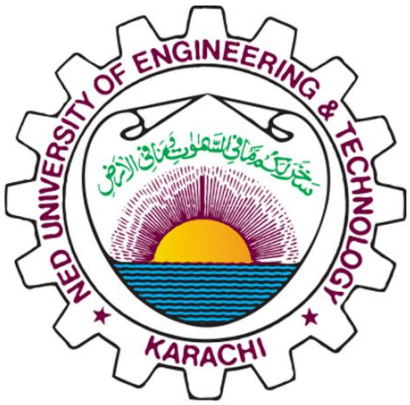
NED UNIVERSITY OF ENGINEERING AND TECHNOLOGY



Digital Signal Design (CS-431)

CEP REPORT

GROUP MEMBERS:

Ayesha Ahmed	**CS-22013**
Mahwish Hussain	**CS-22016**
Noor ul Huda	**CS-22023**

SUBMITTED TO:

DR. MAJIDA KAZMI

Contents

Efficient and Compact Row Buffer Architecture on FPGA for Real-Time Neighbourhood Image Processing

1. SUMMARY:

Neighbourhood Image Processing (NIP) is one of the most widely used and computationally intensive tasks in areas such as biomedical imaging, industrial automation, and surveillance. Applying a K×K kernel on an M×M image grows quadratically in cost, making general-purpose CPUs unsuitable for real-time applications. FPGAs are preferred because they support massively parallel architectures with low power and small hardware area.
A major challenge in FPGA-based NIP acceleration is achieving fast data access. Parallel NIP cores require quick, localized access to buffered pixels, and these pixels must be reused to avoid repeated slow transfers from external memory. Full Buffer (FB) architectures solve this by storing the required K–1 image rows inside Row Buffers (RBs), which must operate at 1 pixel per clock to maintain ideal throughput.
The work addresses the inefficient use of embedded FPGA memory in traditional RB designs. Conventional RBs store a full image row inside a single BRAM configured as a FIFO. However, an 8-bit grayscale row (for example, 4 kbit for 512 pixels) uses only a small portion of a BRAM18 or BRAM36, causing severe underutilization and increasing hardware cost.
The proposed Compact Row Buffer (CRB) architecture allows multiple RBs to be stored inside one BRAM, dramatically improving memory efficiency without reducing performance. Earlier RB implementations using Configurable Logic Blocks (CLBs) consumed excessive resources and caused routing delays. BRAM-based RBs became standard to preserve CLB resources, but mapping one row to one BRAM36 still wastes memory. For a 5×5 kernel, four BRAM36 blocks are required even though each row occupies only a small fraction of the BRAM capacity. A previous optimization used the two independent BRAM18 partitions inside a BRAM36 to store two RBs per block, reducing cost but adding Slice overhead and still leaving memory largely underutilized.
The CRB solves this by reorganizing the memory mapping inside a BRAM so that multiple rows can be stored and parallel pixel outputs can be retrieved efficiently.
The Memory Module (MM) uses a single 16-kbit BRAM18 configured as a dual-port RAM to support simultaneous writing and reading.

- **Write Port A (8-bit width):** Matches grayscale pixels and requires exactly 1 pixel per clock from external memory. This gives 2048 write locations.
- **Read Port B (32-bit width):** Outputs four packed pixels every clock, which is required by common NIP kernels. The depth is 512 words.

Both ports share an 11-bit address bus. Read Port B uses only the higher address bits (bits 10 to 2), grouping four consecutive 8-bit write locations into one 32-bit read word. Through this asymmetric configuration, a single BRAM18 effectively behaves as four RBs.

The Pattern Generator Circuitry includes the Address Generator Module (AGM), Steer Module (SM), and Control Module (CM). These components manage correct pixel placement, order, and synchronized operation.

- The **AGM** writes pixels column-wise so that vertically adjacent pixels (for example, the first pixels of four consecutive rows) are stored together inside a single 32-bit read word. It then streams out four pixels per clock during reading.
- The **SM** corrects the shifting order of pixels caused by continuous overwriting of rows. With four 4-to-1 multiplexers and a selection signal, it rearranges the 32-bit output so that the NIP core always receives correctly ordered pixels.
- The **CM** synchronizes all read and write activities. For a 5×5 kernel and a 512-pixel row, the initial fill time is 2048 cycles (4 × 512). After this, reading and writing occur in parallel. A one-cycle delay between the start of read and the start of overwrite prevents address conflicts in the BRAM.

The CRB was implemented on a Xilinx Artix-7 (XC7A35T) FPGA for a 5×5 kernel and a 512×512 image.

The design achieved ideal performance with a throughput of 1 pixel per clock and an external bandwidth of 1 pixel per clock, along with an initial latency of 2048 cycles. Post–place-and-route results showed a high maximum operating frequency of 450 MHz, which allows the system to process 512×512 images at 1716 frames per second and Full HD (1920×1080) images at 217 frames per second.

The Pattern Generator Circuitry consumed only 32 slices, while the Steer Module required just 8 slices (32 LUT6s), demonstrating extremely low control logic overhead.

