Design and Implementation of a Segmentation Adder for High-Speed and Low-Power Applications

Aldrin Infant Raj F
Electronics and Communication Engineering
Karunya Institute of Technology and Sciences
Coimbatore, Tamil Nadu, India
aldrininfant@karunya.edu.in

Gladwin Jebas J
Electronics and Communication Engineering
Karunya Institute of Technology and Sciences
Coimbatore, Tamil Nadu ,India
gladwinjebas@karunya.edu.in

Brijin G
of Electronics and Communication Engineering
Karunya Institute of Technology and Sciences
Coimbatore, Tamil Nadu, India
brijing@karunya.edu.in

***Abstract*—This work introduces the design, simulation, and implementation of a segmentation-based adder for enhancing computational speed and minimizing power consumption in arithmetic circuits. The design splits input operands into several segments to facilitate parallel computation and restrict carry propagation. The adder is designed in Verilog HDL and implemented on FPGA (Xilinx Artix-7, Basys3) and ASIC (Skywater 130nm through OpenLane). Simulation through GTKWave, along with synthesis, layout, and GDSII generation validated the design's timing and thermal dependability, along with functional accuracy. The segmented design, relative to conventional structures, shows higher power efficiency as well as scalability.**

***Index Terms*—Segmentation Adder, Verilog HDL, ASIC, FPGA, Sky130 PDK, OpenLane, Low Power, Digital Arithmetic**

I. Introduction

Addition operations are most frequent in arithmetic logic in high-speed digital systems. Conventional adder architectures, such as ripple-carry or carry-lookahead, are associated with delays and high power consumption. This paper introduces a new class of adders that divide numbers into small fragments, compute them in parallel, and minimize delays. The architecture is synthesized using free ASIC tools (OpenLane, Sky130 PDK) and FPGA tools (Vivado).

II. Design Architecture and Methodology

The Segmentation Adder is developed using a structured design process of register-transfer level (RTL) modeling, simulation, synthesis, and implementation on FPGA and ASIC platforms. The process follows traditional VLSI design processes to determine functional correctness, power, and scalability.

A. Architectural Overview

Traditional adder designs such as ripple carry, carry lookahead, and carry select provide various trade-offs among delay, power, and complexity. They possess long carry chains and high dynamic power with full-bit-width activity. The Segmentation Adder avoids these disadvantages by splitting the input operands into multiple segments and calculating partial sum in parallel. Each segment is an independent adder unit and possesses little or no carry, thereby reducing critical path delays and energy.

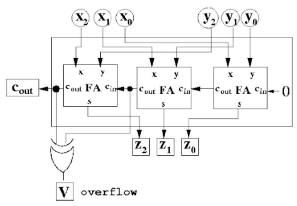


Fig. 1. 3-bit segmented adder with overflow detection

The Fig. 1 shows a 3-bit segmented adder with overflow detection. It adds two 3-bit numbers and generates a 3-bit sum and an overflow bit by XORing the carry-in and carry-out of the most significant bit.

III. Literature Review

A number of adder architectures have been proposed over the years to counter the speed and power limitations of the arithmetic blocks. Although the ripple-carry adder (RCA) is simple to design, it has large carry propagation delay and therefore is not suitable for high-speed applications. To counter this, carry lookahead adders (CLA) and carry select adders (CSA) have been proposed that offer improved delay performance but at the cost of increased area and complexity.

A. Concept of Segmentation

The segmentation adder breaks an n-bit binary adder into segments independent of each other. Each segment calculates a partial sum and carries information selectively to the subsequent segment, if required. This decreases the effective critical path delay and reduces switching activity, thus enhancing overall speed and power efficiency.

B. Benefits of Segmentation

The Segmented adders are scalable, reusable, and modular. They are more thermally stable as a result of localized activity and are configurable, thus making them appropriate for DSP and embedded systems.

The hardware description of the segmentation adder was written in Verilog HDL and simulated to verify its functionality. The design was synthesized using Xilinx Vivado Design Suite version 2018.2 targeting the Artix-7 FPGA device. The segmentation adder design was optimized to add two 8-bit operands and verify the overflow conditions. The operands were provided through the onboard switches of the board, and the result was shown through the onboard LEDs and seven-segment displays.

A. *Hardware Setup and Functional Testing*

The sixteen BASYS 3 board slide switches SW0–SW15 were used to input the operands, where SW[7:0] was used for the first operand (A) and SW[15:8] for the second operand (B). The 8-bit sum output was routed to LEDs LD0–LD7, and an extra LED (LD8) was utilized to indicate an overflow situation when the resulting sum exceeded the 8-bit range (greater than 255).

