



Design, Simulation and FPGA Implementation of an 8-bit Arithmetic and Logic Unit in System Verilog on the Boolean Board

Pooja Dhengale¹, Keshika Pillai², Siddhi Kini³, Anuradha Joshi⁴, Amaya Pethe⁵

¹ Department of Electronics and Telecommunication Vidyalankar Institute of Technology, Mumbai pooja.dhengale@vit.edu.in

² Department of Electronics and Telecommunication Vidyalankar Institute of Technology, Mumbai keshika.pillai@vit.edu.in

³ Department of Electronics and Telecommunication Vidyalankar Institute of Technology, Mumbai siddhi.kini@vit.edu.in

⁴ Department of Electronics and Computer Science Vidyalankar Institute of Technology, Mumbai anuradha.joshi@vit.edu.in

⁵ Department of Electronics and Computer Science Vidyalankar Institute of Technology, Mumbai amaya.pethe@vit.edu.in

ABSTRACT—

This paper presents the design, verification and hardware realisation of an 8-bit Arithmetic and Logic Unit (ALU) written in SystemVerilog and implemented on a Xilinx Spartan-7 (XC7S50csga324-1) FPGA using the Boolean development board. The ALU executes eight operations selected by a 3-bit control code: addition, subtraction, bitwise AND, OR, XOR, NOT, logical shift left and logical shift right. Two status flags, Carry and Zero, are generated alongside the 8-bit result. A combinational core module (alu_8bit) is wrapped by a board-level top module (alu_top) that maps the operands to 16 slide switches, the operation code to three push buttons, and the result and flags to on-board LEDs. Functional correctness is first established with a self-contained testbench in Vivado behavioural simulation, after which the design is synthesised, implemented, and programmed to the board through a bitstream, where every operation is exercised and observed in real time. Unlike purely simulation-based ALU studies, this work closes the loop from RTL to working silicon and documents the constraint mapping, elaborated and implemented netlists, and hardware test results, providing a reproducible reference for students and designers.

Keywords—Arithmetic Logic Unit, FPGA, SystemVerilog, Spartan-7, Vivado, Boolean board, digital design.

1. INTRODUCTION

The Arithmetic and Logic Unit (ALU) is the computational heart of every processor and digital signal-processing datapath: it performs the integer arithmetic and bitwise logic operations on which higher-level instructions are built [1], [5]. Because of its regular structure and the ease with which it can be described in a hardware description language (HDL), the ALU is the canonical first design that students and engineers implement when learning RTL design and FPGA prototyping.

Field-programmable gate arrays (FPGAs) allow such a design to be tested on real hardware within minutes, without the cost and turn-around time of an ASIC flow. Modern low-cost boards expose switches, buttons and LEDs that make it possible to drive and observe a small datapath directly, giving immediate physical feedback on the behaviour of the RTL.

A number of published works describe Verilog ALUs and verify them through waveform simulation and post-synthesis reports [2], [3], [4]. In most cases the result is a simulation waveform and a resource or power report; the design is not exercised on a physical device. This paper takes the additional step of implementing an 8-bit ALU on an FPGA board, mapping its interface to physical I/O through a constraints file, generating a bitstream, and verifying each operation on the hardware.

The main contributions of this work are:

- a synthesizable 8-bit, 8-operation SystemVerilog ALU with Carry and Zero flags;
- a board-level top module and XDC constraint set that make the ALU fully operable using only the switches, buttons and LEDs of the Boolean board;
- a complete verification flow—behavioural simulation, RTL elaboration, synthesis, implementation and on-board testing—with all intermediate artefacts documented; and
- a comparison against a simulation-only reference design, highlighting the practical considerations that arise during FPGA realisation.

The remainder of the paper is organised as follows. Section II reviews related work. Section III describes the ALU architecture and the operation set. Section IV presents the FPGA implementation and I/O mapping. Section V describes the verification methodology, Section VI presents the simulation and FPGA hardware results for each operation, Section VII discusses resources and limitations, and Section VIII concludes.

