VHDL Merge Sort Algorithm Implementation on FPGA

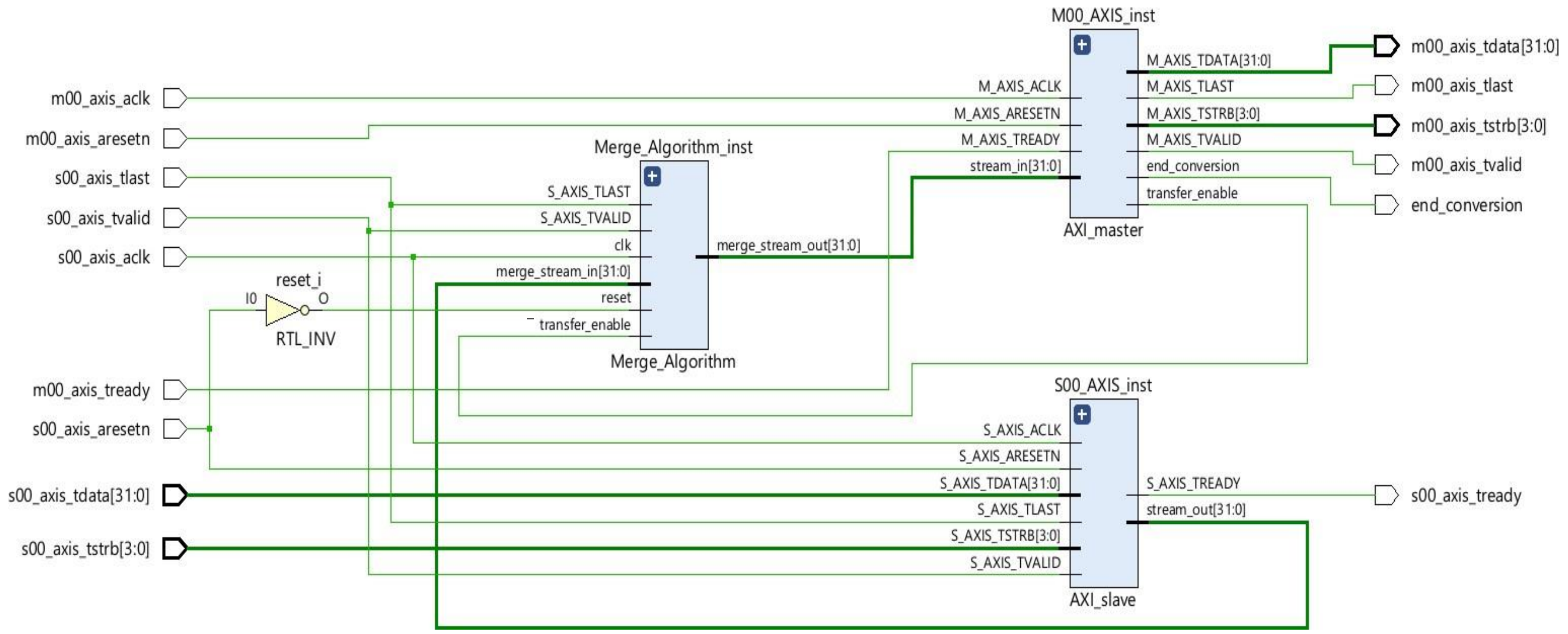
Marco La Barbera

February 2024

The algorithm

- **Splitting**: The input list is divided into smaller sub-list.
- **Sorting**: Each sub-list is sorted.
- **Merging**: Sorted sub-list are merged pairwise to produce larger sorted sub-list.
- **Combining**: Merged sub-list are recursively combined until the entire list is sorted.
- **Output**: The sorted list is produced.

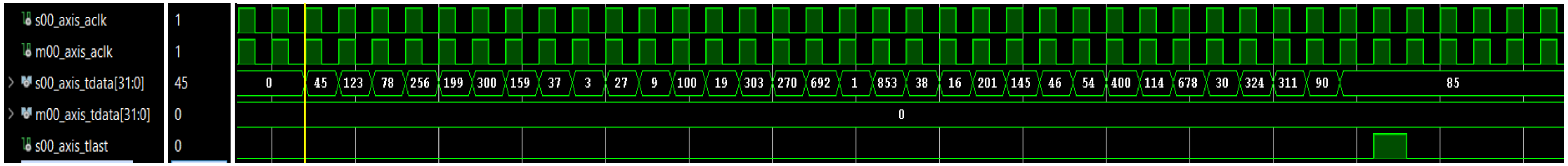
RTL and Implementation with AXI Stream Interface