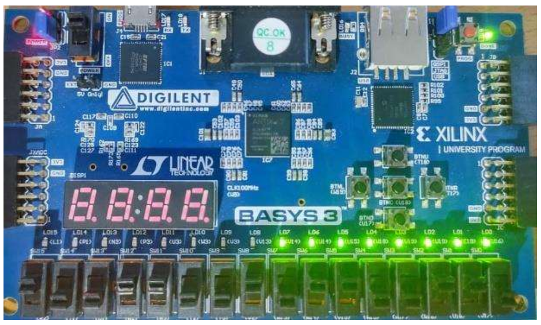


Fig. 1. BASYS 3 FPGA board showing segmentation adder addition of 9 and 6 (No overflow).

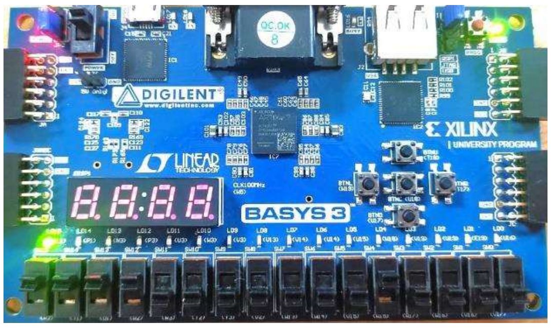


Fig. 2. BASYS 3 FPGA board showing segmentation adder addition of 240 and 16 (Overflow occurred).

Two classic experiments were conducted:

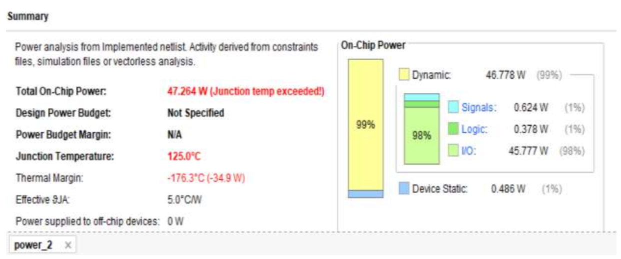
- In case 1, as can be observed from Fig. 1, operands 9 and 6 were added. The resulting sum of 15 is within the 8-bit limit, and overflow was not demonstrated.

- In the second case, as one will observe from Fig. 2, operands 240 and 16 were added. The sum of 256 was larger than the maximum value that can be represented in 8 bits and therefore set the overflow indicator LED high.

  The functional correctness of the design was thoroughly tested by hand with a large number of operand combinations, and sum and overflow conditions were properly reported and displayed.

B. *Power Analysis and Thermal Considerations*

After successful hardware verification, a detailed post-implementation power analysis was conducted utilizing the internal power estimation tool of Vivado. Estimation was done on the implemented netlist with default switching activity factors.


Summary

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

Total On-Chip Power:	47.264 W (Junction temp exceeded!)
Design Power Budget:	Not Specified
Power Budget Margin:	N/A
Junction Temperature:	125.0°C
Thermal Margin:	-176.3°C (-34.9 W)
Effective ϑJA:	5.0°C/W
Power supplied to off-chip devices:	0 W

On-Chip Power

Dynamic:	46.778 W	(99%)	
	Signals:	0.624 W	(1%)
	Logic:	0.378 W	(1%)
	I/O:	45.777 W	(98%)
Device Static:	0.486 W	(1%)	

Fig. 3. Vivado-generated post-implementation power report for the segmentation adder.

The summarized findings from the power report are presented in Fig. 3 and outlined below:

Parameter	Value
Total On-Chip Power	47.264 W
Dynamic Power	46.778 W (99%)
Static Power	0.486 W (1%)
Junction Temperature	125.0 °C
Thermal Margin	-176.3 °C
Dominant Dynamic Power Contributor	45.777 W

The dynamic power was found to be the predominant component, accounting for about 99% of the total on-chip power. Among the dynamic sources, the power consumed by the I/O blocks was considerable, owing mainly to the heavy toggling of the slide switches and LEDs while the circuit was in operation. The routing and logic resources utilized by almost negligible dynamic power is indicative of the lightweight computation-based nature of the segmentation adder.

