Accelerating Merge Sort on FPGAs: Design and Implementation Using HLS

@NecstLab Meeting Room – 17/06/2024

Claudio Di Salvo <claudio.disalvo@mail.polimi.it>

Introduction to FPGAs

- **Definition:** Field-Programmable Gate Array (FPGA) is an integrated circuit designed to be configured by the customer or designer after manufacturing.
- **Key Feature:** Reprogrammability: Unlike ASICs, FPGAs can be reprogrammed to perform different functions after deployment.
- **Components:** Comprised of an array of programmable logic blocks, I/O blocks, and a hierarchy of reconfigurable interconnects.

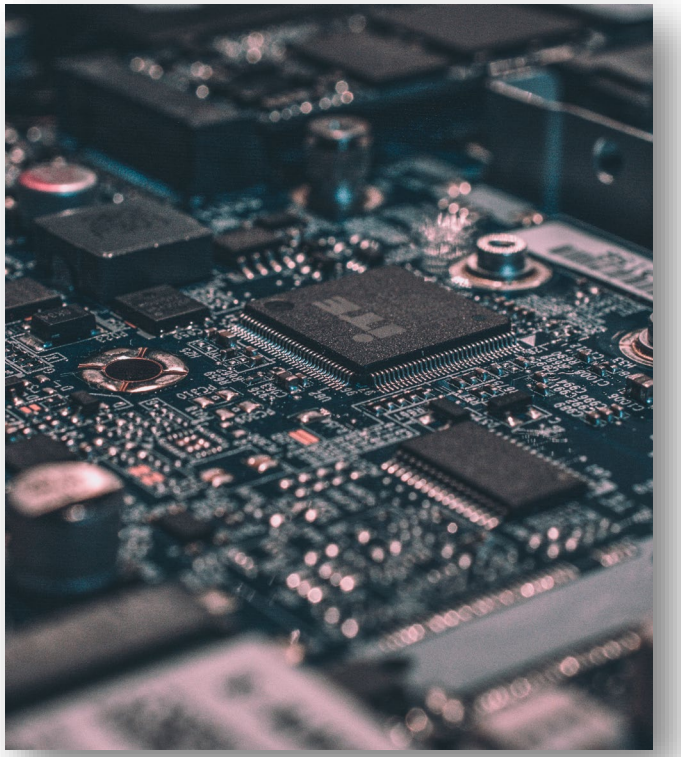

Foto di Alexandre Debiève su Unsplash

FPGA Architecture

- **Logic Blocks:** Configurable logic blocks (CLBs) containing lookup tables (LUTs), flip-flops, and multiplexers for implementing digital logic.
- **Interconnects:** A network of programmable interconnects that allows the CLBs to be connected in various configurations.
- **I/O Blocks:** Input/output blocks facilitate communication between the FPGA and external components.