2. RELATED WORK

Surekha *et al.* [1] describe the design and implementation of an ALU with emphasis on the arithmetic and logic building blocks. Huang [2] presents an 8-bit ALU based on Verilog HDL and evaluates its correctness through simulation. The most closely related work to this paper is the Verilog 8-bit ALU of Hemanth *et al.* [3], which supports addition, subtraction, multiplication, AND, OR, NAND, NOR and XOR, produces a 16-bit result with carry and zero flags, and reports simulation waveforms together with power (10.397

W, predominantly dynamic) and utilisation figures (139 slice LUTs, 37 bonded IOBs) obtained from Vivado. Wider datapaths have also been explored; for instance, Brahmaiah *et al.* [4] present an optimised 32-bit ALU.

The present work differs from these studies in three respects. First, it targets a specific commercial FPGA board and reports the complete path from RTL to a programmed device. Second, the operation set is chosen for a compact 8-bit datapath and includes unary operations (NOT, shifts) that are convenient to demonstrate on a switch-and-LED interface. Third, the paper documents the pin-level constraint mapping and the physical operation of the design, which are typically omitted in simulation-only reports. Table I summarizes the comparison.

TABLE I

COMPARISON WITH THE REFERENCE ALU [3]

Feature	Reference [3]	This work
Operand width	8-bit	8-bit
Result width	16-bit	8-bit + flags
Operations	ADD, SUB, MUL, AND, OR, NAND, NOR, XOR	ADD, SUB, AND, OR, XOR, NOT, SHL, SHR
Flags	Carry, Zero	Carry, Zero
HDL	Verilog	SystemVerilog
Validation	Simulation	Simulation + FPGA hardware
Target	Not on hardware	Boolean board
I/O mapping	None	Switches, buttons, LEDs (XDC)

3. ALU ARCHITECTURE

A. Interface and Operation Set

The `alu_8bit` module has two 8-bit operand inputs, A and B, a 3-bit operation select `ALU_Sel`, an 8-bit result `ALU_Out`, and two single-bit flags, Carry and Zero. The design is purely combinational: the output changes whenever an input changes, so no clock is required. The operations are encoded as listed in Table II.

TABLE II

ALU OPERATION CODES

Sel	Operation	Expression	Carry
000	ADD	$A + B$	carry-out
001	SUB	$A - B$	$A \geq B$
010	AND	$A \& B$	0
011	OR	$A B$	0
100	XOR	$A \wedge B$	0
101	NOT	$\sim A$	0

Sel	Operation	Expression	Carry
110	SHL	A << 1	0
111	SHR	A >> 1	0

B. Flag Generation

For addition, both operands are zero-extended to 9 bits and summed; the lower 8 bits form the result and bit 8 is the carry-

out. For subtraction, the result is $A - B$ modulo 256 and the Carry output is asserted when $A \geq B$, i.e. it acts as a *no-borrow* indicator. For all logical and shift operations Carry is held at 0. The Zero flag is derived from the final result: it is asserted whenever ALU_Out equals 8'h00, independent of the selected operation.

C. Core Description

The core is written as an `always_comb` block with a case statement on `ALU_Sel`, as outlined below. Default assignments at the top of the block prevent inferred latches, and a default branch covers unused encodings.

```
always_comb begin
    ALU_Out = 8'b0; Carry = 1'b0; case (ALU_Sel)
        3'b000: begin
            temp = {1'b0,A} + {1'b0,B};
            ALU_Out = temp[7:0]; Carry = temp[8];
        end
        3'b001: begin ALU_Out = A - B;
            Carry = (A >= B); end
        3'b010: ALU_Out = A & B;
        3'b011: ALU_Out = A | B;
        3'b100: ALU_Out = A ^ B;
        3'b101: ALU_Out = ~A;
        3'b110: ALU_Out = A << 1;
        3'b111: ALU_Out = A >> 1;
        default: ALU_Out = 8'b0; endcase
    Zero = (ALU_Out == 8'b0); end
```

Listing 1. Core of the alu_8bit module.