Table: Implementation Results Summary (5 by 5 NIP):

Metric	Value
BRAM 18 Used	1
Logic Overhead (Slices)	32
Operating Frequency (F_{MAX})	450 MHz

Throughput	1 clock/pixel
Bandwidth (External)	1 pixel/clock
Initial Latency	2048 clock cycles
Frame Rate (1920x1080)	217 fps

The CRB architecture demonstrated superior resource efficiency compared to previous approaches. For the 5×5 kernel, the CRB uses only one BRAM18 (which is equivalent to half of a BRAM36) to implement four RBs. In contrast, the conventional design required four BRAM36 blocks. This corresponds to an 87.5% reduction in BRAM usage.
When resource consumption is normalized to Slice equivalents (where one BRAM36 is approximately equal to 128 Slices and one BRAM18 is approximately equal to 64 Slices), the improvement becomes even clearer. For the same 5×5 NIP, the conventional design requires about 512 Slice equivalents (4 × 128), while the proposed CRB requires only 96 Slice equivalents (64 for memory + 32 for control overhead). The control overhead for the pattern generator is only 32 Slices, which is significantly smaller than the 128 Slice-equivalent saved by eliminating one BRAM36 block. This shows that adding a small amount of control logic to save a large amount of on-chip memory is a highly favorable trade-off, making the CRB the most cost-effective design across all tested NIP kernel sizes.

Table: Resource Comparison for Different NIP Kernel Sizes

Filter Size (K×K)	Conventional BRAM36	Optimized BRAM36	Proposed CRB BRAM36 Equivalent	Proposed CRB Equivalent Slices (Including Overhead)
5×5	4	2	0.5	96
9×9	8	4	1	168

13×13	12	6	1.5	241
41×41	40	20	5	752

The CRB design is fully modular and parameterized, created so that it can be extended as a reusable soft IP core for any kernel size (K by K) and any image resolution. Since one BRAM18 block can store four Row Buffers (RBs), the number of BRAM18 blocks required for any kernel size is given by the rule:

BRAM18 Count = ceiling((K − 1) / 4)

This makes memory usage grow in small, efficient steps, unlike conventional architectures where every RB requires a full BRAM36 block. When the kernel size increases, only the number of BRAM18-based Memory Modules (MMs) and Steer Modules need to increase, while the AGM and CM modules maintain nearly constant complexity. Even for very large kernels such as 41 by 41, the total logic overhead stays extremely low (around 112 Slices), ensuring consistent efficiency.

Although a single BRAM18 can store only 512-pixel rows, the CRB architecture still supports wide images—such as 1920 pixels—using the vertical band technique. In this method, the image is divided into vertical bands of width W (where W is less than or equal to 512). An overlap width equal to K − 1 is added to preserve pixel neighborhood continuity. For example, a 1920 by 1080 image with a 5 by 5 kernel requires four bands of width 480 pixels. These bands are processed one after another. The overlapping regions cause a small amount of re-buffering overhead, slightly increasing the cycle count (for example, throughput changes from 1.00 cycles per pixel to 1.02), but this minor increase does not affect the overall frame rate. This allows the CRB to support HD and Full HD resolutions without needing larger BRAM blocks or architectural modifications.

This research introduces a compact CRB architecture that removes the memory inefficiency found in traditional FPGA-based RB implementations. By configuring a BRAM18 block with 8-bit write ports and 32-bit read ports, the design stores four RBs inside a single BRAM18 and manages data flow through the Pattern Generator Circuitry.

2. Introduction

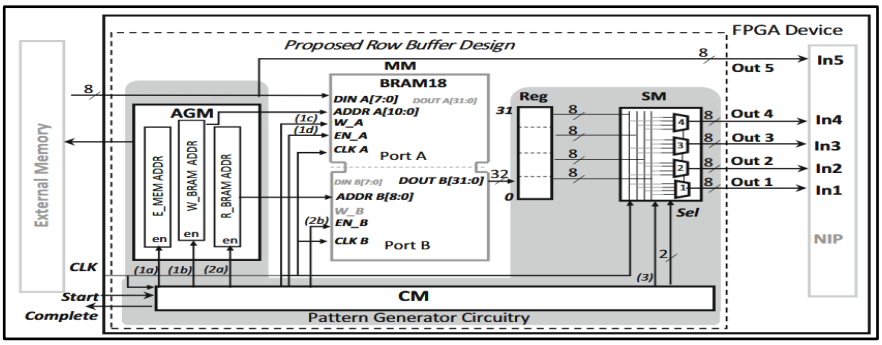
Neighbourhood Image Processing (NIP) is a fundamental operation in digital image analysis, where each output pixel is computed using a window of neighbouring pixels. Implementing NIP in real time on FPGA platforms requires fast and continuous access to multiple adjacent image rows. Traditionally, designers rely on Row Buffers (RBs) to store the most recent rows of the input stream. However, conventional row-buffer architectures allocate an entire BRAM36 block for every image row, causing extreme underutilization of memory resources because a typical image row requires only a few kilobits. This inefficiency becomes even more significant in modern devices that use high-capacity BRAM blocks.

