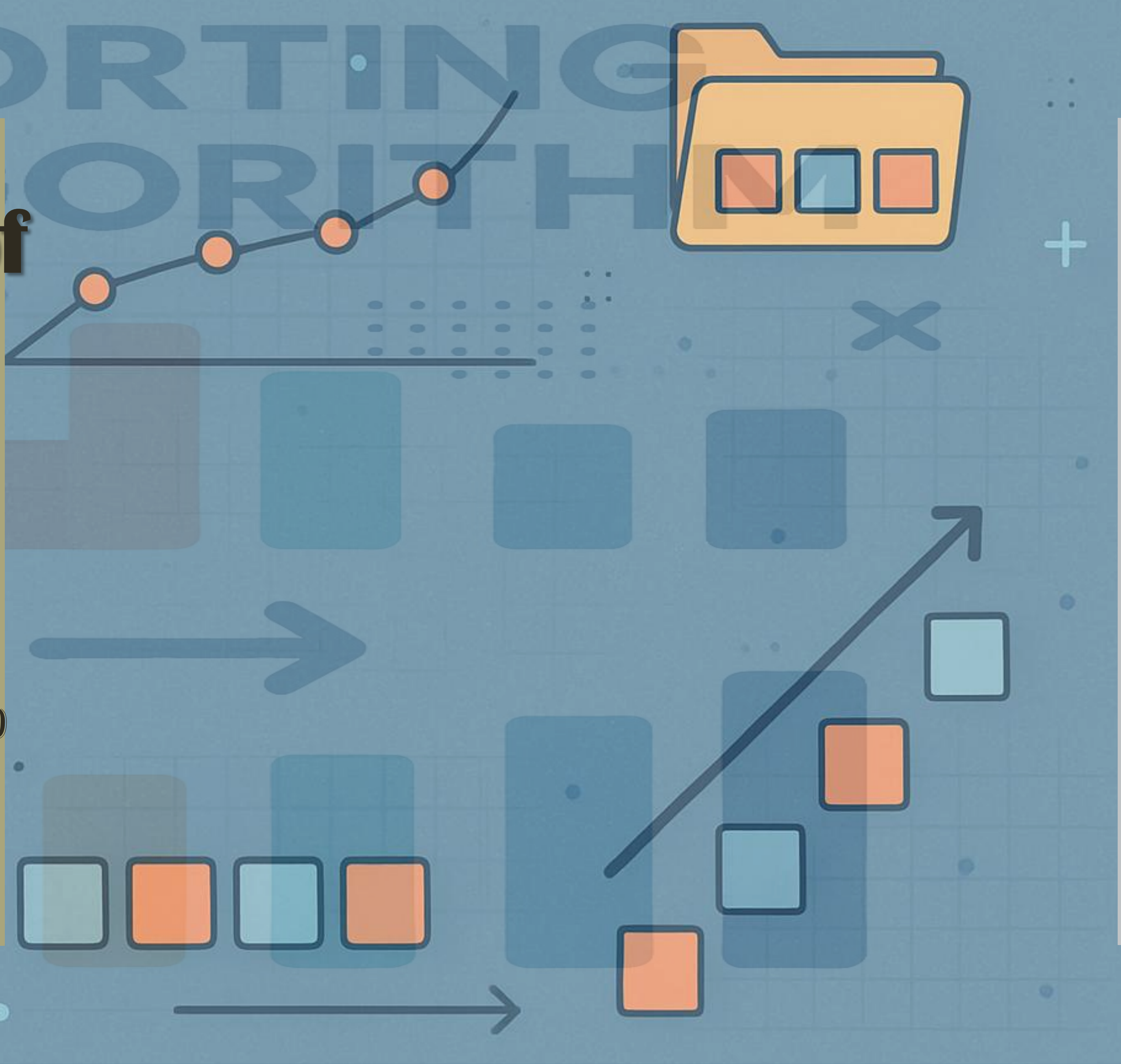
Implementation of partial sorting algorithm in Verilog

-BY

HIMANSHU KUMAR(122201041)

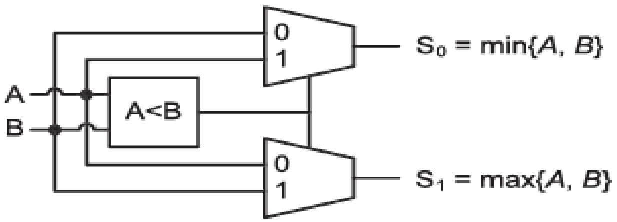
VAIBHAV(122201017)

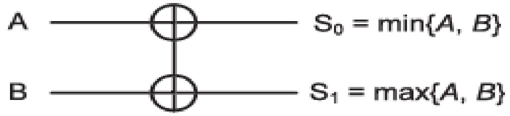


Compare & Exchange Blocks (CAE blocks):-

1) Ascending Compare-and-Swap

```
module compare_swap_asc (
    input  signed [15:0] a,
    input  signed [15:0] b,
    output signed [15:0] s0,
    output signed [15:0] s1
);
    assign s0 = (a < b) ? a : b;  // min
    assign s1 = (a < b) ? b : a;  // max
endmodule
```