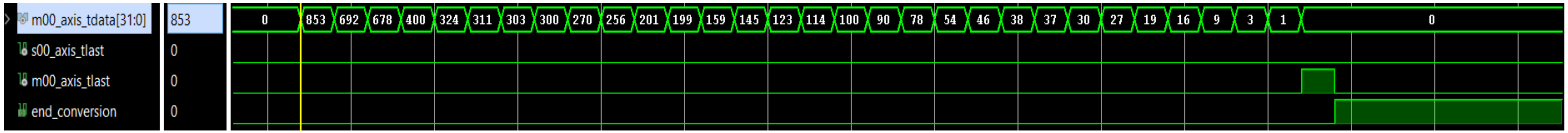
Algorithm 1 VHDL Merge-Sort Algorithm

```
Merge_Sort_algorithm : Process ( reset , array_in )
variable a,b : integer;
begin
if reset = '1' then
    array_out <= (Others => (Others => '0'));
else
    for i in 0 to (N_El - 1)/part loop
        a:= 0;
        b:= 0;
        for j in 0 to (part - 1) loop
            if b >= (part/2) then
                array_out((part)*i+j) <= array_in((part)*i+j);
            elsif a >= (part/2) and j >= (part/2) then
                array_out((part)*i+j) <= array_in((part)*i+j -(part/2));
            elsif array_in((part)*i+j-a) <= array_in((part)*i+j+(part/2)-b) then
                array_out((part)*i+j) <= array_in((part)*i+j-a);
                b := b + 1;
            elsif array_in((part)*i+j-a) > array_in((part)*i+j+(part/2)-b) then
                array_out((part)*i+j) <= array_in((part)*i+j+(part/2)-b);
                a := a + 1;
            end if;
        end loop;
     end loop;
end if;
end process;
```