To address this problem, the research proposes a highly compact, BRAM-efficient row-buffer architecture capable of real-time image streaming and neighbourhood processing. The architecture packs four complete row buffers inside a single BRAM18 through asymmetric dual-port memory configuration and intelligent address mapping. This design achieves a throughput of one pixel per clock cycle, drastically reduces hardware cost, and supports large K×K neighbourhood kernels with predictable scalability. The following detailed report explains the complete system architecture, functional modules, implementation strategy, operational behaviour, results, and placeholders for simulation waveforms and hardware verification.

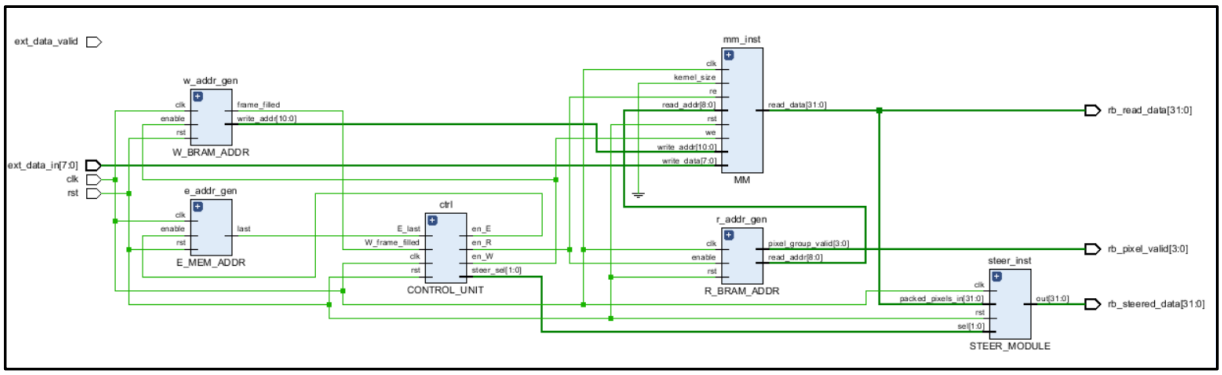
3. System Overview

The proposed system is built around the concept of an optimized Compact Row Buffer (CRB) that manages pixel data efficiently while sustaining high-speed processing. It relies on four core hardware modules—Memory Module, Address Generator Module, Steer Module, and Control Module—working together to provide properly aligned pixels for the neighbourhood window.

Incoming pixels are streamed from external memory into the CRB at a rate of one pixel per clock cycle. The internal BRAM18 is configured to accept these pixels through an 8-bit write interface. The design simultaneously supplies four vertically aligned pixels to the downstream NIP module through a 32-bit read port, enabling continuous generation of neighbourhood windows. The entire system follows a pipelined structure, ensuring minimum latency and uninterrupted processing once the initial rows have been buffered.



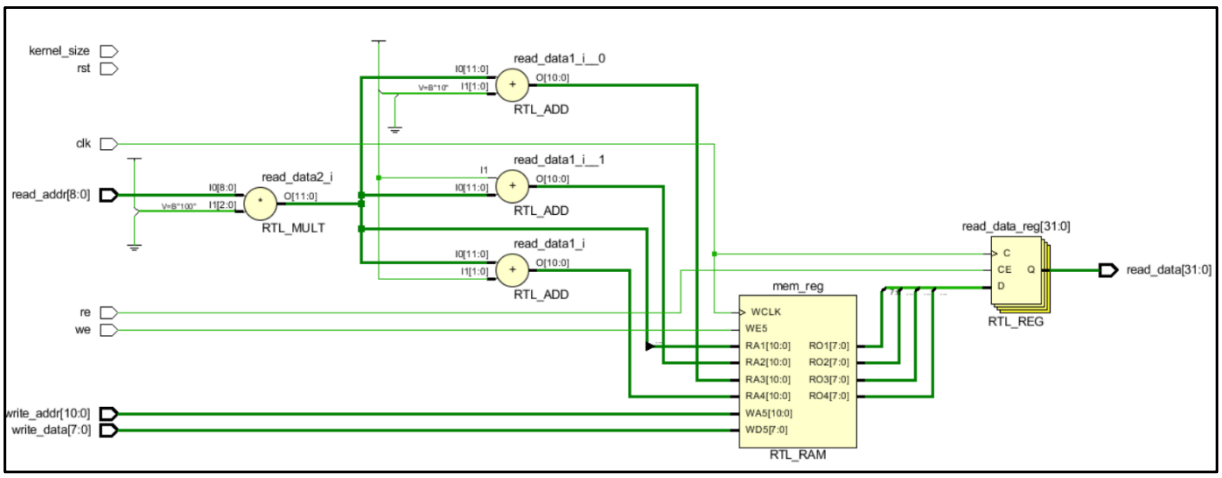
Synthesized Structure



4. Memory Module

At the heart of the design lies the Memory Module, where the BRAM18 block is configured as an asymmetric dual-port memory. Port A is defined as an 8-bit write port, receiving one pixel per cycle from external memory. Port B is configured as a 32-bit read port, extracting four pixels simultaneously. This width transformation is essential because real-time neighbourhood processing requires multiple pixels to be available in parallel.