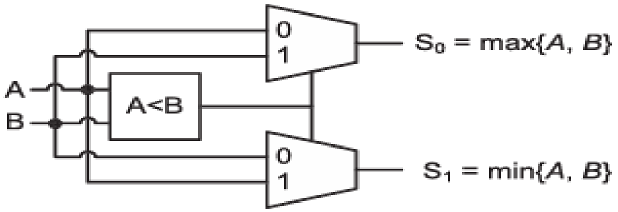


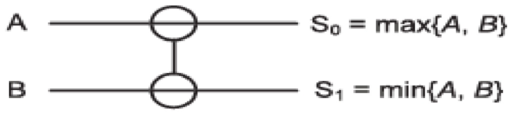
→

- **OUTPUTS:**
  - •s0: Smaller of a and b
  - •s1: Larger of a and b
- Sorts two inputs in **ascending order** (top = max, bottom = min)

2) Descending Compare-and-Swap

```
module compare_swap_desc (
    input  signed [15:0] a,
    input  signed [15:0] b,
    output signed [15:0] s0,
    output signed [15:0] s1
);
    assign s0 = (a > b) ? a : b;  // max
    assign s1 = (a > b) ? b : a;  // min
endmodule
```





→

- **OUTPUTS:**
  - •s0: Larger of a and b
  - •s1: Smaller of a and b
- Sorts two inputs in **descending order** (top = min, bottom = max)

MAX_K module :-

Overview:

The **MAX-K unit** processes K inputs obtained from two **Bitonic Merge (BM)** units.

Each BM unit provides $K/2$ inputs:

- One set is **ascendingly sorted**
- The other set is **descendingly sorted**

Step 1: MAX-K Processing

The MAX-K unit selects the **top** $K/2$ **maximum values** from the combined K inputs.

The output is not fully sorted but follows a specific pattern:

- The **top** $K/4$ **elements** are in **ascending order**.
- The **bottom** $K/4$ **elements** are in **descending order**.

Step 2: Final Sorting using BM-$K/2$

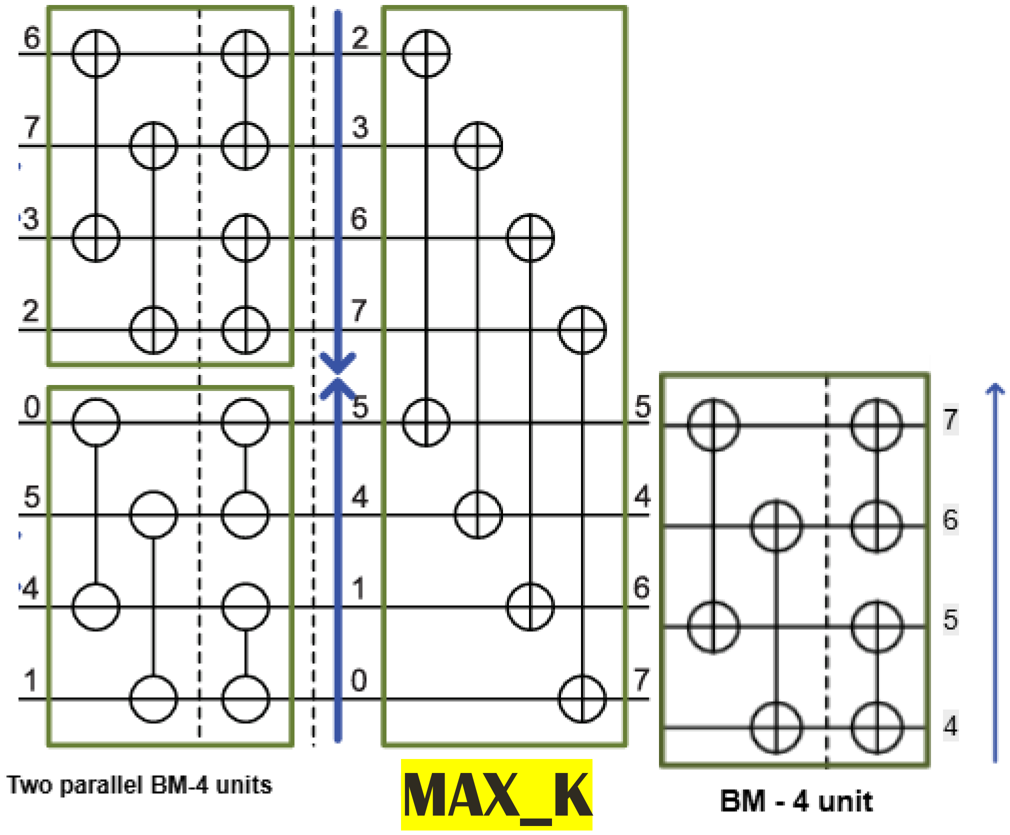
To sort the $K/2$ selected elements in **descending order**, the output of the MAX-K unit is fed into the **BM-$K/2$** unit.