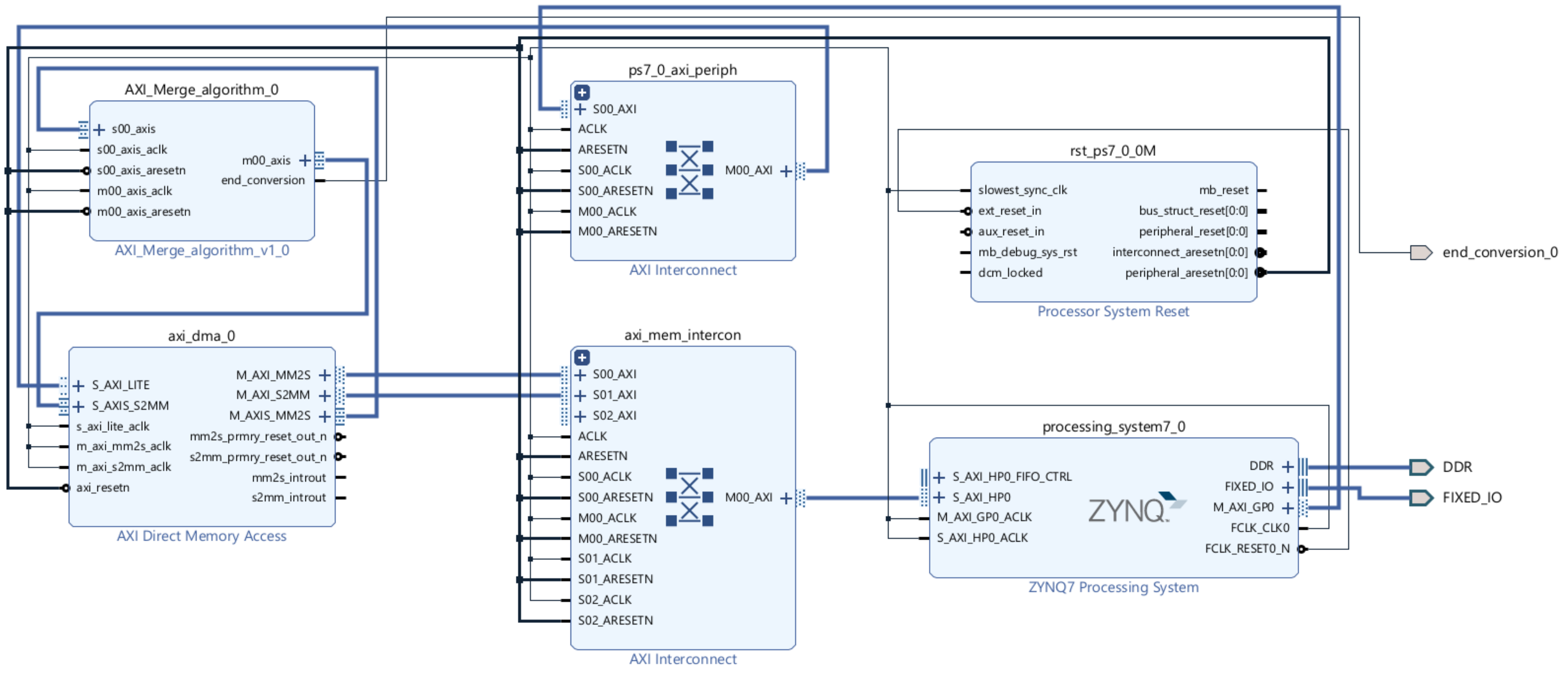
Slave - Input

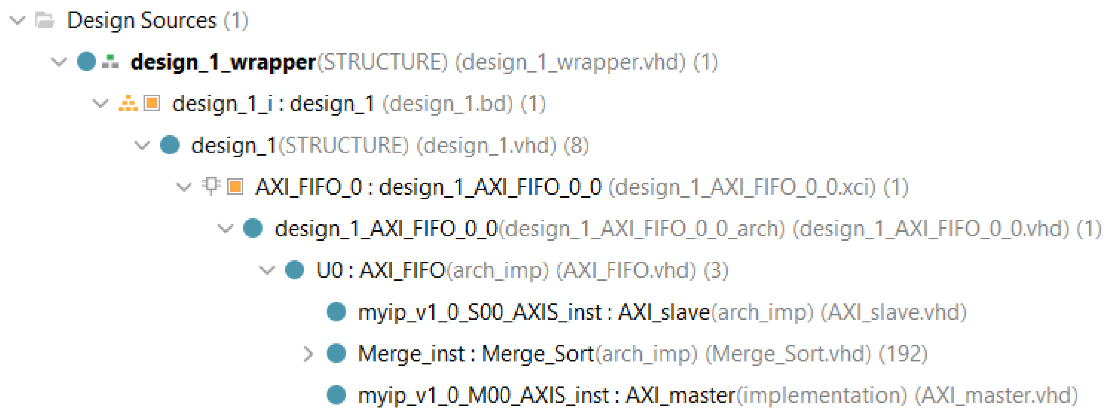


Master - Output



Block Design implementation

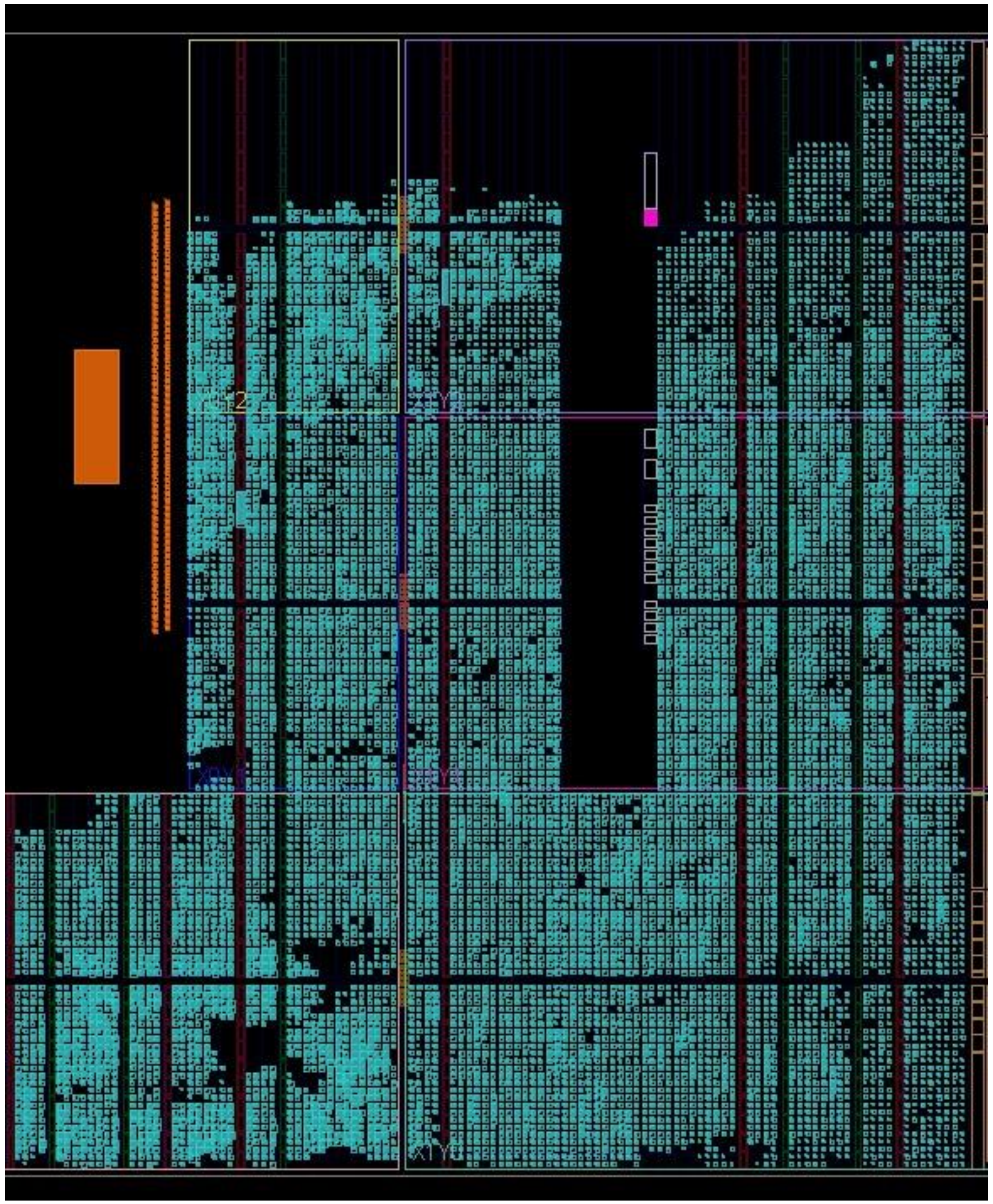




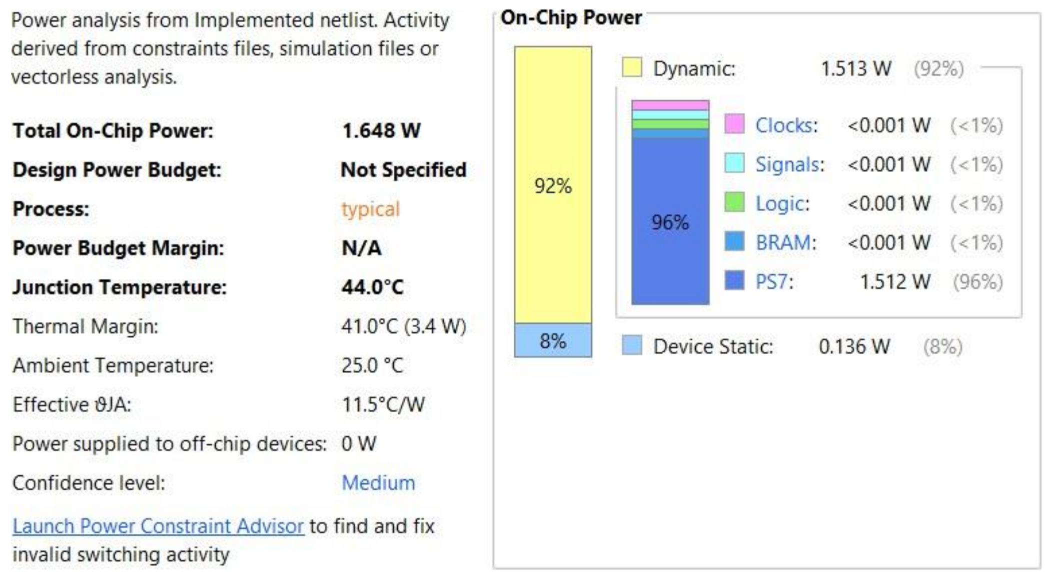
Design Sources (1)
- design_1_wrapper(STRUCTURE) (design_1_wrapper.vhd) (1)
  - design_1_i : design_1 (design_1.bd) (1)
    - design_1(STRUCTURE) (design_1.vhd) (8)
      - AXI_FIFO_0 : design_1_AXI_FIFO_0_0 (design_1_AXI_FIFO_0_0.xci) (1)
        - design_1_AXI_FIFO_0_0(design_1_AXI_FIFO_0_0_arch) (design_1_AXI_FIFO_0_0.vhd) (1)
          - U0 : AXI_FIFO(arch_imp) (AXI_FIFO.vhd) (3)
            - myip_v1_0_S00_AXIS_inst : AXI_slave(arch_imp) (AXI_slave.vhd)