Fig. 1 shows the RTL schematic produced by Vivado elaboration (13 cells, 127 nets). The synthesiser maps the eight operations to an adder, subtractor, AND, OR and XOR gates, an inverter, two constant shifters and a comparator (for the subtract-carry), followed by two 8:1 multiplexers that select the result and the Carry output, and a zero-detect block that drives the Zero flag.

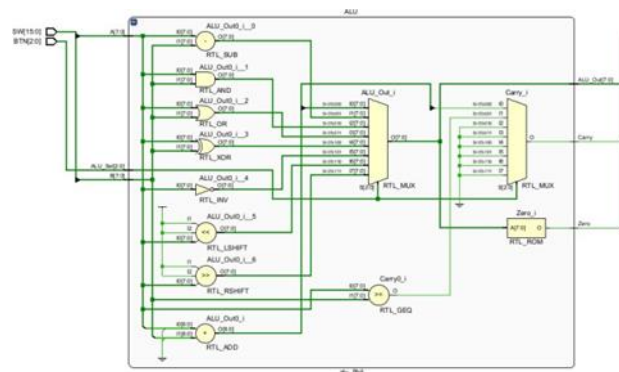


Fig. 1. Elaborated RTL schematic of the alu_8bit module (Vivado 2024.1).

FPGA IMPLEMENTATION

A. Target Platform

The design is implemented on the Boolean board from Real Digital, which is built around an AMD/Xilinx Spartan-7 XC7S50-CSGA324-1 FPGA [7], [9]. The board provides 16 slide switches, 16 discrete LEDs, push buttons, eight-digit seven-segment displays and a USB-JTAG programming interface, as visible in Fig. 2. Only the switches, three push buttons and the LEDs are used in this design.



Fig. 2. Boolean board (Spartan-7 XC7S50) used for prototyping the ALU.

B. Top-Level Module and I/O Mapping

The top module `alu_top` connects the physical I/O to the ALU as follows. Switches `SW[15:8]` supply operand A and `SW[7:0]` supply operand B. The operation select `ALU_Sel` is driven directly by the three push buttons `BTN[2:0]`; with no button pressed the select is 000 (ADD), and pressing a combination of buttons selects the corresponding code of Table II. The 8-bit result is displayed on `LED[7:0]`, the Carry flag on `LED[8]` and the Zero flag on `LED[9]`; `LED[15:10]` are tied low. Because the design is combinational, no clock, reset or debouncing logic is needed—the LEDs simply follow the switches and buttons. The connection is shown in Fig. 4 and the pin assignments in Table III.

TABLE III

I/O MAPPING AND PIN CONSTRAINTS (LVCMOS33)

Signal	Board resource	FPGA pins
A[7:0]	SW[15:8]	K1, K2, L1, M1, M2, N1, N2, P1
B[7:0]	SW[7:0]	P2, R1, R2, T1, T2, U1, U2, V2
ALU_Sel[2:0]	BTN[2:0]	H2, J5, J2 (BTN2..0)
ALU_Out[7:0]	LED[7:0]	E5, E3, E2, E1, F2, F1, G2, G1
Carry	LED[8]	E6
Zero	LED[9]	C3

All 35 ports use the LVCMOS33 standard at 3.3 V; the switches are placed in I/O bank 34 and the LEDs and buttons in bank 35, and each pin is fixed by the XDC file through `set_property PACKAGE_PIN ... IOSTANDARD LVCMOS33`.

C. Design Flow

The design was developed in Vivado 2024.1 and passed through the standard flow [6]: (i) behavioural simulation of `alu_8bit_tb`; (ii) RTL elaboration and schematic inspection; (iii) synthesis, followed by I/O planning against the XDC constraints; (iv) implementation (placement and routing); and (v) bitstream generation and device programming through the Hardware Manager. Vivado reported `write_bitstream Complete` for the final design. After implementation the netlist comprises 94 cells and 159 nets for 35 I/O ports (Fig. 3), reflecting technology mapping of the RTL into LUTs and I/O buffers.