Since the input follows a **bitonic sequence** ($K/4$ ascending followed by $K/4$ descending), BM-$K/2$ efficiently sorts them.

Result:

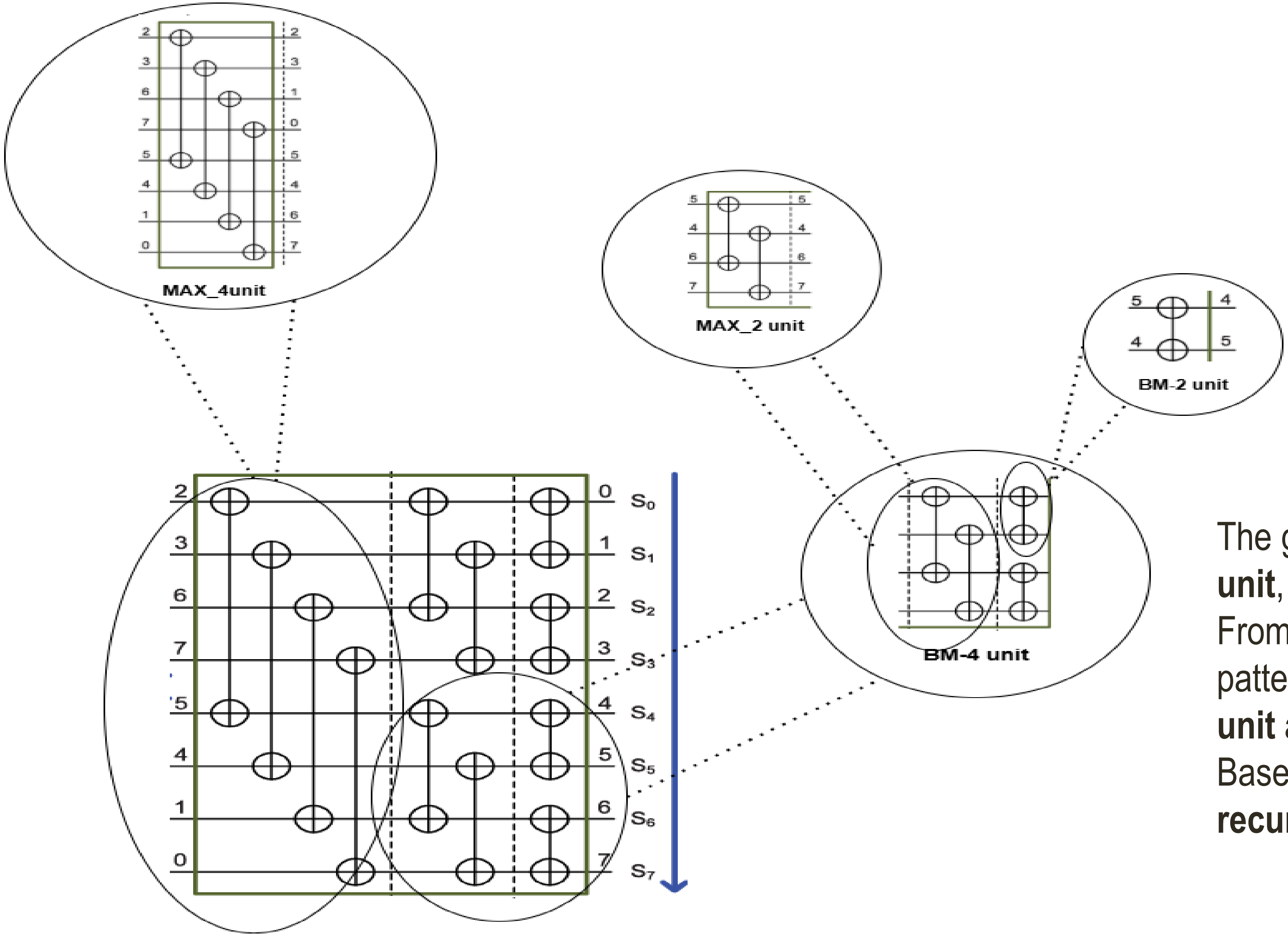
The final output consists of the $K/2$ **largest elements** in **strict descending order**.

This output is ready for the next processing stage or for final use.



[Fig: Example of max_4, where the max_selector takes inputs from BM-4 (asc.) and BM-4 (desc.)]