The static power or leakage was 0.486 W, typical for low-end Artix-7 devices at ambient conditions. Nevertheless, the excessive junction temperature of 125.0 °C, well above the recommended operating limits of the FPGA, compounded thermal stress. The negative thermal margin of –176.3 °C also ensured that under continuous operation without active cooling provisions, the design would exhibit potential thermal reliability concerns.

These thermal results point toward the necessity for the use of power optimization techniques, such as clock gating, minimizing unnecessary switching, and, possibly, I/O drive strengths optimization to avoid thermal loads in future designs.

Fig. 4. Power Report of the segmentation Adder in optimized design.

The power analysis summary from the tool is displayed in Fig. 4. The digital circuit power consumption is analysed.

C. *Waveform Analysis*

The waveform is developed after analysing the power from the tool named GTKWAVE. It shows about how the values in input change at the intervals, and how the output and overflow flag respond to it.

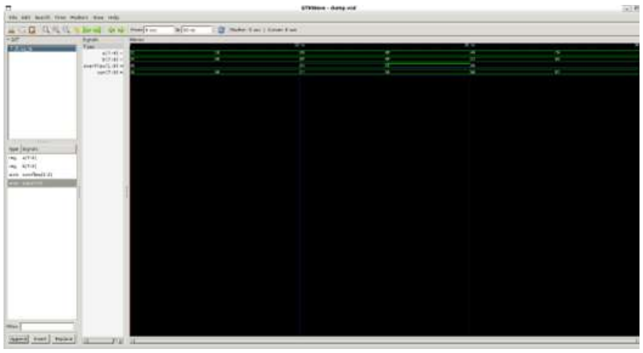


Fig. 5. Analysis of the waveform file generated using GTKwave.

The waveform which is generated from the tool is shown in Fig.5. It is mostly useful for debugging or it will be also helpful for verifying digital designs like the adders.

The first step used in the ASIC flow to synthesize the Carry Select Adder (CSA) is used by YOSYS. It is used to translate the high-level Verilog into netlist by using the standard cells which is from the SkyWater 130nm PDK.

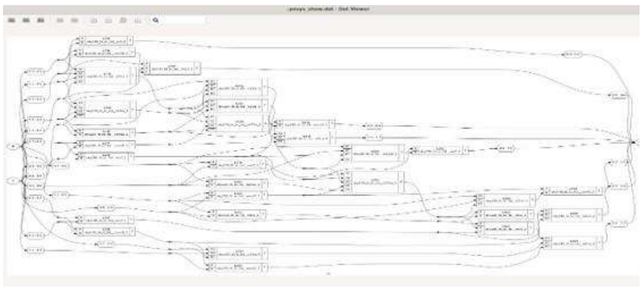
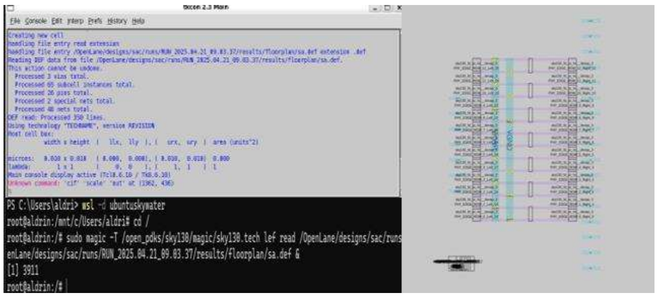


Fig. 6. Flow diagram generated by synthesis using YOSYS tool

It is created to show the complete synthesis and implementation process. Fig.6 presents the logic gate-level netlist visualization from the YOSYS Dot viewer. It is used for the design flow after the synthesis process. Each block represents the logic gate or the operations (AND, OR, XOR, etc.), and also the signal flow between them is represented by the arrow. Fig. 6 reveals about the process where the input bits are combined together and logic elements to generate the output. It mostly helps the designers to verify and understand the internal structure of synthesized Verilog modules.