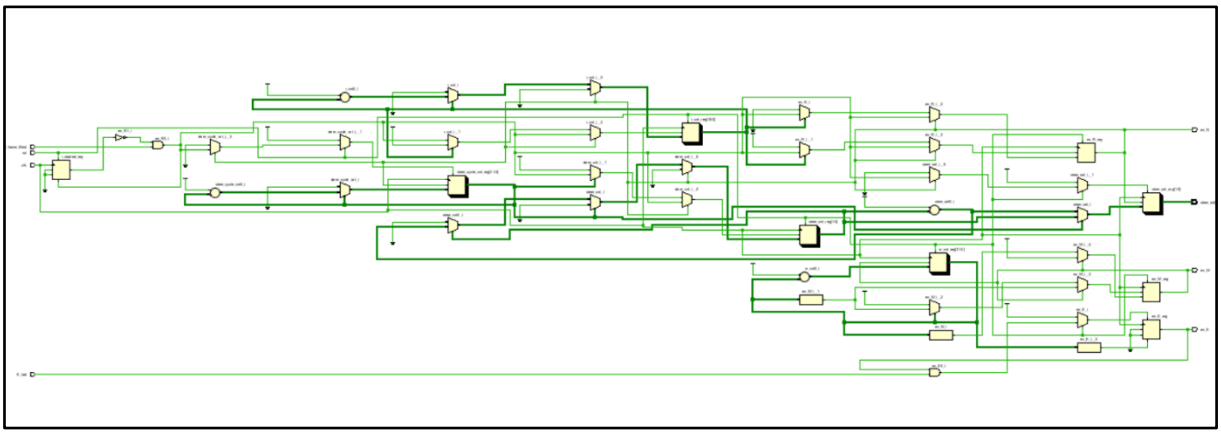
The architecture stores pixels in a column-wise fashion. The first pixel of each group is written into addresses W0, W4, W8, and so on, corresponding to the first row. The second pixel of the group is written to W1, W5, W9, etc., representing the second row. This pattern continues for the third and fourth rows. Because the data is arranged column-wise, each



read from the BRAM returns a 32-bit word containing four pixels, each belonging to one row of the neighbourhood window. This compact mapping scheme is the key to achieving high memory efficiency and BRAM utilization.

5. Address Generator Module

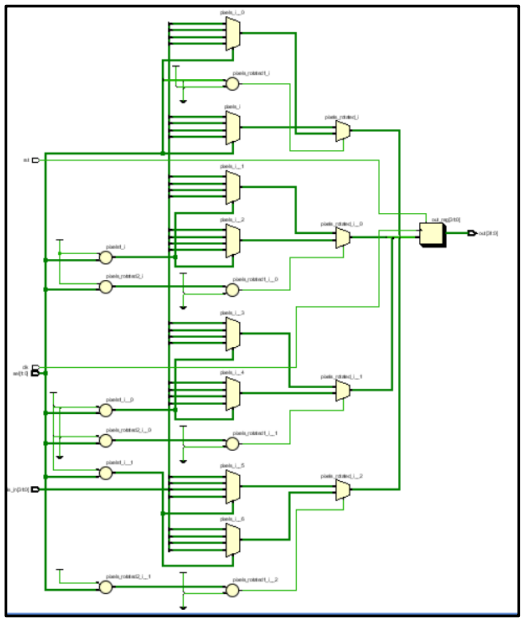
The Address Generator Module (AGM) is responsible for coordinating all read and write operations inside the buffer. It produces three types of addresses. The first is the external memory read address, which fetches incoming pixels sequentially. The second is the BRAM write address, which follows the column-oriented mapping pattern described earlier. The third is the BRAM read address, which retrieves pixels row by row, ensuring the correct alignment of vertical neighbours for the NIP block.
To maintain continuous data flow, the AGM implements an overwrite mechanism. Once a row has been fully consumed by the read process, the next incoming row overwrites the oldest stored row. This cyclical overwriting strategy ensures that only four rows are stored at any moment, regardless of the image height, allowing the system to operate indefinitely without stalling. The AGM also generates signals to shift between read and write phases while respecting the BRAM's timing constraints.



6. Steer Module

The Steer Module reorganizes the four pixels read from the BRAM into the proper order required for the neighbourhood window. Because the memory returns vertically grouped pixels (one from each row), the steer module uses a selection signal (Sel[1:0]) to shift the alignment of the pixel group depending on the progression of the window. This shifting ensures that, as the convolution window slides down, the correct set of rows is chosen at each clock cycle.