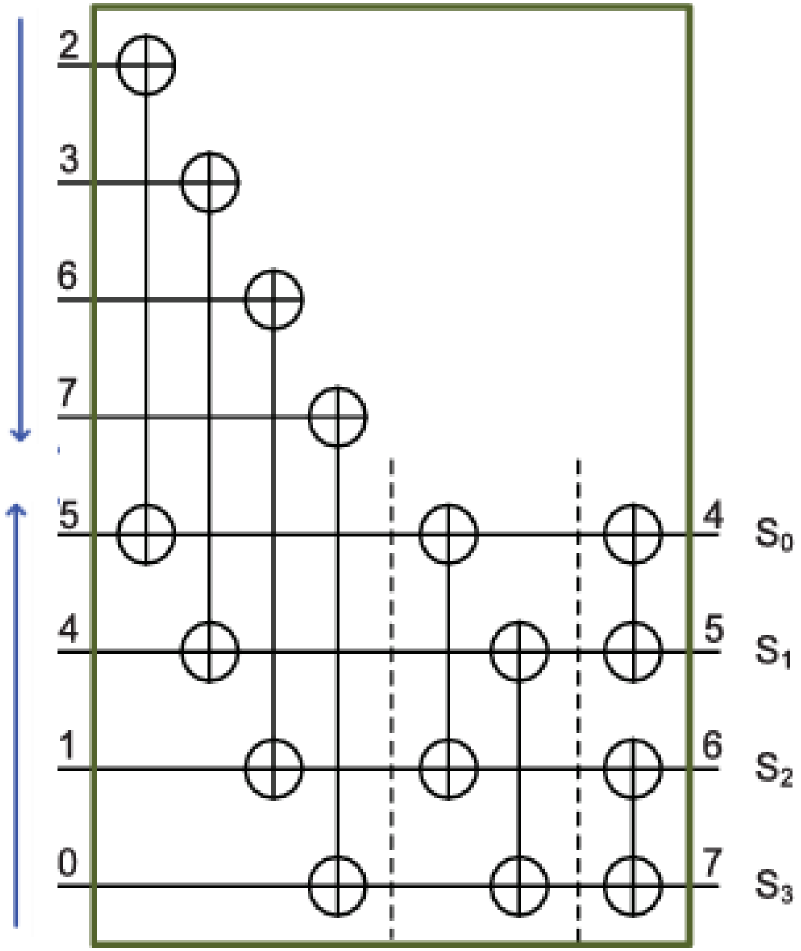
BM - K module :-



(BM - 8 unit)

The given diagram represents a **Bitonic Merge Sort unit**, specifically a **BM8 unit**.
From the diagram, we observe a recurring structural pattern that a **BM-K unit** is composed of **one MAX-K unit** and **two BM-K/2 units**.
Based on this observation, we implemented a **recursive algorithm** to construct the BM unit efficiently.

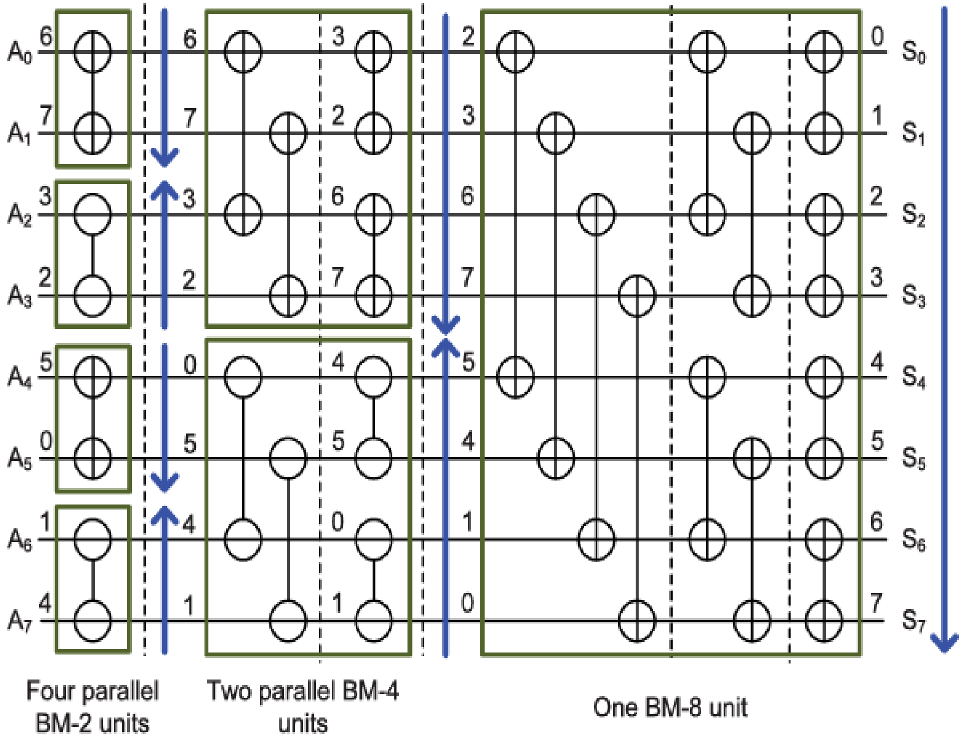
MAXNBY2 module :-



[Fig :- example (MAX8/2 unit)]

- The unit shown beside is an example of **MaxN/2 unit**, which performs **partial sorting** to reduce the number of compare-and-swap operations. This approach results in **lower area usage** and **fewer hardware components**.
- This block takes **N inputs**, where:
  - The first **N/2 elements are in ascending order**
  - The remaining **N/2 elements are in descending order**
- It outputs the **top N/2 maximum elements**, which are fully sorted. Structurally, this unit is composed of two well-known blocks:
  - A **MAX-N unit**, followed by
  - A **BM-N/2 unit**
- Thus, we implement the MaxN/2 unit by combining a **MAX-N unit** with a **BM-N/2 unit**.
- If we instead use **descending compare-and-swap operations in BM-N/2**, the output will be in descending order, forming a **descending MaxN/2 unit** (e.g., **Max8/2**).