            - Merge_inst : Merge_Sort(arch_imp) (Merge_Sort.vhd) (192)
            - myip_v1_0_M00_AXIS_inst : AXI_master(implementation) (AXI_master.vhd)

Resource Constraints and Optimization

With 32 numbers to sort, this algorithm reached the LUT limit (35.300/53.200), making it impossible to scale up to the required 1024 numbers.



Power Consumption


Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

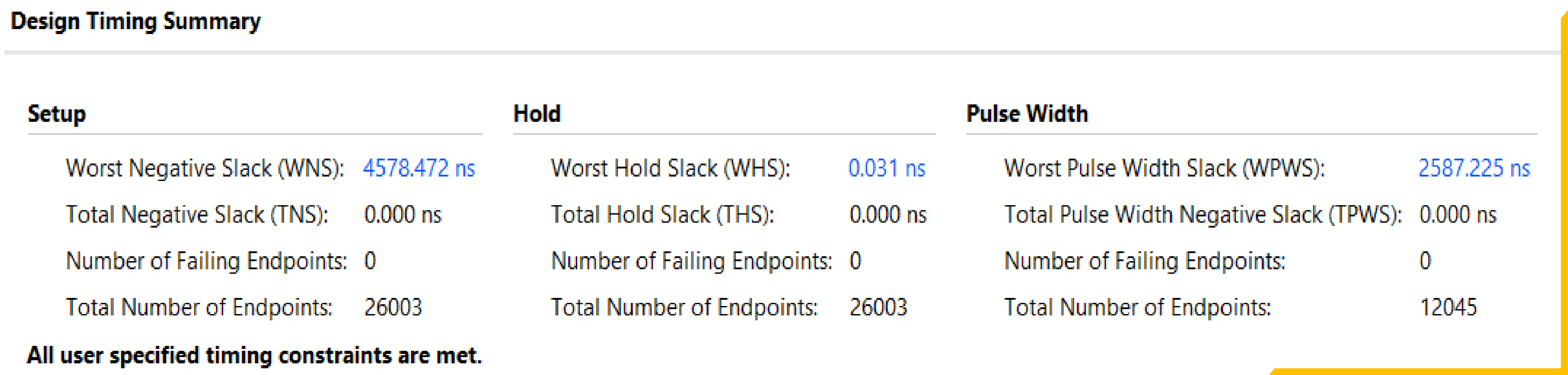
Total On-Chip Power:	**1.648 W**
Design Power Budget:	**Not Specified**
Process:	typical
Power Budget Margin:	**N/A**
Junction Temperature:	**44.0°C**
Thermal Margin:	41.0°C (3.4 W)
Ambient Temperature:	25.0 °C
Effective ϑJA:	11.5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