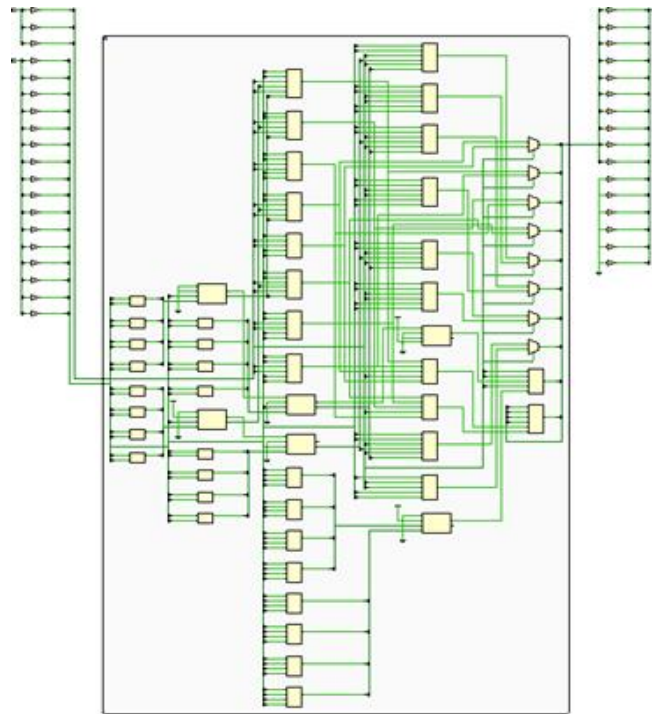


Fig. 3. Implemented design schematic (94 cells, 35 I/O ports, 159 nets).

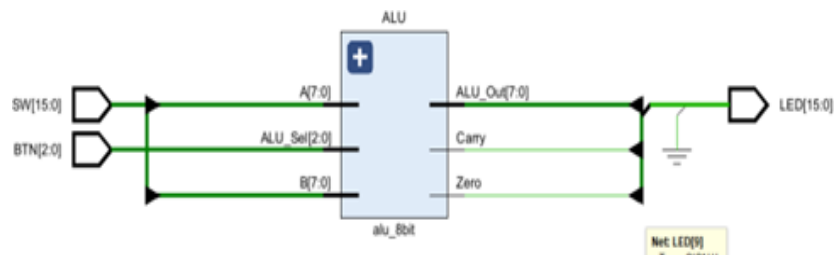


Fig. 4. Top-level connection of alu_8bit to switches (SW[15:0]), buttons (BTN[2:0]) and LEDs (LED[15:0]).

VERIFICATION METHODOLOGY

A. Behavioural Simulation

The testbench `alu_8bit_tb` instantiates the ALU as the device under test and applies one directed vector per operation, holding each for 10 ns, for a total simulation time of 80 ns. The same operands are used throughout so that the results are easy to check by hand: $A = 5$ (00000101) and $B = 3$ (00000011) for the arithmetic and bitwise operations, and A

$= 5$ alone for NOT and the shifts, where B is ignored. With these inputs the expected results are 8 for ADD, 2 for SUB, 00000001 for AND, 00000111 for OR, 00000110 for XOR,

11111010 for NOT, 00001010 for SHL and 00000010 for

SHR. The Carry flag is expected to be 1 only for SUB, because $A \geq B$ means no borrow occurs. The Zero flag is expected to remain 0 for all eight operations, since no result is zero.

The waveform in Fig. 5 shows the final vector of the testbench: with $A = 05h$, $B = 03h$ and $ALU_Sel = 7$, $ALU_Out = 02h$, i.e. $00000101 \gg 1 = 00000010$, with Carry

$= 0$ and Zero $= 0$, matching the expected result.