Complete Bitonic Merge (BM) Sorting Unit:-



[Fig : - example of complete BM unit in which random unordered 8-inputs are getting sorted in ascending order]

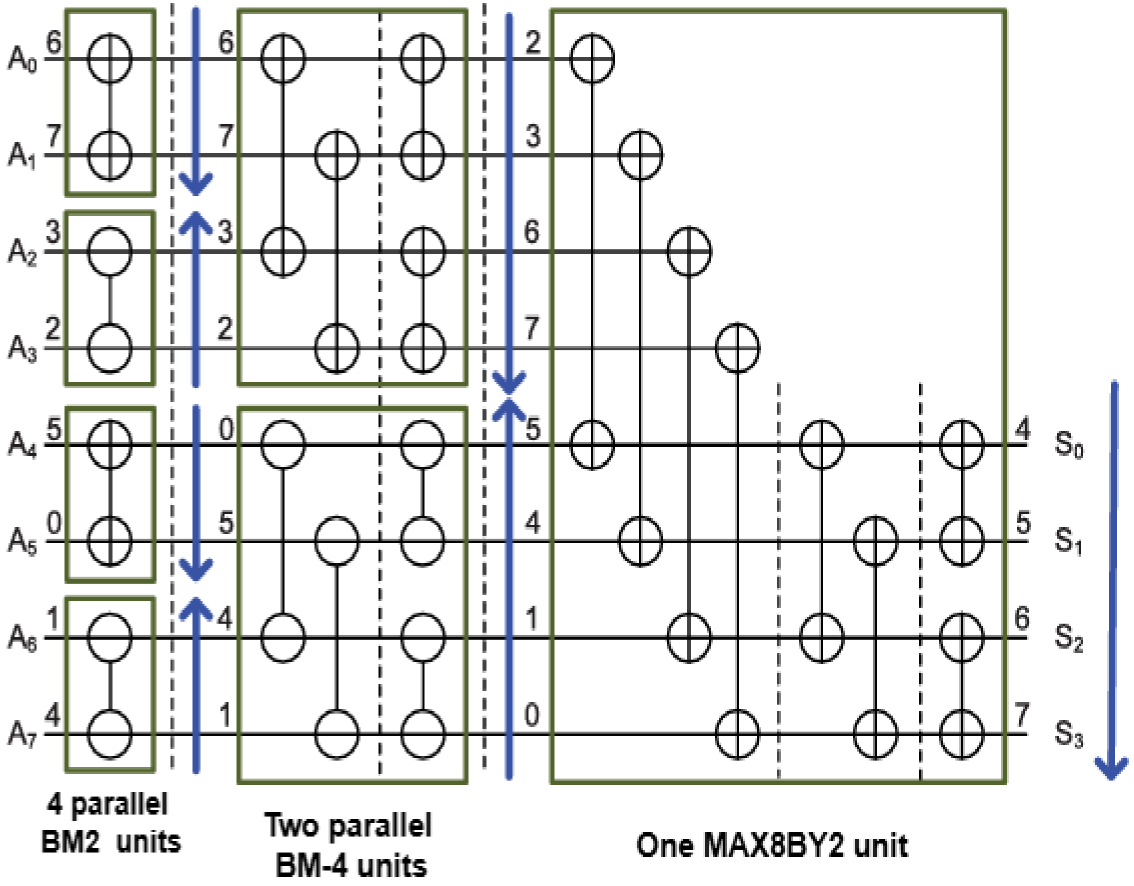
- In the complete **Bitonic Sorting** process, we utilize the previously described units to construct the full sorting block.

- As shown in the image, the system takes **8 random input numbers** and produces **8 sorted output numbers**.

- The structure includes:
  - **Four BM2 units** (4 × 2 = 8), arranged in **alternating ascending and descending order**
  - **Two BM4 units** (2 × 4 = 8), also used in an **alternating ascending/descending configuration**
  - **One BM8 unit** (1 × 8 = 8), which operates in **ascending order**

- Following this observed pattern, we developed a **general recursive implementation** for the Bitonic Sorting algorithm.