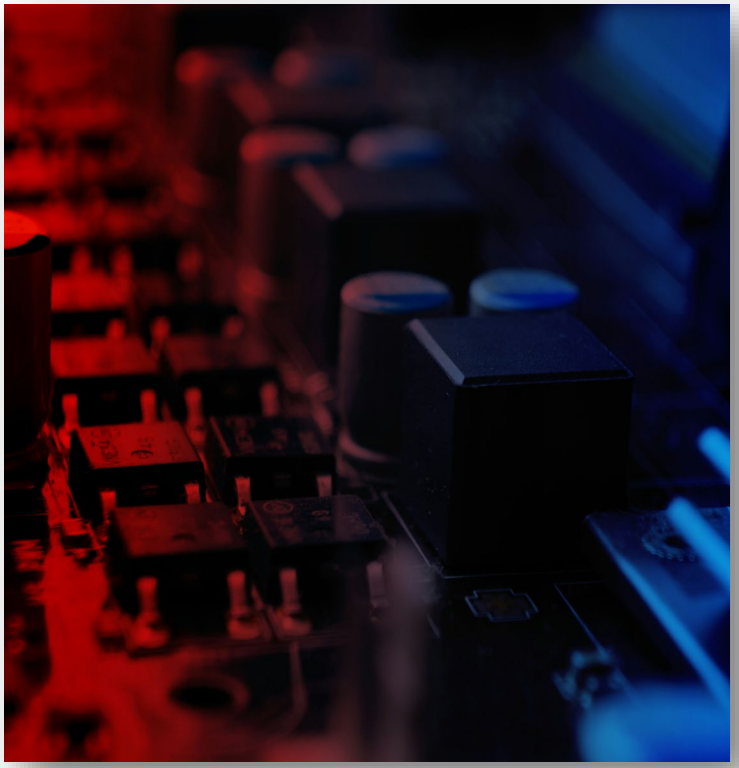

Foto di Алекс Арцибашев su Unsplash

Merge Sort

- **Definition:** Merge Sort is a highly efficient sorting algorithm based on the divide-and-conquer paradigm.
- **Implementation:** The iterative approach of Merge Sort is suitable for FPGA implementation, leveraging parallel processing capabilities.
- **Process:** The data structure is divided into subarrays, sorted individually, and merged iteratively until fully sorted.

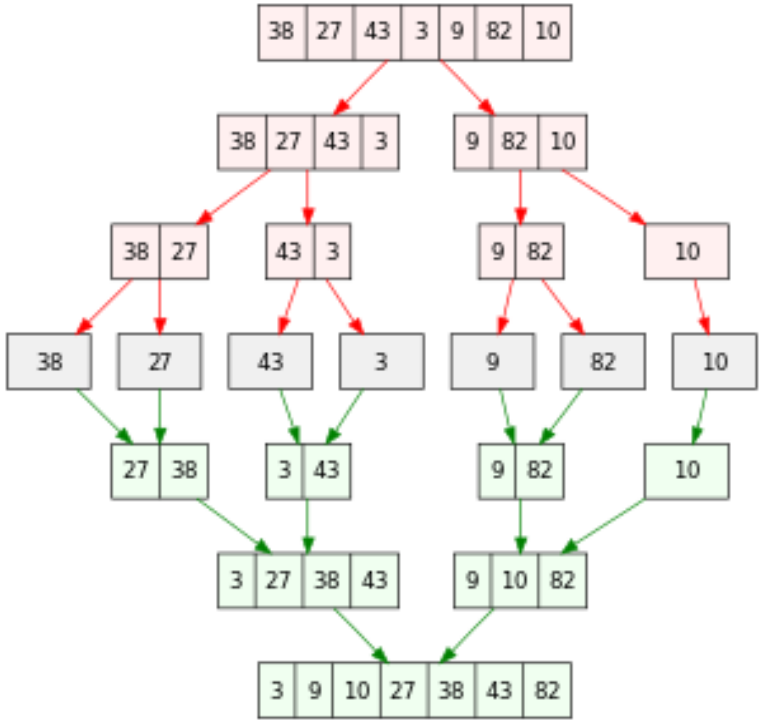

Foto di Wikipedia

Splitting Method

- **Overview:** This approach involves diving iteratively the array into smaller stream until them reach a single element stream.
- **Idea:** Once them are single stream, that means them are sorted and can be compared to each other and then merged. This approach has a backpropagation idea of the result.
- **Optimization:** Utilizing some pragma directive the code is optimized by the compiler in the synthesis part.

```
iteration:
for (int width = 1; width < size; width *= 2) {
    for (int i = 0; i < size; i += 2 * width) {
        int left_end = std::min(i + width, size);
        int right_end = std::min(i + 2 * width, size);
        left_right:
        for (int j = i; j < right_end; j++) {
        #pragma HLS PIPELINE II=1
            if (j < left_end) {
                left_stream.write(buffer[j]);
            } else {
                right_stream.write(buffer[j]);
            }
        }
        merge_sort_primitive(left_stream, right_stream, temp_stream);
        buffer_write:
        for (int j = i; j < right_end; j++) {
        #pragma HLS PIPELINE II=1
            buffer[j] = temp_stream.read();
        }
    }
```

Single Stream Approach

- **Overview:** This approach assumes the input stream originates from a single source.
- **Algorithm:** This algorithm leverage the splitting method previously enounced.
- **Performance:** Achieves significant speedup and efficient resource utilization, as evidenced by experimental results.
- This approach can be enhanced with futher optimization, such a better exploit of LUT to reduce the utilization.