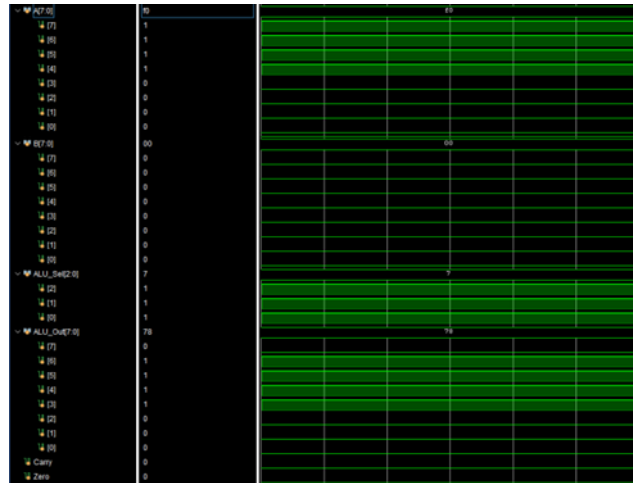


Fig. 5. Behavioural simulation waveform

B. Hardware Verification

After programming the bitstream, each operation is reproduced on the board. Operand A (00000101) is set on SW[15:8] and operand B (00000011) on SW[7:0], the required button combination is held, and the LED pattern is compared with the expected result and flag values. Photographs of the board for each operation are given in Figs. 6–13 (Section VI)

SIMULATION AND FPGA HARDWARE RESULTS

This section reports the behaviour of every operation. For each opcode the switch settings, the button combination, the expected result and flag states are listed, followed by a photograph of the Boolean board showing the measured LED pattern. In all photographs LED[7:0] show the result, LED[8] the Carry flag and LED[9] the Zero flag. The behavioural simulation waveform for the final operation (SHR) is given in Fig. 5, and the expected result for each operation is stated in the subsections below.

A. Opcode 000 (Addition)

With ALU_Sel = 000 (no button pressed), operand A = 00000101 and operand B = 00000011, the expected output is 00001000. The ALU adds the two operands. The sum $5 + 3$

= 8 fits 8 bits, so no carry-out is generated, and Carry stays off; the result is non-zero. The measured LED pattern on the FPGA board is shown in Fig. 6.

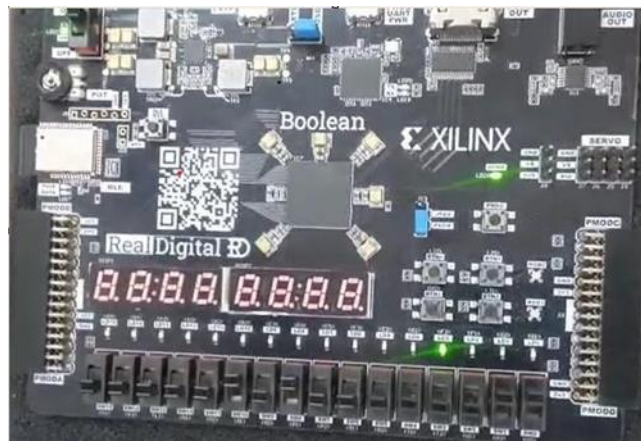


Fig. 6. FPGA output when ALU_Sel is 000 (ADD)

B. Opcode 001 (Subtraction)

With ALU_Sel = 001 (BTN[0] pressed), operand A = 00000101 and operand B = 00000011, the expected output is 00000010. The ALU subtracts B from A: $5 - 3 = 2$. Since A

$\geq$ B, no borrow occurs, so Carry is on (for subtraction it acts as a no-borrow flag). The measured LED pattern on the FPGA board is shown in Fig. 7.



Fig. 7. FPGA output when ALU_Sel is 001 (SUB)

C. Opcode 010 (AND)

With ALU_Sel = 010, operand A = 00000101 and operand B = 00000011, the expected output is 00000001 on LED[7:0]. A bitwise AND is performed; each result bit is 1 only where both operand bits are 1. The measured LED pattern on the FPGA board is shown in Fig. 8.



Fig. 8. FPGA output when ALU_Sel is 010 (AND).