Floor Planning



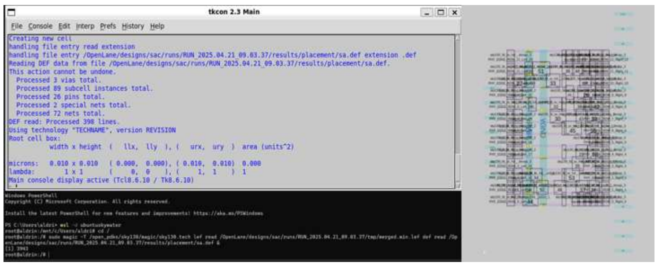
```
Creating new cell
handling file entry read extension
handling file entry /OpenLane/designs/sac/runs/RUN_2025.04.21_09.03.37/results/floorplan/sa.def extension .def
Reading DEF data from file /OpenLane/designs/sac/runs/RUN_2025.04.21_09.03.37/results/floorplan/sa.def.
This action cannot be undone.
  Processed 3 vias total.
  Processed 65 subcell instances total.
  Processed 26 pins total.
  Processed 2 special nets total.
  Processed 48 nets total.
DEF read: Processed 350 lines.
Using technology "TECHNAME", version REVISION
Root cell box:
          width x height  (   llx,  lly  ), (   urx,  ury  )  area (units^2)

microns:   0.010 x 0.010   ( 0.000,  0.000), ( 0.010,  0.010)  0.000
lambda:        1 x 1       (     0,  0    ), (     1,  1    )  1
Main console display active (Tcl8.6.10 / Tk8.6.10)
Unknown command: 'cif' 'scale' 'out' at (1362, 436)
```

```
PS C:\Users\aldri> wsl -d ubuntuskywater
root@aldrin:/mnt/c/Users/aldri# cd /
root@aldrin:/# sudo magic -T /open_pdks/sky130/magic/sky130.tech lef read /OpenLane/designs/sac/runs
enLane/designs/sac/runs/RUN_2025.04.21_09.03.37/results/floorplan/sa.def &
[1] 3911
root@aldrin:/#
```

Fig. 7. Floor Planning Process

The initial step in the physical design is Floor planning in which the basic layout of the chip will be defined. It also allocates the space for the core area, input/output ports, and also for defining the ground and power regions. Fig. 7 shows about the process of floor planning.



```
Creating new cell
handling file entry read extension
handling file entry /OpenLane/designs/sac/runs/RUN_2025.04.21_09.03.37/results/placement/sa.def extension .def
Reading DEF data from file /OpenLane/designs/sac/runs/RUN_2025.04.21_09.03.37/results/placement/sa.def.
This action cannot be undone.
  Processed 3 vias total.
  Processed 89 subcell instances total.
  Processed 26 pins total.
  Processed 2 special nets total.
  Processed 72 nets total.
DEF read: Processed 398 lines.
Using technology "TECHNAME", version REVISION
Root cell box:
          width x height  (   llx,  lly  ), (   urx,  ury  )  area (units^2)

microns:   0.010 x 0.010   ( 0.000,  0.000), ( 0.010,  0.010)  0.000
lambda:        1 x 1       (     0,  0    ), (     1,  1    )  1
Main console display active (Tcl8.6.10 / Tk8.6.10)
```

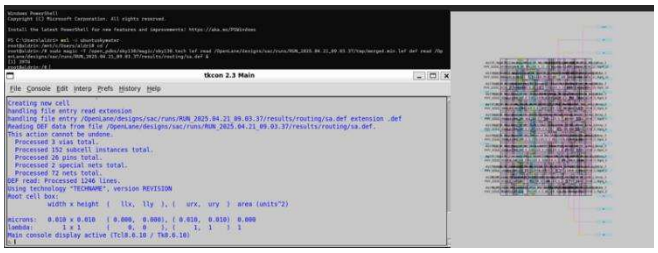
```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\aldri> wsl -d ubuntuskywater
root@aldrin:/mnt/c/Users/aldri# cd /
root@aldrin:/# sudo magic -T /open_pdks/sky130/magic/sky130.tech lef read /OpenLane/designs/sac/runs/RUN_2025.04.21_09.03.37/tmp/merged.min.lef def read /Op
enLane/designs/sac/runs/RUN_2025.04.21_09.03.37/results/placement/sa.def &
[1] 3943
root@aldrin:/#
```

Fig. 8. Placement Process

The next stage in the physical design is placement. In placement the standard cells from the synthesized netlist are assigned and fixed within the chip layout. It minimizes the timing delay and the length of the wire by ensuring the connectivity and efficient use of the chip area. Fig. 8 demonstrate about the process taken place in placement.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\aldri> wsl -d ubuntuskywater
root@aldrin:/mnt/c/Users/aldri# cd /
root@aldrin:/# sudo magic -T /open_pdks/sky130/magic/sky130.tech lef read /OpenLane/designs/sac/runs/RUN_2025.04.21_09.03.37/tmp/merged.min.lef def read /Op
enLane/designs/sac/runs/RUN_2025.04.21_09.03.37/results/routing/sa.def &
[1] 3974
root@aldrin:/#
```