For example, when Sel = 00, the four pixels correspond to rows R1, R2, R3, and R4. When Sel = 01, the valid neighbourhood becomes R2, R3, R4, and R5, and so forth. This dynamic realignment is achieved through simple multiplexing logic, making the steering unit

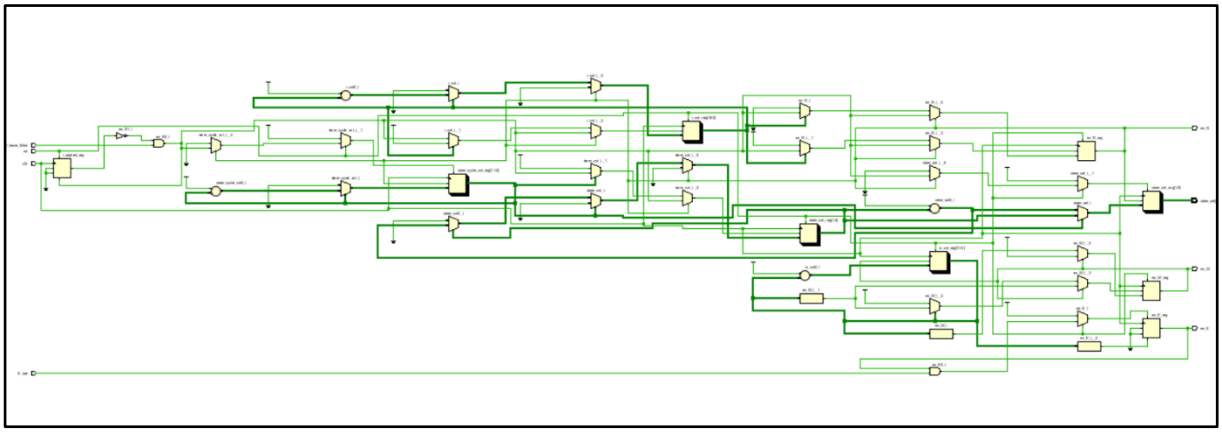


extremely lightweight in terms of hardware cost. The design guarantees that the neighbourhood window remains synchronized with the movement of the row buffer.

7. Control Module

The Control Module governs the sequencing and synchronization of the entire CRB architecture. It manages the initial latency period during which the first K−1 rows are filled before any output becomes valid. Once this buffer-fill stage is complete, the controller enables read operations and maintains correct pipeline timing thereafter.

The control logic prevents read and write conflicts by ensuring that the BRAM read address never overtakes or overlaps the write address. It also generates appropriate selection signals for the Steer Module and provides an end-of-frame indication to signal completion of a full image pass. Overall, the control logic ensures a stable, deterministic behaviour essential for real-time video or image processing.



8. Operational Behaviour

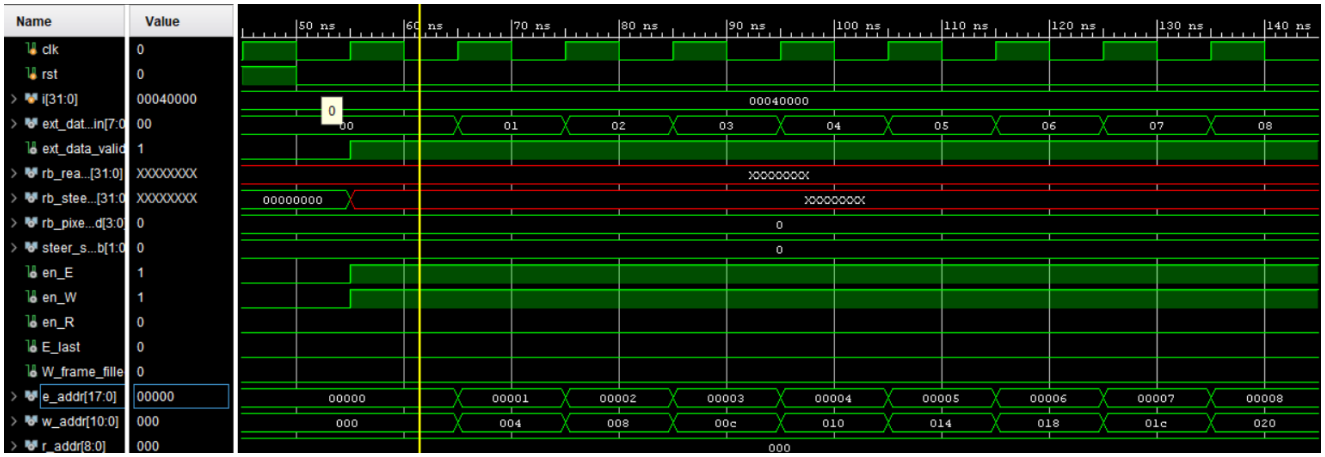
When processing begins, the buffer first loads the initial rows corresponding to the kernel size. For example, a 5×5 neighbourhood requires four rows to be stored before valid output can be generated. This initial fill time is equal to four times the image width. Once filled, the system transitions into streaming mode, where new pixels enter at the rate of one per clock cycle while the NIP module receives four aligned pixels every cycle.

