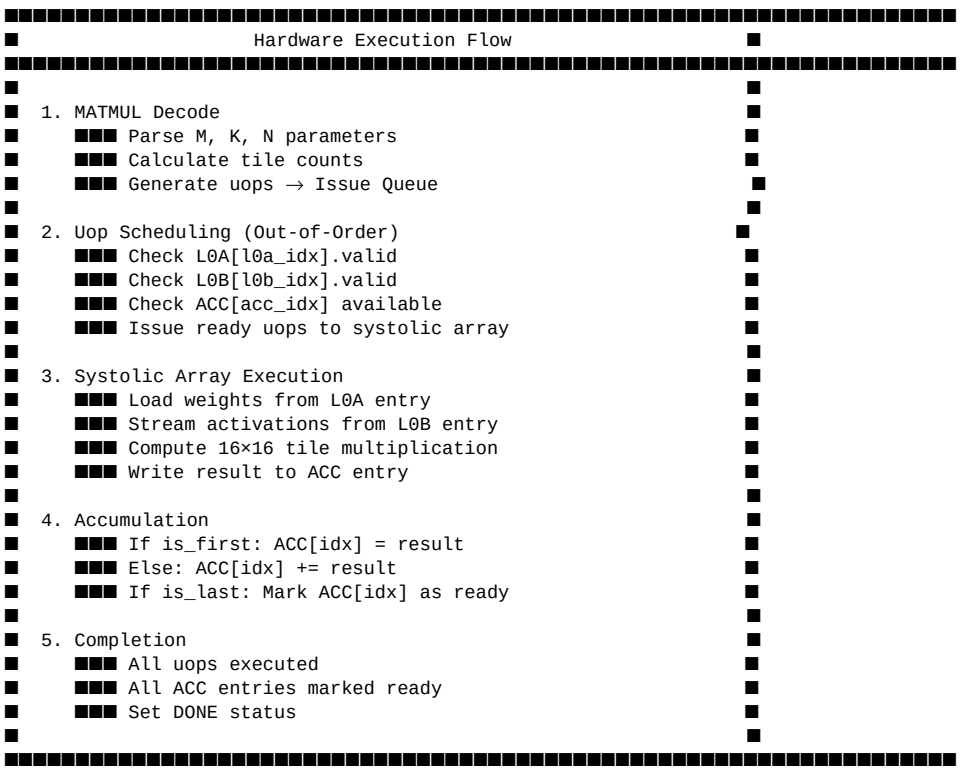
```
Status Register (0x0008):
  Bit 0    : DONE       - MATMUL complete
  Bit 1    : BUSY       - Computation in progress
  Bit 2    : L0A_BUSY   - L0A load in progress
  Bit 3    : L0B_BUSY   - L0B load in progress
  Bit 4    : ACC_BUSY   - ACC store in progress
  Bit 5    : QUEUE_FULL - Issue queue full
  Bit 6    : QUEUE_EMPTY- Issue queue empty
  Bit 15:7 : Reserved
  Bit 23:16: Queue entries used
  Bit 31:24: Queue entries free
  Bit 63:32: Cycle counter
```