```
Creating new cell
handling file entry read extension
handling file entry /OpenLane/designs/sac/runs/RUN_2025.04.21_09.03.37/results/routing/sa.def extension .def
Reading DEF data from file /OpenLane/designs/sac/runs/RUN_2025.04.21_09.03.37/results/routing/sa.def.
This action cannot be undone.
  Processed 3 vias total.
  Processed 152 subcell instances total.
  Processed 26 pins total.
  Processed 2 special nets total.
  Processed 72 nets total.
DEF read: Processed 1246 lines.
Using technology "TECHNAME", version REVISION
Root cell box:
          width x height  (   llx,  lly  ), (   urx,  ury  )  area (units^2)

microns:   0.010 x 0.010   ( 0.000,  0.000), ( 0.010,  0.010)  0.000
lambda:        1 x 1       (     0,  0    ), (     1,  1    )  1
Main console display active (Tcl8.6.10 / Tk8.6.10)
```

Fig. 9. Routing Process

The process done after the placement is Routing. It is the process of connecting the placed standard cells by using the metal layers and also to form the actual signal paths. It automatically create the necessary interconnects for the power, ground, ensuring all the designs are met. Fig. 9 reveals the process of the routing.

Final Physical layout

After the completion of the placement and routing process the final physical layout of the segmentation adder was generated. It shows all the interconnections, cells and the metal layers. The layout which is converted into .mag file by using the magic tool is used to view, edit and also to verify the physical layout.

The final output of the ASIC design flow is GDS (Graphic Data System) file. It is also close to fabrication design which contains all the information of the chip layout, including the layers, cells, and routing. It is the standard format for the fabrication of the chip.

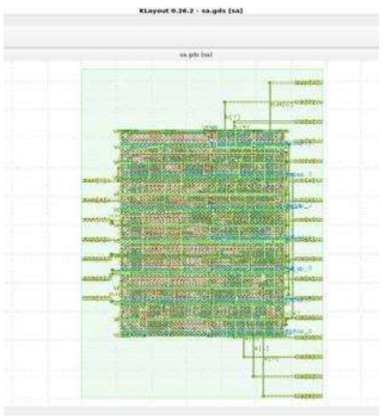


Fig. 10. GDS File diagram of the Segmentation Adder

KLayout is a powerful open-source layout viewer which is used to visualize and verify the GDS file. Fig. 10 demonstrates about the GDS file diagram.

IV. ACKNOWLEDGMENT

The Authors shows their heartfelt appreciation to the management of Karunya Institute of Technology and Sciences for providing the required resources and helping to complete the project to be successful. And a special thanks to the VLSI laboratory, Department of ECE, for giving the necessary facilities for carrying out this project.

V. REFERENCES

[1] D. M. Harris and S. L. Harris, *Digital Design and Computer Architecture*, 2nd ed. Burlington, MA, USA: Morgan Kaufmann, 2012.

[2] N. H. E. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. Boston, MA, USA: Pearson, 2011.

[3] SkyWater Technology Foundry, "Sky130 PDK Documentation." [Online]. Available: https://github.com/google/skywater-pdk.

[4] Efabless Corporation, "OpenLane: Open-source ASIC implementation flow." [Online]. Available: https://github.com/The-OpenROAD-Project/OpenLane.

[5] L. Clarke and S. Shukla, "Comparison of adder architectures for low power and high speed applications," in *Proc. IEEE Conf. VLSI Design*, 2020, pp. 1–6.

[6] R. P. Brent and H. T. Kung, "A regular layout for parallel adders," *IEEE Trans. Comput.*, vol. C-31, no. 3, pp. 260–264, Mar. 1982.

[7] YosysHQ, "Yosys Open SYnthesis Suite." [Online]. Available: https://yosyshq.net/yosys/.

[8] GTKWave, "GTKWave Waveform Viewer." [Online]. Available: http://gtkwave.sourceforge.net/.

[9] KLayout, "Layout Viewer for IC Design." [Online]. Available: https://www.klayout.de/.

[10] J. Han and M. Orshansky, "Approximate computing: An emerging paradigm for energy-efficient design," in *Proc. 18th IEEE Eur. Test Symp.*, 2013, pp. 1–6.

[11] J. Miao et al., "Modeling and synthesis of quality-energy optimal approximate adders," in *Proc. IEEE/ACM Int. Conf. Comput.-Aided Design (ICCAD)*, 2012, pp. 728–735.