On-Chip Power

Dynamic:	1.513 W	(92%)	
Clocks:	<0.001 W	(<1%)	
Signals:	<0.001 W	(<1%)	
Logic:	<0.001 W	(<1%)	
BRAM:	<0.001 W	(<1%)	
PS7:	1.512 W	(96%)	
Device Static:	0.136 W	(8%)	

Timing requirements


Design Timing Summary

Setup

Worst Negative Slack (WNS):	4578.472 ns
Total Negative Slack (TNS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	26003

Hold

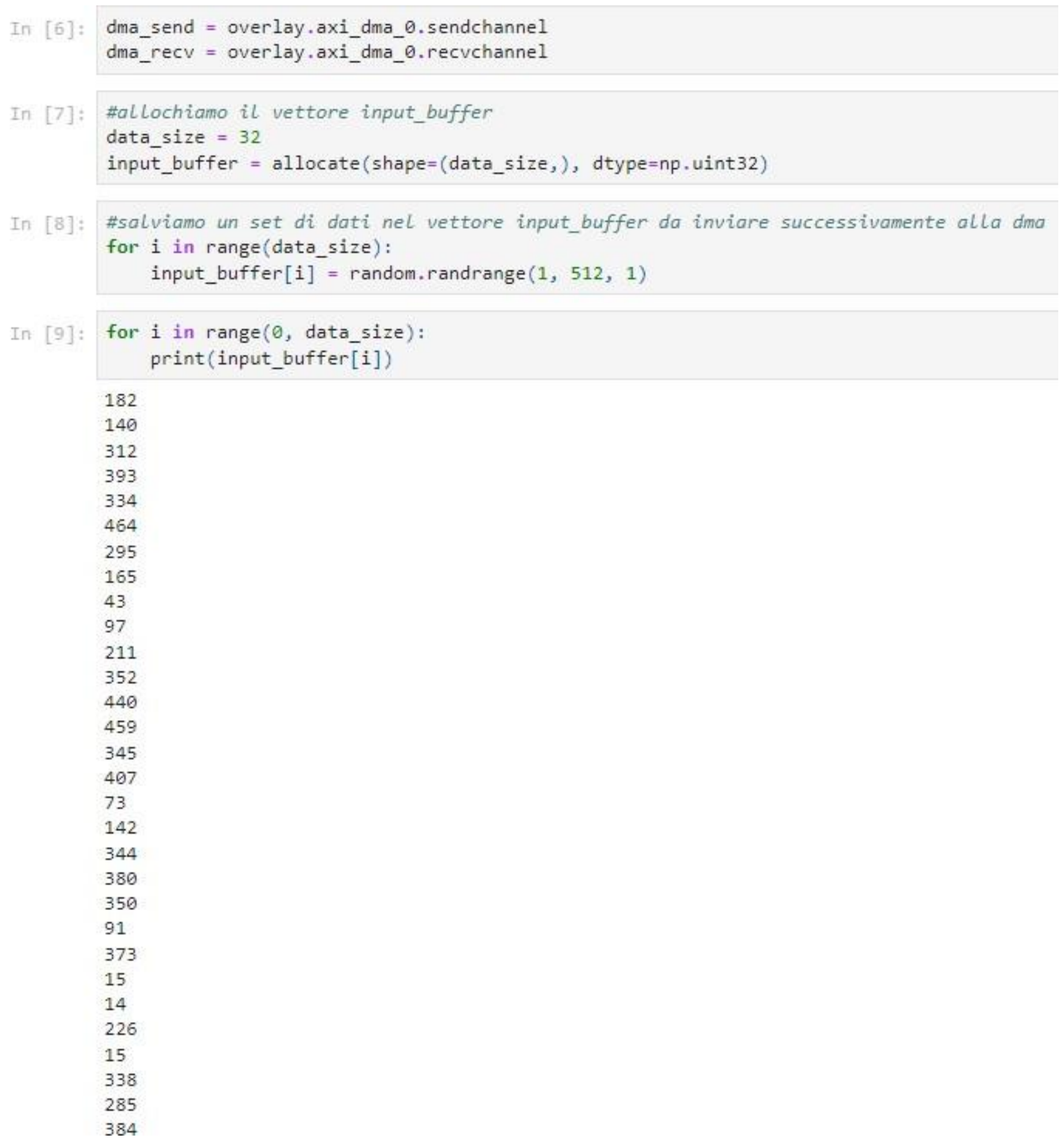
Worst Hold Slack (WHS):	0.031 ns
Total Hold Slack (THS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	26003