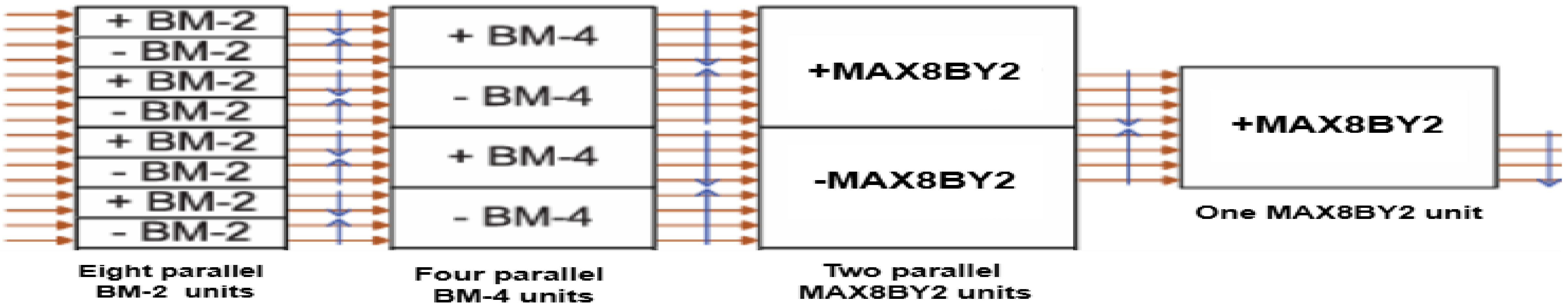
PARTIAL SORTING OF N-to-N/2:-



[Fig : - example of partial sorting of of 8-to-4 in which random unordered 8-inputs are getting 4-sorted output in ascending order]

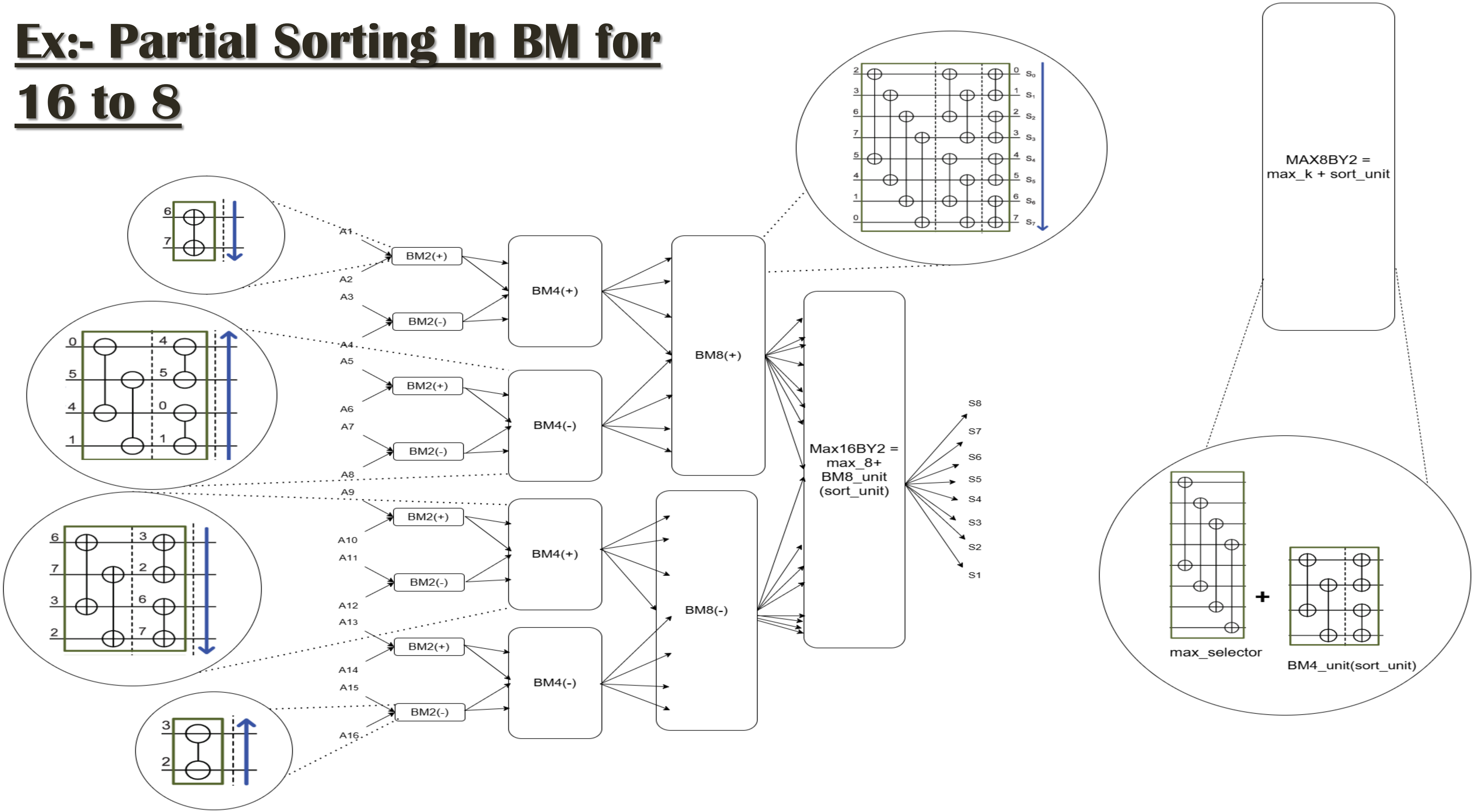
- The diagram beside illustrates **partial sorting from 8 to 4 (example of partial sorting of N-to-N/2)**.
- In this structure:
  - We use **four BM2 units** (4 × 2 = 8), arranged in **alternating ascending and descending order**.
  - Followed by **two BM4 units** (2 × 4 = 8), also in an **alternating pattern**.
- Finally, we use **one Max8by4 unit**, which outputs the **top 4 elements in ascending order**.
- This design is similar to the general Bitonic sorting structure; Concept is like when we reach the stage of **BM N/2 units**, After that we have to place it with a **MaxNby2 unit** to obtain the partially sorted output.