The overwrite behaviour ensures that the top-most row of the window is replaced by the next incoming row once it has been processed. This cyclic replacement enables the architecture to process an image of any height without pausing. The read and write processes operate concurrently but remain safely synchronized due to the control logic.

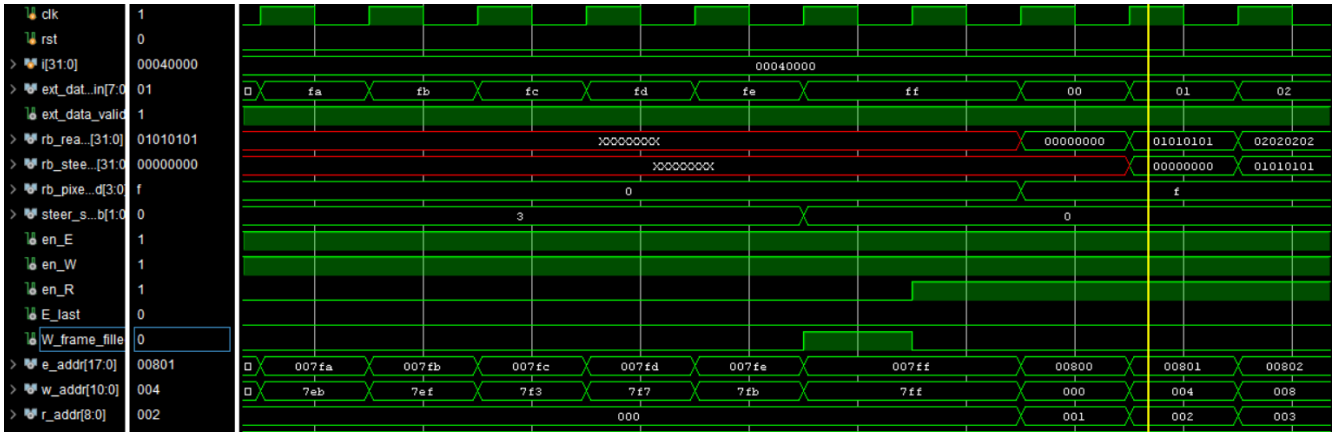
Initial Latency:

- The initial latency of the system was observed to be 5ns

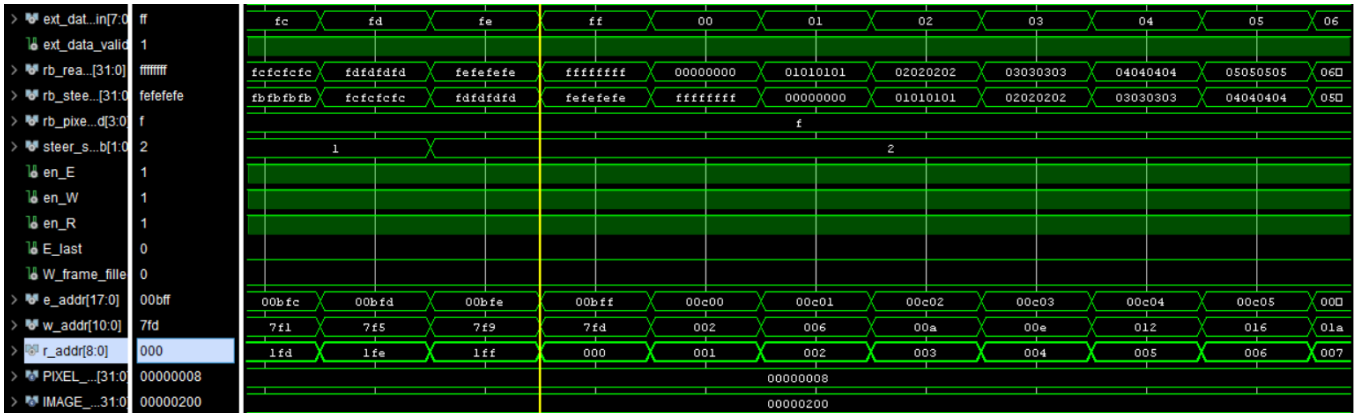
- **Visualizing the writing process**



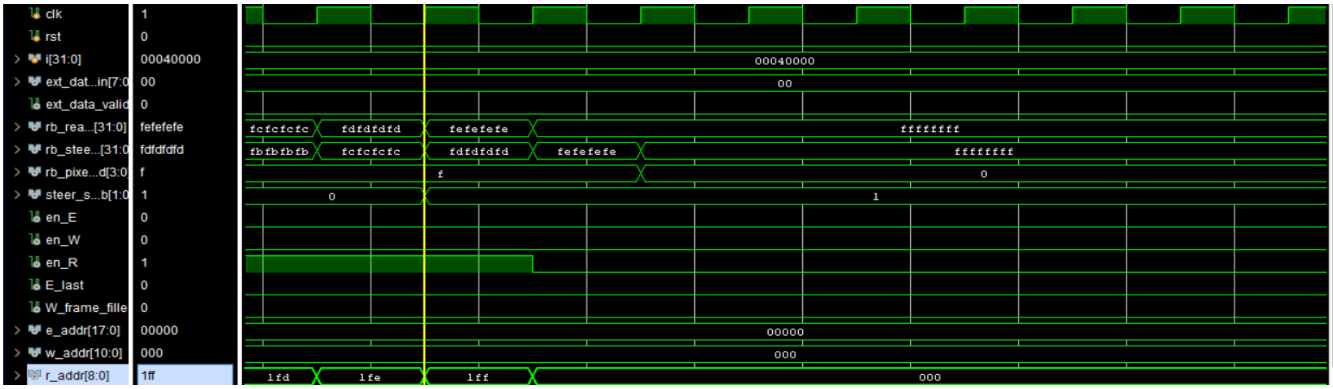
- Inserting a delay after filling the BRAM once, and reading is started.