Pulse Width

Worst Pulse Width Slack (WPWS):	2587.225 ns
Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0
Total Number of Endpoints:	12045

All user specified timing constraints are met.

Jupiter Environment



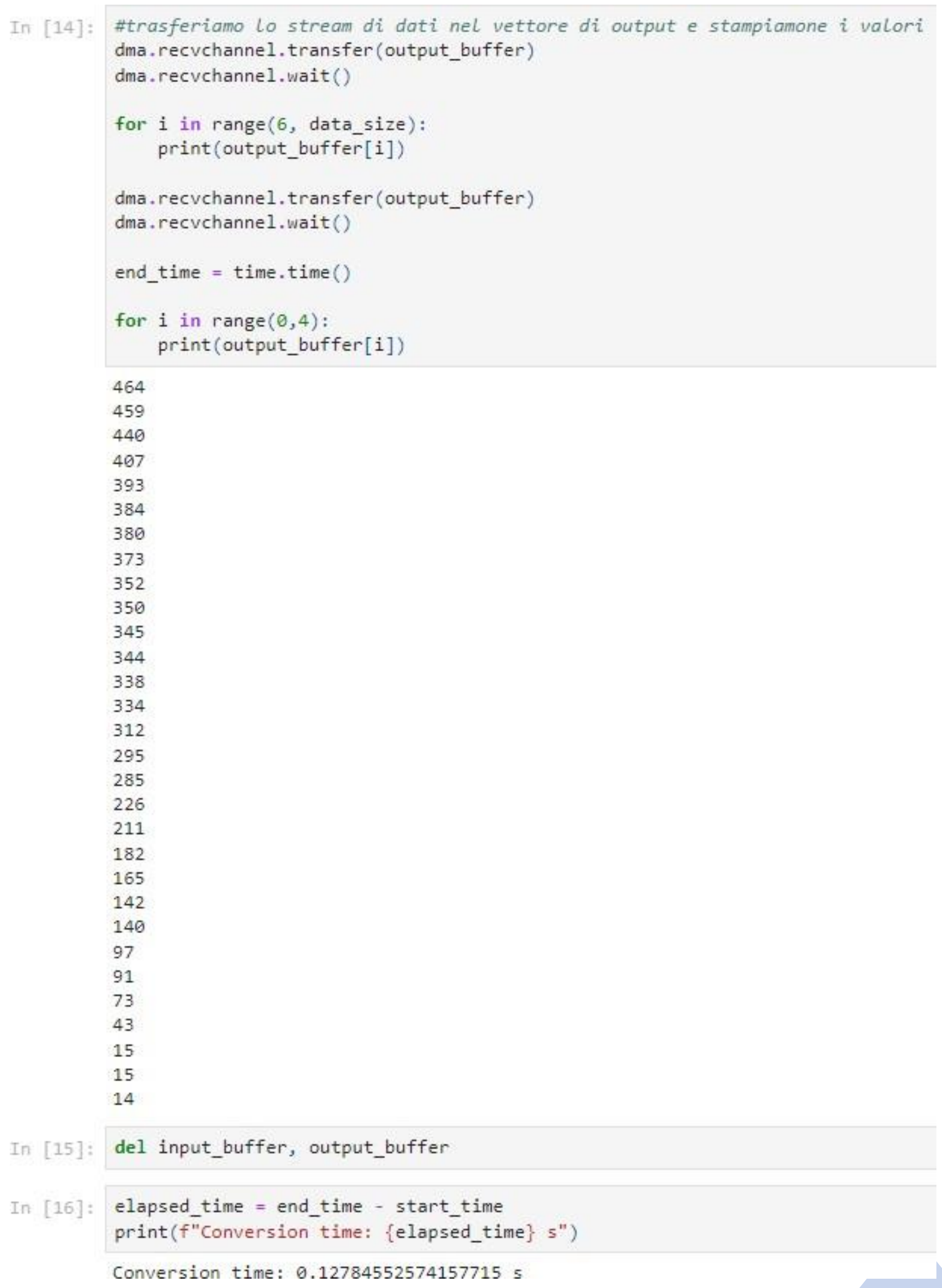
```
In [6]: dma_send = overlay.axi_dma_0.sendchannel
        dma_recv = overlay.axi_dma_0.recvchannel
```

```
In [7]: #allochiamo il vettore input_buffer
        data_size = 32
        input_buffer = allocate(shape=(data_size,), dtype=np.uint32)
```

```
In [8]: #salviamo un set di dati nel vettore input_buffer da inviare successivamente alla dma
        for i in range(data_size):
            input_buffer[i] = random.randrange(1, 512, 1)
```

```
In [9]: for i in range(0, data_size):
            print(input_buffer[i])
```

```
182
140
312
393
334
464
295
165
43
97
211
352
440
459
345
407
73
142
344
380
350
91
373
15
14
226
15
338
285
384
```



```
In [14]: #trasferiamo lo stream di dati nel vettore di output e stampiamone i valori
         dma.recvchannel.transfer(output_buffer)
         dma.recvchannel.wait()

         for i in range(6, data_size):
             print(output_buffer[i])

         dma.recvchannel.transfer(output_buffer)
         dma.recvchannel.wait()

         end_time = time.time()

         for i in range(0,4):
             print(output_buffer[i])
```

```
464
459
440
407
393
384
380
373
352
350
345
344
338
334
312
295
285
226
211
182
165
142
140
97
91
73
43
15
15
14
```

```
In [15]: del input_buffer, output_buffer
```