SW Execution Time	HW Execution Time	Speedup (SW / HW)
0.100081	0.001850	54.094072
0.099518	0.001846	53.907659
0.098053	0.001767	55.493591
0.102514	0.001718	59.669026
0.099518	0.001762	56.490729
0.098169	0.001705	57.571449
0.102723	0.001680	61.148453
0.099497	0.001955	50.905221
0.097792	0.001708	57.262460
0.102454	0.001730	59.231289

Table 2: Execution Times and Speedup for single stream

Resource	Utilization	Available	Utilization %
LUT	14043	53200	26.40
LUTRAM	250	17400	1.44
FF	20202	106400	18.99
BRAM	2.5	140	1.79
BUFG	1	32	3.13

Table 1: Resource Utilization for single stream

Two Streams Approach

- **Overview:** This approach assume the input is coming from two different source.
- **Challenges:** Merging two different stream in parallel way can lead to a deadlock, since one of them can be not already fulfilled.
- **Workaround:** To avoid the deadlock a workaround was needed to be developed to sync the two stream.
- **Idea:** The idea behind this workaround is to copy the two streams into a temp ones.
- **Resource Utilization:** Higher resource demand compared to single stream approach, impacting efficiency.
- **Problems:** This solution seems to encounter gmem configuration problems.

Resource	Utilization	Available	Utilization %
LUT	38909	53200	73.14
LUTRAM	384	17400	2.21
FF	49974	106400	46.97
BRAM	6	140	4.29
BUFG	1	32	3.13

Table 3: Resource Utilization two streams approach

Conclusion

- **Key Findings:** Single Stream Approach demonstrated significant speedup and efficient resource utilization. Two Streams Approach encountered challenges with higher resource demand and potential deadlocks.
- **Future Work:** Refine the Two Streams Approach to mitigate resource utilization challenges and optimize the IP core for diverse applications

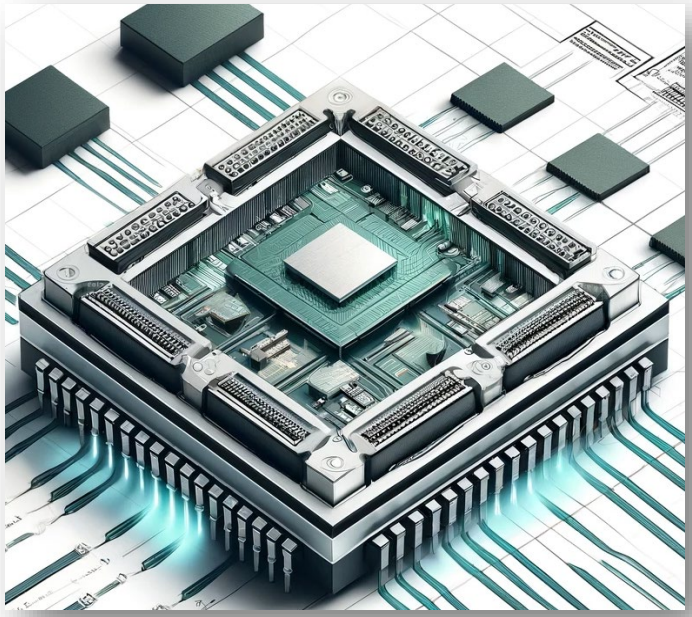

Foto di Dall-e

Thanks For Your Attention

Claudio Di Salvo
<claudio.disalvo@mail.polimi.it>