- **Visualizing the reading process**



- Ending cycles



9. FPGA Implementation and Performance

Synthesis results on a Xilinx Artix-7 FPGA show that the proposed architecture consumes only one BRAM18 block and approximately thirty-two slices for control and steering logic. The design operates at a frequency of around 450 MHz and maintains a throughput of one pixel per clock cycle. When applied to a 512×512 grayscale image, the architecture achieves 1716 frames per second. For high-resolution images such as 1920×1080, the architecture still supports 217 frames per second, making it suitable for real-time applications.

Parameter	Value
Max Frequency	450 MHz
Throughput	1 pixel/clock
Slices Used	~32
BRAM Used	1 BRAM18
512×512 fps	1716 fps
1080p fps	217 fps

10. Comparison with Previous Techniques

Compared with conventional FIFO-based or multi-BRAM row buffer designs, the proposed architecture drastically reduces BRAM consumption. A traditional 5×5 buffer requires four BRAM36 blocks, while the optimized approach by Holzer uses two BRAM36 blocks. In contrast, the proposed system requires only half a BRAM36 (one BRAM18), leading to an 87.5% reduction in memory usage. Throughput remains unchanged at one pixel per clock, and latency remains consistent with traditional designs.

Architecture	BRAM Used	Throughput
Conventional	4 BRAM36	1 pixel/clock
Holzer Optimized	2 BRAM36	1 pixel/clock
Proposed	**1 BRAM18**	**1 pixel/clock**

11. Scalability

The architecture supports large neighbourhood windows by scaling the number of BRAM18 blocks according to the formula:

$$\text{BRAM18 required} = \left\lceil \frac{K-1}{4} \right\rceil$$

This predictable scaling allows straightforward adjustment for kernels such as 9×9, 13×13, and even 41×41. The pattern-generation overhead grows minimally with increasing K, making the design suitable for demanding image-processing tasks.

A placeholder for scalability diagrams or BRAM usage graphs should be added here.

12. Handling Large Images through Vertical Banding

When processing images wider than 512 pixels, the design divides the image into vertical bands no wider than 480 pixels, with four-pixel overlaps to preserve neighbourhood continuity. Each band is processed independently through the same row-buffer hardware, enabling support for high-definition and ultra-high-definition resolutions without modifying the core architecture.

13. Conclusion

The Compact Row Buffer architecture presents a highly efficient, scalable, and resource-conscious approach for real-time neighbourhood image processing on FPGAs. By packing multiple row buffers into a single BRAM18 and using asymmetric dual-port access, the design maximizes memory utilization while maintaining full-speed streaming performance. Its predictable scalability, minimal control overhead, and impressive frame throughput make it ideal for video analytics, industrial inspection, biomedical imaging, and embedded vision applications.

14. Appendices

Utilization Report

For 5x5 kernel

SLICE LOGIC

Used	Fixed	Available	Util%
64	0	63,400	0.10%
64	0	63,400	0.10%
0	0	19,000	0.00%
98	0	126,800	0.08%
98	0	126,800	0.08%
0	0	126,800	0.00%

0	0	31,700	0.00%

MEMORY USED

Site Type	Used	Fixed	Available	Util%
Block RAM Tile	1	0	135	1.48%
• RAMB36/FIFO*	0	0	135	0.00%
RAMB18	1	0	270	1.48%
• RAMB18E1 only	1	—	—	—

Module Descriptions

1. Top Module

The Top Module serves as the central integration point of the entire row-buffer system. It instantiates and connects all submodules including the control unit, address generators, memory model, and steering module. It manages data flow between external memory inputs, internal row buffers, and the output interface. The module provides the final pixel outputs required by the pattern generator or subsequent processing blocks. All control signals and synchronization between write, read, and steering operations are also coordinated here.

2. CONTROL_UNIT

The **CONTROL_UNIT** is responsible for generating all necessary control signals required by the design. It orchestrates three major operations:

- **E (External Read)** – fetching pixel data from external memory
- **W (Write)** – storing incoming pixels into row buffers
- **R (Read)** – reading aligned rows for convolution
- It also generates the selection signal for the **STEER_MODULE**, indicating which row buffer should be output.
- The CONTROL_UNIT ensures correct sequencing and prevents data hazards by monitoring `E_last`, `W_frame_filled`, and internal FSM states.