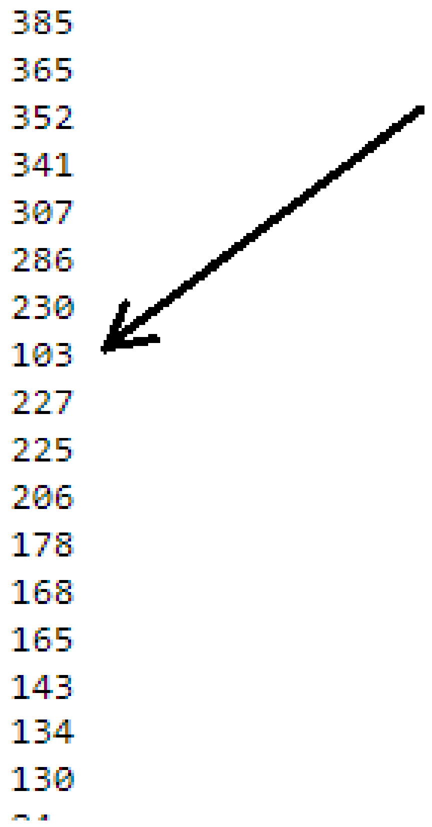
```
In [16]: elapsed_time = end_time - start_time
         print(f"Conversion time: {elapsed_time} s")
```

```
Conversion time: 0.12784552574157715 s
```

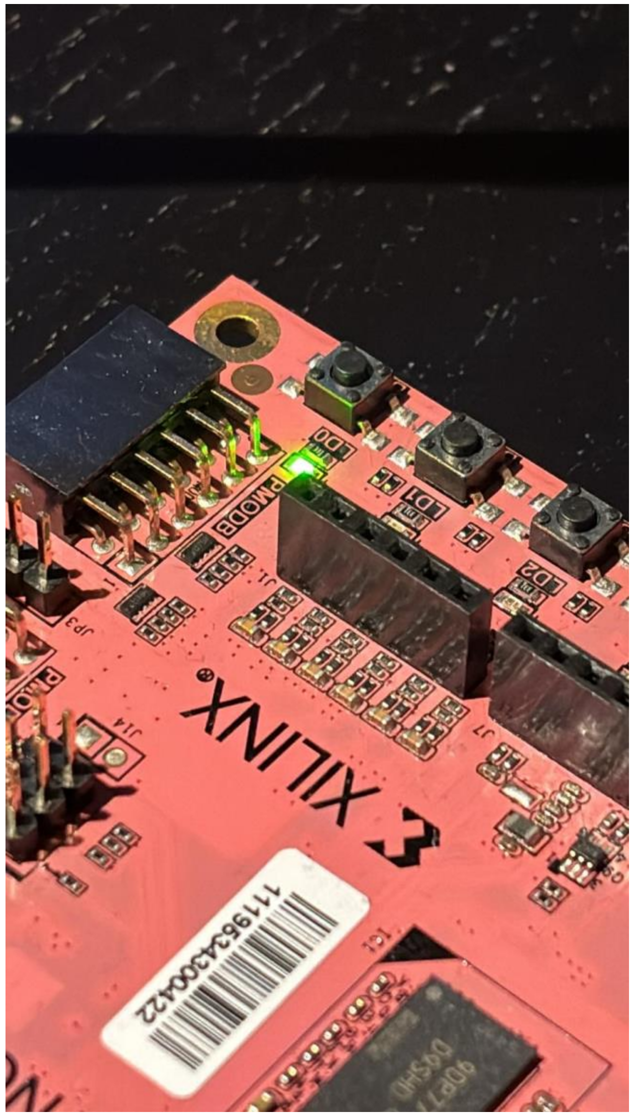
LED Indicator Implementation

To provide visual feedback on completion, an LED indicator system was implemented.