PARTIAL SORTING OF N-to-N/4:-



- The unit shown above demonstrates the implementation of **partial sorting**.
- It is composed of the following building blocks:
  - **Eight BM2 units** (8 × 2 = 16), arranged in **alternating ascending and descending order**
  - **Four BM4 units** (4 × 4 = 16), also in an **alternating pattern**
  - **Two Max8by2 units**, alternately configured in **ascending and descending order**
  - Followed by **one Max8by2 unit** in **ascending order**
- The pattern observed here closely resembles the structure of a **complete Bitonic sorting unit** for 16 elements. The key difference is introduced after the **BM4 stage**, which corresponds to **N/4** (where N = 16).
  - Instead of continuing with BM units, we insert:
  - **Two Max8by2 units** (configured alternately)
  - Followed by **one Max8by2 unit** in ascending order
  - This gives a total of **2 + 1 = 3 Max8by2 units**, where **2 = $\log_2(4)$** corresponds to the number of additional stages needed for partial sorting after the BM-N/4 level.

Ex:- Partial Sorting In BM for 16 to 8



Generalised PARTIAL SORTING For 2^m to 2^n (m>n) :-

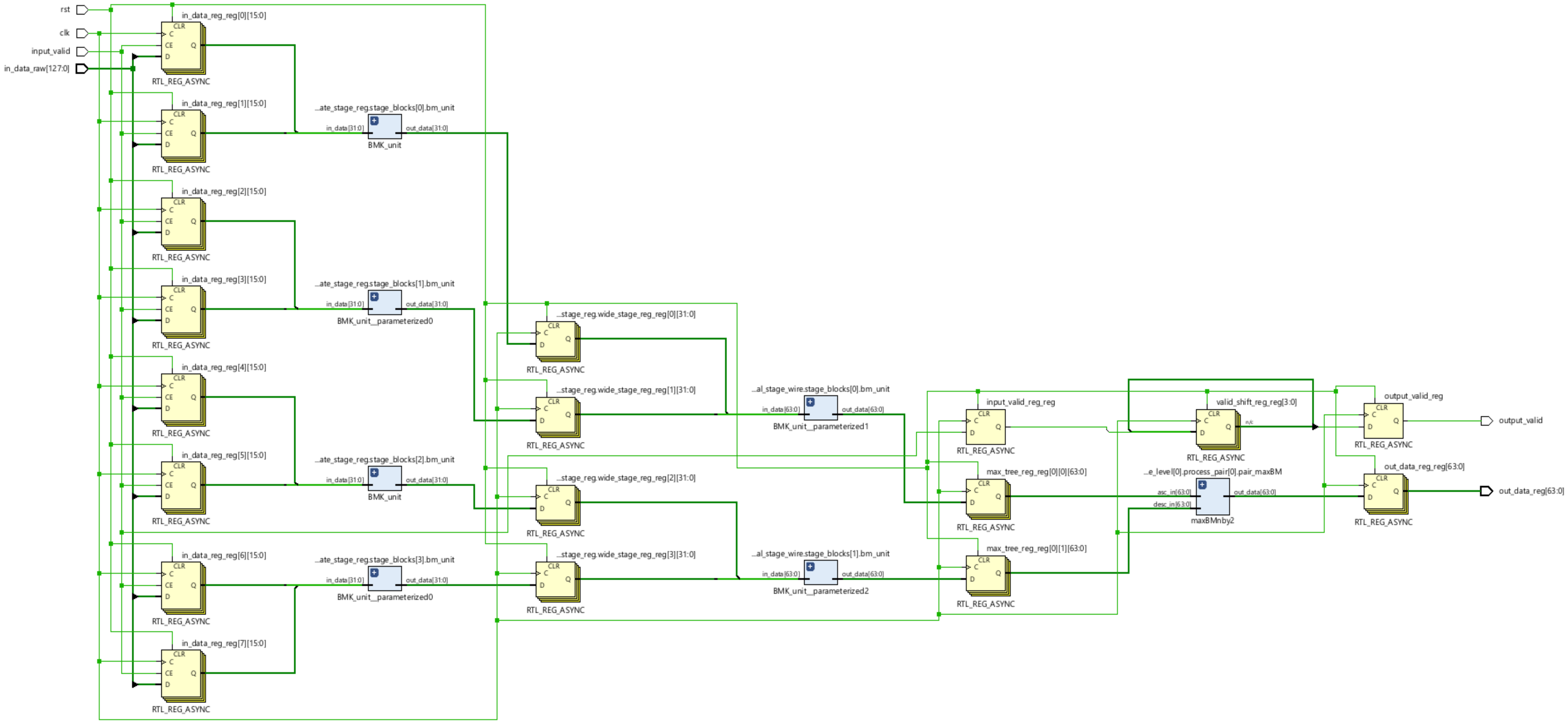
- By analyzing the patterns in **Partial Sorters** for sizes **N to N/2**, **N to N/4**, and **N to N/8**, we identified a generalized structure for a **Partial Sorter from** 2^n **to** 2^m where m<n