D. Opcode 011 (OR)

With ALU_Sel = 011 (BTN[1] and BTN[0] pressed), operand A = 00000101 and operand B = 00000011, the expected output is 00000111 on LED[7:0]. A bitwise OR is performed; a result bit is 0 only where both operand bits are

0. The measured LED pattern on the FPGA board is shown in Fig. 9.

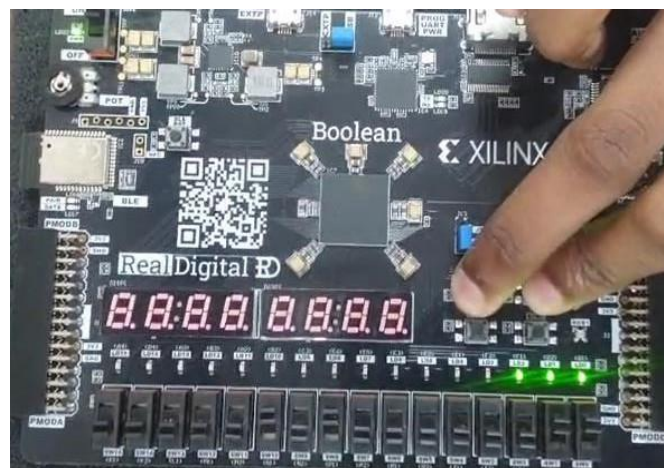


Fig. 9. FPGA output when ALU_Sel is 011 (OR)

E. Opcode 100 (XOR)

With ALU_Sel = 100 (BTN[2] pressed), operand A = 00000101 and operand B = 00000011, the expected output is 00000110 on LED[7:0]. A bitwise exclusive-OR is performed; a result bit is 1 only where the operand bits differ. The measured LED pattern on the FPGA board is shown in Fig. 10.



Fig. 10. FPGA output when ALU_Sel is 100 (XOR)

F. Opcode 101 (NOT)

With ALU_Sel = 101 (BTN[2] and BTN[0] pressed), operand A = 00000101, the expected output is 01011111 on LED[7:0]. This unary operation inverts every bit of A; operand B is ignored. The measured LED pattern on the FPGA board is shown in Fig. 11.



Fig. 11. FPGA output when ALU_Sel is 101 (NOT)

G. Opcode 110 (Shift Left)

With ALU_Sel = 110 (BTN[2] and BTN[1] pressed), operand A = 00000101, the expected output is 00001010 on LED[7:0]. A is shifted left by one position with a 0 shifted into the LSB, which is equivalent to multiplying by two: $0Fh$

$\ll 1 = 1Eh$. Operand B is ignored. The measured LED pattern on the FPGA board is shown in Fig. 12.

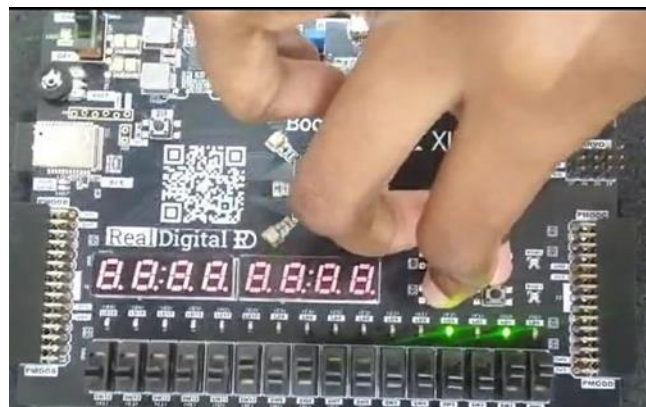


Fig. 12. FPGA output when ALU_Sel is 110

H. Opcode 111 (Shift Right)

With ALU_Sel = 111 (BTN[2], BTN[1] and BTN[0]

pressed), operand A = 00000101, the expected output is 00000010 on LED[7:0]. A is shifted right by one position with a 0 shifted into the MSB. Operand B is ignored. Carry and Zero are both off. The measured LED pattern on the FPGA board is shown in Fig. 13.