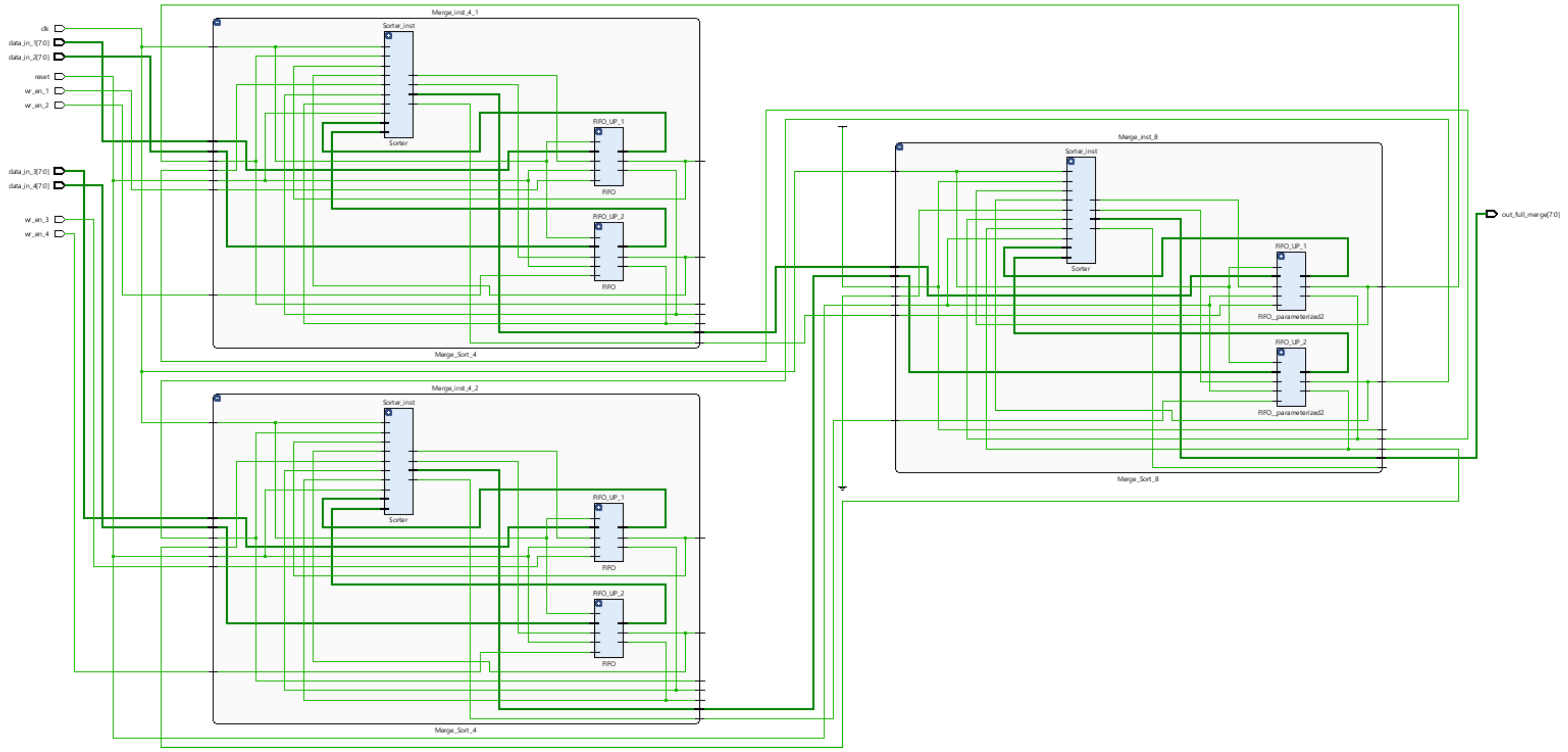
However, while printing the python output, few times a bug jumps out: one number is not correctly reordered, indicating a flaw in the implementation.



```
385
365
352
341
307
286
230
103
227
225
206
178
168
165
143
134
130
```



Conclusion – The «Merge Tree» Implementation


clk
data_in_1[7:0]
data_in_2[7:0]
reset
wr_en_1
wr_en_2
data_in_3[7:0]
data_in_4[7:0]
wr_en_3
wr_en_4
Merge_inst_4_1
Sorter_inst
Sorter
FIFO_UP_1
FIFO
FIFO_UP_2
FIFO
Merge_Sort_4
Merge_inst_4_2
Sorter_inst
Sorter
FIFO_UP_1
FIFO
FIFO_UP_2
FIFO
Merge_Sort_4
Merge_inst_8
Sorter_inst
Sorter
FIFO_UP_1
FIFO_parameterized2
FIFO_UP_2
FIFO_parameterized2
Merge_Sort_8
out_full_merge[7:0]