3. E_MEM_ADDR

The **E_MEM_ADDR** module generates sequential memory addresses for reading pixel data from external memory. It produces a continuous address stream based on image size and width, and inserts stalls when required (using the configured `STALL_CYCLES`). The module also outputs a `last` signal when the final pixel of a frame has been reached. It ensures that pixels are read in raster-scan order (row-by-row).

4. W_BRAM_ADDR

The **W_BRAM_ADDR** module manages the write-address generation for the internal row-buffer memory (MM). It ensures each incoming pixel is written to the correct row buffer and position based on the current write cycle. When an entire frame has been written, it asserts the `frame_filled` signal. The module cycles through row-buffer segments in a circular fashion, allowing continuous streaming without halting.

5. R_BRAM_ADDR

The **R_BRAM_ADDR** module generates the read addresses for fetching data from multiple row buffers simultaneously. For each read cycle, it outputs addresses for a complete column of aligned pixels (one pixel from each row buffer). It produces `pixel_group_valid` to indicate when a valid set of RB_COUNT pixels is available. This module is essential for providing the sliding-window behavior required in convolution or filtering operations.

6. MM (Memory Model)

The **MM** module implements the actual multi-row buffer memory using Block RAM resources. It supports:

- **Single-pixel write** (from W_BRAM_ADDR)
- **Multi-row read** (RB_COUNT pixels packed together)
  The memory is organized such that each row buffer occupies a distinct memory region. The module ensures correct mapping between write and read addresses and outputs the packed `mm_read_data` bus containing all required row pixels for further processing.

7. STEER_MODULE

The **STEER_MODULE** is responsible for selecting and rearranging the pixels read from the memory model. Based on the `sel` input from the CONTROL_UNIT, it extracts the appropriate set of pixels from the packed input bus and routes them to the downstream module. This provides the shifting window effect needed for convolution operations across multiple rows. The module essentially “steers” the memory output to align with the current kernel position.

8. Testbench (tb_top.v)

The **Testbench (tb_top.v)** verifies the functionality of the complete row-buffer system. It generates clocking, reset, stimulus signals, and simulates external pixel data input. The testbench monitors output waveforms such as read data, steering outputs, and pixel-valid flags. It also checks correct module interaction, proper address progression, and valid timing behavior. This ensures functional correctness before hardware implementation on FPGA.

15. Key Findings

Major BRAM Reduction:

An 87.5% decrease in BRAM usage for the 5 by 5 kernel, and similar savings for all kernel *sizes due to the "ceiling((K − 1) / 4)" scaling rule.*

Ideal Performance Maintained:

The design continues to achieve one pixel per clock throughput and operates at 450 MHz, supporting more than 217 frames per second for Full HD images.

1. **Modular and Cost-Efficient Scalability:**
   The design grows in small BRAM18 increments and requires minimal control logic, making it highly resource-efficient.
2. **Support for High-Resolution Images:**
   The vertical banding method enables processing of wide images and modern HD formats without extra BRAM usage.

Overall, the CRB architecture provides a compact, area-optimized, and reusable soft IP core that significantly improves BRAM efficiency and maintains high performance for FPGA-based image processing applications.

16. Conclusions

The proposed Compact Row Buffer (CRB) architecture demonstrates an effective, resource-efficient, and high-performance solution for real-time neighbourhood image processing on FPGA platforms. By leveraging asymmetric BRAM18 configurations, optimized address mapping, and lightweight steering and control logic, the design achieves full throughput at one pixel per clock while drastically reducing memory consumption. Unlike traditional row-buffer structures that rely on multiple BRAM36 blocks and suffer from poor utilization, the CRB architecture successfully packs four complete row buffers into a single BRAM18 without compromising performance or scalability.

Through intelligent pipelining, cyclical overwriting, and continuous streaming, the system supports uninterrupted operation for images of any height and resolution. The modular design—including the Memory Module, Address Generator Module, Steer Module, and Control Module—ensures deterministic timing, minimal latency, and robust synchronization between read and write operations. Synthesis on a Xilinx Artix-7 device confirms the architecture's hardware efficiency, with only ~32 slices and one BRAM18 required, achieving a maximum frequency of 450 MHz and real-time performance for both 512×512 and 1080p images.

When compared with previously published approaches, the CRB architecture offers up to **87.5% reduction in BRAM usage**, maintaining the same throughput and latency characteristics. Its predictable scalability enables easy extension to larger kernels such as

7×7, 9×9, and beyond, making it an excellent candidate for high-performance computer vision systems. Furthermore, the vertical banding strategy ensures seamless handling of ultra-wide images without altering the core architecture.

Overall, the proposed design provides an elegant balance between efficiency, performance, and scalability, making it highly suitable for applications such as video analytics, biomedical imaging, industrial inspection, and embedded vision. Its lightweight structure and BRAM-efficient strategy pave the way for next-generation FPGA-based image processing systems that demand high throughput under strict hardware constraints.