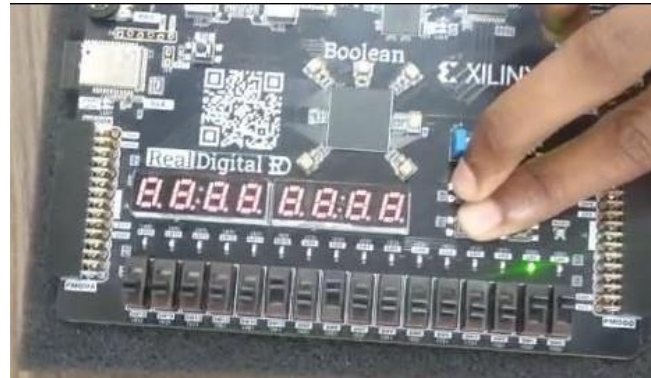


Fig. 13. FPGA output when ALU_Sel is 111 (SHR)

DISCUSSION

A. Functional Agreement

Behavioural simulation and on-board tests agree with the expected values for all eight operations. The Carry flag correctly reports the carry-out of addition and the no-borrow condition of subtraction, and the Zero flag is asserted only when the 8-bit result is zero. On hardware the outputs respond immediately to any change of the switches or buttons, consistent with the purely combinational design.

B. Limitations

Carry is meaningful only for ADD and SUB; shift operations discard the shifted-out bit rather than reporting it. NOT and the shifts ignore operand B, and the multiplication and NAND/NOR functions of [3] are not included. The operation code is entered through push buttons, so it is only held while pressed; a registered version with a clock, debounced inputs and seven-segment output would be more convenient.

CONCLUSION AND FUTURE SCOPE

An 8-bit, 8-operation ALU with Carry and Zero flags was designed in SystemVerilog, verified by simulation, and successfully implemented on a Spartan-7 FPGA using the Boolean board. The board-level top module and XDC constraints allow every operation to be exercised with switches, push buttons and LEDs, and the complete flow from RTL to programmed hardware is documented. Compared with simulation-only ALU studies, the hardware realisation demonstrates the practical considerations of pin assignment, I/O standards and bitstream generation.

Future work includes extending the datapath to 16/32 bits, adding multiplication, rotate and arithmetic-shift operations,

adding signed overflow (V) and negative (N) flags, registering inputs and outputs for synchronous operation, displaying results on the seven-segment display, and reporting measured timing, power and area against alternative adder architectures such as carry-lookahead.

ACKNOWLEDGMENT

The authors thank Vidyalankar Institute of Technology for providing the laboratory facilities and the Boolean development boards used in this work.

REFERENCES

- [1] G. Surekha, G. Madesh, M. P. Kumar, and H. Sriramoju, "Design and implementation of arithmetic and logic unit (ALU)," in Proc. 2nd Int. Conf. Applied Artificial Intelligence and Computing (ICAAIC), May 2023, doi: 10.1109/ICAAIC56838.2023.10140574.
- [2] Z. Huang, "Design and implementation of an 8-bit ALU based on Verilog HDL," in Proc. 3rd Int. Conf. Computing Innovation and Applied Physics, 2024, doi: 10.54254/2753-8818/14/20240939.
- [3] T. S. Hemanth, S. Gagan, and M. N. Lohith Kumar, "Design a 8-bit ALU for arithmetic and logical operations: A Verilog approach," Dept. of Electronics and Instrumentation Engineering, Siddaganga Institute of Technology, Tumakuru, India.
- [4] V. P. Brahmaiah, Y. Sirisinagandla, B. V. Mouktika, and Y. P. Sai, "Design and implementation of optimized based 32-bit arithmetic and logical unit," in Proc. Global Conf. Information Technologies and Communications (GCITC), Dec. 2023, doi: 10.1109/GCITC60406.2023.10425953.

-
- [5] K. Anala, G. Venkata Girish, A. Kamala Kumari, V. Yaswanth, and B. Manoj Kumar, "Design and Implementation of 8 bit Arithmetic Logic Unit on FPGA using VHDL," in Proc. ASAR International Conference on Engineering and Technology (ICET-20), Oct. 2021.
- [6] AMD/Xilinx, Vivado Design Suite User Guide: Design Flows Overview, UG892.
- [7] Real Digital, Boolean Board Reference Manual.