9. Execution Flow

9.1 Software Flow

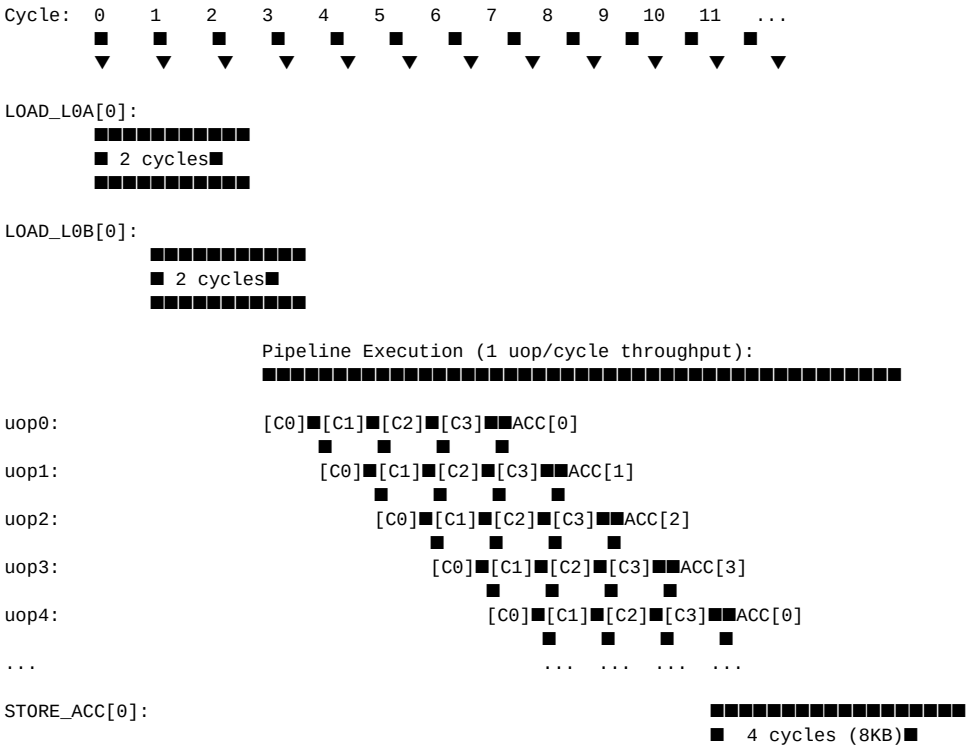
```
1. Configure MATMUL parameters:
   - Write M, K, N to MATMUL Instruction Register
   - Write addr_A, addr_B, addr_C to Address Registers

2. Pre-load input tiles:
   - For each required A tile: LOAD_L0A(entry_idx, tile_addr)
   - For each required B tile: LOAD_L0B(entry_idx, tile_addr)
   - Wait for loads to complete (poll status)

3. Start computation:
   - Write START to Control Register
   - Hardware decomposes MATMUL into uops
   - Uops execute as inputs become ready

4. Store results:
   - Poll ACC status for completed entries
   - For each ready ACC entry: STORE_ACC(entry_idx, dest_addr)

5. Repeat for next batch of tiles (if matrix larger than buffer)
```

9.2 Hardware Flow



10. Pipelined Execution Timing

10.1 Detailed Pipeline Timing



10. Pipelined Execution Timing

10.1 Detailed Pipeline Timing

```
Cycle:  0    1    2    3    4    5    6    7    8    9   10   11   ...
        ■    ■    ■    ■    ■    ■    ■    ■    ■    ■    ■    ■
        ▼    ▼    ▼    ▼    ▼    ▼    ▼    ▼    ▼    ▼    ▼    ▼

LOAD_L0A[0]:
        ■■■■■■■■■■■■
        ■ 2 cycles■
        ■■■■■■■■■■■■

LOAD_L0B[0]:
             ■■■■■■■■■■■■
             ■ 2 cycles■
             ■■■■■■■■■■■■

                       Pipeline Execution (1 uop/cycle throughput):
                       ■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■

uop0:                  [C0]■[C1]■[C2]■[C3]■■ACC[0]
                            ■    ■    ■    ■
uop1:                       [C0]■[C1]■[C2]■[C3]■■ACC[1]
                                 ■    ■    ■    ■
uop2:                            [C0]■[C1]■[C2]■[C3]■■ACC[2]
                                      ■    ■    ■    ■
uop3:                                 [C0]■[C1]■[C2]■[C3]■■ACC[3]
                                           ■    ■    ■    ■
uop4:                                      [C0]■[C1]■[C2]■[C3]■■ACC[0]
                                                ■    ■    ■    ■
...                                             ...  ...  ...  ...

STORE_ACC[0]:                                                  ■■■■■■■■■■■■■■■■■■■■
                                                               ■  4 cycles (8KB)■
```

```
                                                            ■■■■■■■■■■■■■■■■■■■■■■

Legend:
  [C0] = PE Cluster 0 (rows 0-3)
  [C1] = PE Cluster 1 (rows 4-7)
  [C2] = PE Cluster 2 (rows 8-11)
  [C3] = PE Cluster 3 (rows 12-15)
  ■ACC = Write complete tile to ACC buffer
```

