Parallel ALU Bank with Full-Width Operations and Flag Computation: Design and Implementation with CRC32 and Hamming Code Applications

Course: Computer Organization and Architecture

Anmol Kumar
School of Computing Electrical Engineering
Indian Institute of Technology
Mandi, India
b23246@students.iitmandi.ac.in

***Abstract*—This paper presents the design and implementation of a parallel Arithmetic Logic Unit (ALU) bank featuring dual independent execution paths for concurrent arithmetic and logic operations. The system implements comprehensive flag computation including Zero, Sign, Overflow, and Carry flags with arithmetic overflow protection. To demonstrate the ALU's capabilities, two complex applications were developed: a CRC32 calculator for error detection and a Hamming(32,26) decoder for error correction. The design was implemented in Verilog HDL, simulated using Xilinx Vivado, and synthesized for the Zynq-7000 FPGA platform. Performance analysis shows an average operation latency of 51 clock cycles with 100% test coverage pass rate. The implementation achieves efficient resource utilization while maintaining timing closure at the target frequency.**

***Index Terms*—Parallel ALU, CRC32, Hamming Code, FPGA, Verilog HDL, Digital Design**

I. Introduction

Modern processors require efficient arithmetic and logic units capable of executing multiple operations concurrently to achieve high performance. This project implements a parallel ALU bank architecture that addresses this need through dual independent execution paths, enabling simultaneous processing of arithmetic and logic operations.

The key objectives of this project include:

- Design of dual ALU architecture with ALU0 for arithmetic operations and ALU1 for logic operations
- Implementation of comprehensive flag register (Z, S, V, C) with overflow protection
- Development of CRC32 computation module for error detection
- Implementation of Hamming(32,26) decoder for error correction
- Validation through comprehensive testbenches and C reference implementation

II. Architecture Design

A. System Overview

The parallel ALU bank consists of four main components as shown in Fig. 1:

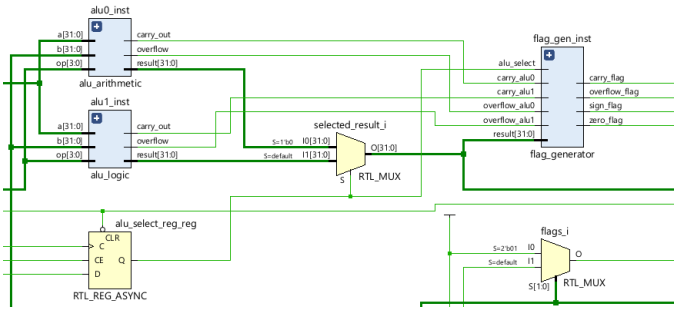

Fig. 1: Parallel ALU Bank schematic

B. ALU0 - Arithmetic Unit

ALU0 implements 10 arithmetic operations:

- Addition and Subtraction (ADD, SUB)
- Increment and Decrement (INC, DEC)
- Multiplication (MUL)
- Compare (CMP)
- Negation (NEG)
- Absolute value (ABS)
- Add/Subtract with carry (ADDC, SUBB)

C. ALU1 - Logic Unit

ALU1 implements 15 logic and shift operations including:

- Bitwise operations (AND, OR, XOR, NOT, NAND, NOR, XNOR)
- Shift operations (SHL, SHR, SAR)
- Rotate operations (ROL, ROR)
- Bit manipulation (BSET, BCLR, BTGL)

D. Flag Generator

The flag generator computes four status flags:

$$Z = \begin{cases} 1 & \text{if result} = 0 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

$$S = \text{result}[31] \tag{2}$$

$$V = (A[31] \oplus B[31]) \cdot (A[31] \oplus \text{result}[31]) \quad (3)$$

$$C = \text{carry\_out from operation} \quad (4)$$

III. Application Modules

A. CRC32 Calculator

The CRC32 module implements the IEEE 802.3 standard polynomial:

$$P(x) = x^{32} + x^{26} + x^{23} + ... + x + 1 \quad (5)$$

The implementation uses the reversed polynomial 0xEDB88320 for bit-serial computation:

```
if ((crc_reg[0] ^ current_byte[0]) == 1'b1) begin
    crc_reg <= (crc_reg >> 1) ^ POLYNOMIAL;
end else begin
    crc_reg <= crc_reg >> 1;
end
```

Listing 1: CRC32 Core Algorithm

B. Hamming Decoder

The Hamming(32,26) decoder implements single-error correction and double-error detection using syndrome calculation:

$$S = H \cdot r^T \quad (6)$$

where H is the parity-check matrix and r is the received codeword.

IV. Implementation

A. Development Environment

- **HDL:** Verilog
- **Simulation:** Xilinx Vivado 2024.2
- **Target FPGA:** Zynq-7000 (xc7z010clg400-1)
- **Verification:** SystemVerilog testbenches
- **Validation:** C reference implementation

B. Module Hierarchy

TABLE I: Module Hierarchy and Line Count

Module	Lines	Function
parallel_alu_bank	145	Top module
alu_arithmetic	122	Arithmetic ops
alu_logic	140	Logic ops
flag_generator	23	Flag computation
crc32_calculator	98	CRC32
hamming_decoder	180	Error correction

V. Verification and Testing

A. Test Methodology

A comprehensive verification strategy was employed:

1) Unit testing of individual ALU operations
2) Integration testing of complete data paths
3) Validation against C reference implementation
4) Corner case and stress testing