- The approach follows the structure of the complete **Bitonic Sorter for** 2^n until we reach the level of **BM-** 2^m units.

- At this point, the structure changes as follows:
  - We use **"m" BM-** 2^m **units**, arranged in **alternating ascending and descending order**
  - Next, we pair the outputs from each **ascending and descending BM-** 2^m **unit**
  - Each pair is then fed into a **Max**2^m **by2 unit**, which selects the top 2^{m-1} elements
  - These Max units are organized in a **tree structure**, where their outputs are further processed by additional Max2^m BY2 units
  - This continues recursively until only **one final** Max2^m BY2 unit remains

- The final Max unit is configured in **descending order** to produce the **partially sorted top** 2^m **elements** in descending order.

Schematic of partial sorting for 8-to-4 with pipelined :-



Comparision Table of different sorting units:-

Sorter Unit	DATA arrival Time in ns (Critical Path)	Area in μm^2	Power in mW
Sorter_8 _to_4	9.76	69440.582056	3.5054
Sorter_16_to_4	9.76	161175.697563	7.7992
Sorter_16_to_8	9.91	221422.279062	10.1341
Sorter_32_to_4	9.77	348220.652916	16.5151
Sorter_32_to_8	10.03	507333.673078	22.6277
Sorter_32_to_16	10.82	659107.498189	28.0751
Sorter_64_to_4	9.76	161175.697563	7.7992
Sorter_64_to_8	9.84	1107709.310906	46.8409
Sorter_64_to_16	10.93	1479133.758506	61.1747
Soter_64_to_32	13.72	1714610.439567	67.7443
Sorter_128_to_4	9.77	1499987.714066	71.1501
Sorter_128_to_8	10.01	2270082.935802	100.6285
Sorter_128_to_16	11.99	2968870.937471	125.9072

Continuing…

Sorter Unit	DATA arrival Time in ns (Critical Path)	Area in µm^2	Power in mW
Sorter_128 _to_32	14.73	3638295.210396	147.9505
Sorter_128_to_64	17.51	4327399.255973	168.4183
Sorter_256_to_4	9. 76	3050510.206716	144.9235
Sorter_256_to_8	11.02	4680408.079414	198.2483
Sorter_256_to_16	12.46	6057940.798922	256.0527
Sorter_256_to_32	16.40	7504066.388765	307.0547
Sorter_256_to_64	19.26	9131967.819051	354.1982
Sorter_256_to_128	21.51	1107709.310906	387.6980
Sorter_512_to_64	20.76	18481756.255650	680.7511
Sorter_512_to_128	25.38	22051140.217641	771.6448
Sorter_512_to_256	28.55	25604820.252861	871.6812
Sorter_1024_to_512	33.42	61102975.980396	1.9801e+03

Inference from Synthesis Results

M-Dominant Critical Path:

For fixed M, critical path remains stable even as N increases.

Example: $M=4 \Rightarrow$ 9.76 ns for N=64 to 256.

Due to pipelined design; final `maxBMnby2` unit dominates delay.

Scalability with M:

Critical path increases with M due to deeper comparison logic.

Example: Sorter_256_to_4: 9.76 ns, Sorter_256_to_32: 16.40 ns.

Resource Utilization:

Area and power increase with both N and M.

For fixed M: scales linearly with N (more comparators).

For fixed N: grows with M (wider data paths, complex logic).

Matches $\mathcal{O}(N \cdot \log^2 M)$ complexity.

FUTURE WORK :-

1. **Pipelining within BM-K and MaxNby2 Blocks:**
   Introduce pipelining inside the BM-K and MaxNby2 units to further reduce the **critical path delay**.
   A. Currently, the critical path depends on the **deepest MaxNby2** or **final BM-K block**.
   B. With pipelining, the critical path will primarily depend only on the **compare-and-swap blocks**, significantly improving performance.

2. **Index Tracking:**
   Attach an **index bit** to each input number to keep track of its **original position**.
   - This will help identify which index corresponds to the **maximum or selected values**, which is especially useful in applications like selection networks or sorting with traceability.

3. **Generalized Partial Sorter for Arbitrary N to 2^{m} (where N > 2^{m}):**
   Extend the current implementation to support **general partial sorting** from any integer **N to 2^{m}** , enabling more flexible and scalable designs.

THANK YOU