10.2 Pipeline Stall Conditions

```
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                            Pipeline Stall Conditions                         ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
■                                                                             ■
■  The pipeline stalls when:                                                  ■
■                                                                             ■
■  1. L0A data not ready:                                                     ■
■     - uop requires L0A[idx] but L0A[idx].valid = 0                         ■
■     - Solution: Wait for LOAD_L0A to complete                              ■
■                                                                             ■
■  2. L0B data not ready:                                                     ■
■     - uop requires L0B[idx] but L0B[idx].valid = 0                         ■
■     - Solution: Wait for LOAD_L0B to complete                              ■
■                                                                             ■
■  3. ACC write conflict:                                                     ■
■     - Two uops in pipeline target same ACC[idx]                            ■
■     - Solution: Out-of-order issue avoids this by checking ACC availability■
■                                                                             ■
■  4. Issue queue empty:                                                      ■
■     - No ready uops to issue                                               ■
■     - Pipeline drains but doesn't stall                                    ■
■                                                                             ■
■  Stall behavior:                                                            ■
■     - Pipeline registers hold their values                                 ■
■     - No new uop enters stage 0                                            ■
■     - Downstream stages continue if they have valid data                   ■
■                                                                             ■
■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■■
```

10.3 Continuous Execution Example

```
Example: MATMUL(32, 32, 32) = 2×2×2 = 8 uops

Cycle:  0   1   2   3   4   5   6   7   8   9  10  11
        ■   ■   ■   ■   ■   ■   ■   ■   ■   ■   ■   ■
        ▼   ▼   ▼   ▼   ▼   ▼   ▼   ▼   ▼   ▼   ▼   ▼

uop0:  [C0][C1][C2][C3]■                          ACC[0] ready
uop1:      [C0][C1][C2][C3]■                      ACC[1] ready
uop2:          [C0][C1][C2][C3]■                  ACC[2] ready
uop3:              [C0][C1][C2][C3]■              ACC[3] ready
uop4:                  [C0][C1][C2][C3]■          ACC[0] += (K accumulate)
uop5:                      [C0][C1][C2][C3]■      ACC[1] +=
uop6:                          [C0][C1][C2][C3]■  ACC[2] +=
uop7:                              [C0][C1][C2][C3]■  ACC[3] += (all done)

Total cycles: 4 (fill) + 8 (uops) - 1 = 11 cycles
Throughput: 8 uops / 11 cycles = 0.73 uops/cycle
(Approaches 1.0 for larger matrices)
```

11. Implementation Modules

11.1 Module Hierarchy

```
cube_v2/
■■■ cube_v2.py              # Top-level module
```

```
■■■ cube_v2_types.py        # Dataclass definitions
■■■ cube_v2_consts.py       # Constants and addresses
■■■ cube_v2_decoder.py      # MATMUL instruction decoder
■■■ cube_v2_issue_queue.py  # 64-entry issue queue
■■■ cube_v2_l0a.py          # L0A buffer (64 entries)
■■■ cube_v2_l0b.py          # L0B buffer (64 entries)
■■■ cube_v2_acc.py          # ACC buffer (64 entries)
■■■ cube_v2_systolic.py     # 16×16 systolic array
■■■ cube_v2_mmio.py         # MMIO interface
■■■ cube_v2_fsm.py          # Main control FSM
```

11.2 Key Interfaces

```
# L0A/L0B Buffer Interface
class BufferEntry:
    data: Wire[4096]      # 16×16×16-bit
    valid: Wire[1]
    loading: Wire[1]
    ref_count: Wire[8]

# ACC Buffer Interface
class AccEntry:
    data: Wire[8192]      # 16×16×32-bit
    valid: Wire[1]
    computing: Wire[1]
    pending_k: Wire[8]

# Uop Interface
class Uop:
    l0a_idx: Wire[7]
    l0b_idx: Wire[7]
    acc_idx: Wire[7]
    is_first: Wire[1]
    is_last: Wire[1]

# Issue Queue Interface
class IssueQueueEntry:
    valid: Wire[1]
    uop: Uop
    l0a_ready: Wire[1]
    l0b_ready: Wire[1]
    acc_ready: Wire[1]
    issued: Wire[1]
```

12. Future Extensions

1. **Double Buffering**: Overlap computation with data loading

2. **Multiple Systolic Arrays**: Parallel tile computation

3. **Sparsity Support**: Skip zero tiles

4. **Quantization**: INT8/INT4 support

5. **Fusion**: Fused MATMUL + activation functions

References

• [Cube v1 Architecture](ARCHITECTURE.md)

• [Cube v1 Visual Guide](VISUAL_GUIDE.md)

- [pyCircuit Usage Guide](../../../../docs/USAGE.md)