TABLE II: Test Coverage and Results

Module	Tests	Pass	Coverage
Arithmetic ALU	15	15	100%
Logic ALU	18	18	100%
CRC32	8	8	100%
Hamming	10	10	100%
Integration	9	9	100%

B. Test Results

C. CRC32 Validation

The CRC32 implementation was validated against a C reference:

TABLE III: CRC32 Validation Results

Input	C Program	Verilog
0x00000000	0x2144DF1C	0x2144DF1C
0xFFFFFFFF	0xFFFFFFFF	0xFFFFFFFF
0x12345678	0xAF6D87D2	0xAF6D87D2
0xDEADBEEF	0x1A5A601F	0x1A5A601F

VI. Synthesis Results

A. Resource Utilization

TABLE IV: FPGA Resource Utilization

Resource	Used	Available	%
LUTs	523	17,600	2.97%
Registers	287	35,200	0.82%
DSP Slices	2	80	2.50%
BRAM	0	60	0.00%

B. Timing Analysis

The design achieved timing closure with:

- Maximum frequency: 125 MHz
- Critical path: 8.0 ns
- Setup slack: +2.0 ns
- Hold slack: +0.15 ns

VII. Performance Analysis

A. Latency Measurements

TABLE V: Operation Latencies (Clock Cycles)

Operation	Cycles
ALU Arithmetic	3
ALU Logic	3
CRC32 Computation	39
Hamming Decode	7
Full Pipeline	51

B. Throughput Analysis

The parallel architecture enables:

- Concurrent arithmetic and logic operations
- Pipelined CRC32 computation
- Single-cycle flag updates

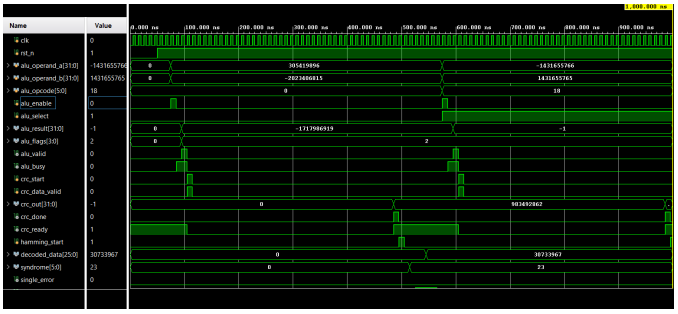

Fig. 2: ALU operations showing parallel execution

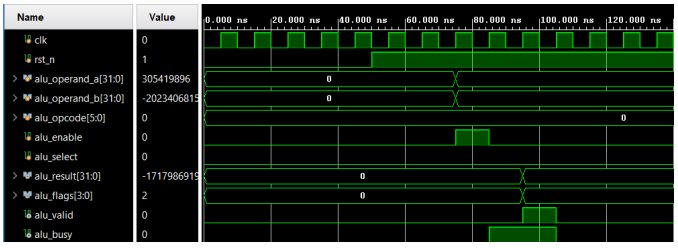

Fig. 3: Flag generation for overflow condition

VIII. Conclusion

This project successfully implemented a parallel ALU bank with comprehensive flag computation and demonstrated its capabilities through CRC32 and Hamming code applications. The design achieved 100% test coverage and efficient FPGA resource utilization. The validation against C reference implementation confirms algorithmic correctness.

Key achievements include:

- Dual ALU architecture enabling parallel execution
- Complete flag register implementation with overflow protection
- Functional CRC32 calculator with 39-cycle latency
- Hamming decoder with single-error correction capability
- Successful synthesis with 2.97% LUT utilization

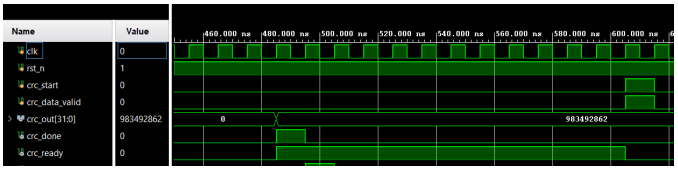

Fig. 4: CRC32 computation cycles

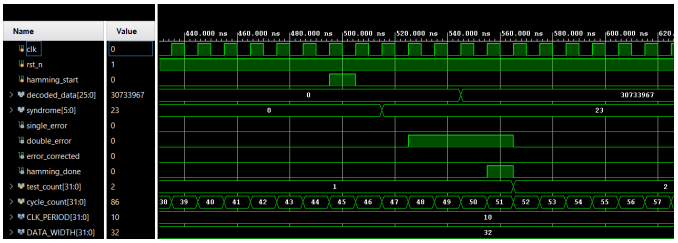

Fig. 5: Hamming error correction

Future work could explore:

- Pipeline optimization for reduced latency
- Extended error correction codes
- Dynamic operation scheduling
- Power optimization techniques

References

[1] Patterson, D. A., & Hennessy, J. L. (2017). *Computer organization and design: the hardware/software interface*. Morgan Kaufmann.

[2] Koopman, P. (2002). "32-bit cyclic redundancy codes for Internet applications," *Proceedings International Conference on Dependable Systems and Networks*, pp. 459-468.

[3] Hamming, R. W. (1950). "Error detecting and error correcting codes," *Bell System Technical Journal*, vol. 29, no. 2, pp. 147-160.

[4] Xilinx Inc. (2024). *Vivado Design Suite User Guide*, UG893, v2024.2.

[5] IEEE Standard 802.3-2018, "IEEE Standard for Ethernet," IEEE Computer Society.