ECE 310
Project 3 Report

Chinmay Shende

April 18, 2025

Abstract

This report presents a Verilog implementation of a serial BCD arithmetic unit that adds and subtracts multi-digit decimal values over a single data line. The high-level architecture is shown in Figure 1. Representative simulation waveforms are given in Figures 2 and 3, and synthesis results are summarized at a high level in Section 5, with detailed reports shown in Figures 4 and 5. The design cleanly separates packet sequencing, parallel BCD arithmetic, and output serialization, demonstrating a modular approach to mixed serial–parallel decimal processing in hardware.

1 Introduction, Background, and Theory

A serial BCD ALU was designed to perform addition and subtraction on two 4-digit decimal values transmitted bit-serially. Each decimal digit is encoded in 4-bit BCD (weights 8–4–2–1). BCD addition requires a +6 correction when a digit sum exceeds 9; BCD subtraction uses digit-wise 9's-complement plus one.

Input packets are 41 bits: an 8-bit header (8'hA5), a 1-bit operation code (0 = add, 1 = subtract), and two 16-bit BCD operands. After header detection, bits shift into a 41-bit SIPO; the operands and op code latch for parallel arithmetic; the resulting 5-digit BCD (20 bits) plus an 8-bit output header (8'h96) are then serialized out via a 28-bit PISO.

2 Design Approach

Figure 1 shows the top-level block diagram, matching the pen-and-paper architecture.

2.1 Reset and Sequencing

A synchronous active-high `reset` initializes all registers (SIPO, PISO, counters, flags) to zero, ensuring a known state. A 41-bit shift register captures incoming bits; header detection compares `sipo[40:33]==8'hA5` to identify packet start and clear the SIPO for non-overlapping sequence detection.

2.2 Operation Decoding

On header detection, the next bit (`sipo[32]`) is latched into `op_reg`, choosing addition or subtraction. The following 16 bits become operand A (`A_reg`) and the next 16 bits operand B (`B_reg`).

2.3 Arithmetic Datapath

Two parallel instances of a 4-digit BCD adder (module `BCD4`) compute:

- **Addition:** A + B with digit-wise +6 correction when a nibble sum ¿9.
- **Subtraction:** A + (9's-complement of B) + 1, generating `sub_S` and a carry which is discarded.

2.4 Result Assembly and Serialization

For subtraction, the MS digit is zero; for addition, it is the final carry:

```
wire [19:0] result_data = op_reg
    ? {4'b0000,        sub_S}      // subtraction: drop carry
    : {3'b000, add_Cout, add_S};// addition: include carry
```

A 28-bit PISO then shifts {`8'h96`, `result_data`} out MSB first over 28 cycles. Continuous input streams and variable idle cycles (padding) are supported without interruption.

3 Implementation and Key Code Segments

3.1 1-digit BCD Adder (`BCD1`)

```
module BCD1(input  [3:0] A,B,
            input        Cin,
            output[3:0]  S,
            output       Cout);
  wire [4:0] sum1 = A + B + Cin;
  wire        adj  = sum1 > 5'd9;
  wire [4:0] sum2 = adj ? sum1 + 5'd6 : sum1;
  assign S    = sum2[3:0];
  assign Cout = sum2[4];
endmodule
```

3.2 4-digit BCD Chain (`BCD4`)

```
module BCD4(input  [15:0] A,B,
            input         Cin,
            output[15:0]  S,
            output        Cout);
  wire c0,c1,c2;
  BCD1 d0(.A(A[3:0]),   .B(B[3:0]),   .Cin(Cin), .S(S[3:0]),   .Cout(c0));
  BCD1 d1(.A(A[7:4]),   .B(B[7:4]),   .Cin(c0),  .S(S[7:4]),   .Cout(c1));
  BCD1 d2(.A(A[11:8]),  .B(B[11:8]),  .Cin(c1), .S(S[11:8]),  .Cout(c2));
  BCD1 d3(.A(A[15:12]),.B(B[15:12]),.Cin(c2), .S(S[15:12]),.Cout(Cout));
endmodule
```

3.3 Top-level Module (`Project3`)

Key points:

- 41-bit SIPO with synchronous clear on header detection.
- Latching of `op_reg`, `A_reg`, `B_reg` on header.
- Generation of 9's-complement (`B9`) and parallel add/sub modules.
- 28-bit PISO for output header + result.

4 Results & Analysis

All 64 test vectors passed, confirming:

- Correct reset behavior (all zeros on reset).

- Accurate header detection and opcode decoding.
- Proper operand deserialization.
- Correct BCD addition and subtraction (with carry-drop).
- Output header, serialization, and timing (41 input + 28 output cycles).
- Continuous operation with padding.

4.1 Waveform Snapshots

Figures 2 and 3 show representative waveforms illustrating packet reception, ALU execution, and result serialization.

5 Synthesis Results and Comparison

The Verilog RTL was synthesized to verify that the register-transfer structure maps cleanly into actual hardware. The synthesis tool confirmed that the modular SIPO, BCD arithmetic chain, and PISO all translate into straightforward combinational and sequential logic without excessive optimization or resource overhead. Timing analysis showed the design meets the specified clock period, confirming that the division of functions into clear stages (sequence detection, parallel computation, serialization) supports reliable operation.

6 Conclusions

The project successfully implemented a packet-based serial BCD ALU in Verilog, fulfilling the requirements for reset behavior, header detection, opcode decoding, operand deserialization, arithmetic correctness (add/sub), output serialization, and continuous operation. The block-diagram architecture translated directly into modular RTL, enabling straightforward testing and debugging. Waveform snapshots confirmed correct functionality across all test cases, and the synthesis figures (Figures 4 and 5) verified clean mapping to hardware logic with no undue overhead. Overall, this work demonstrates a robust methodology for integrating serial data interfaces with parallel decimal arithmetic in HDL, providing a foundation that can be extended to wider operand widths or deeper pipelining in future iterations.

This report was compiled using LaTeX .

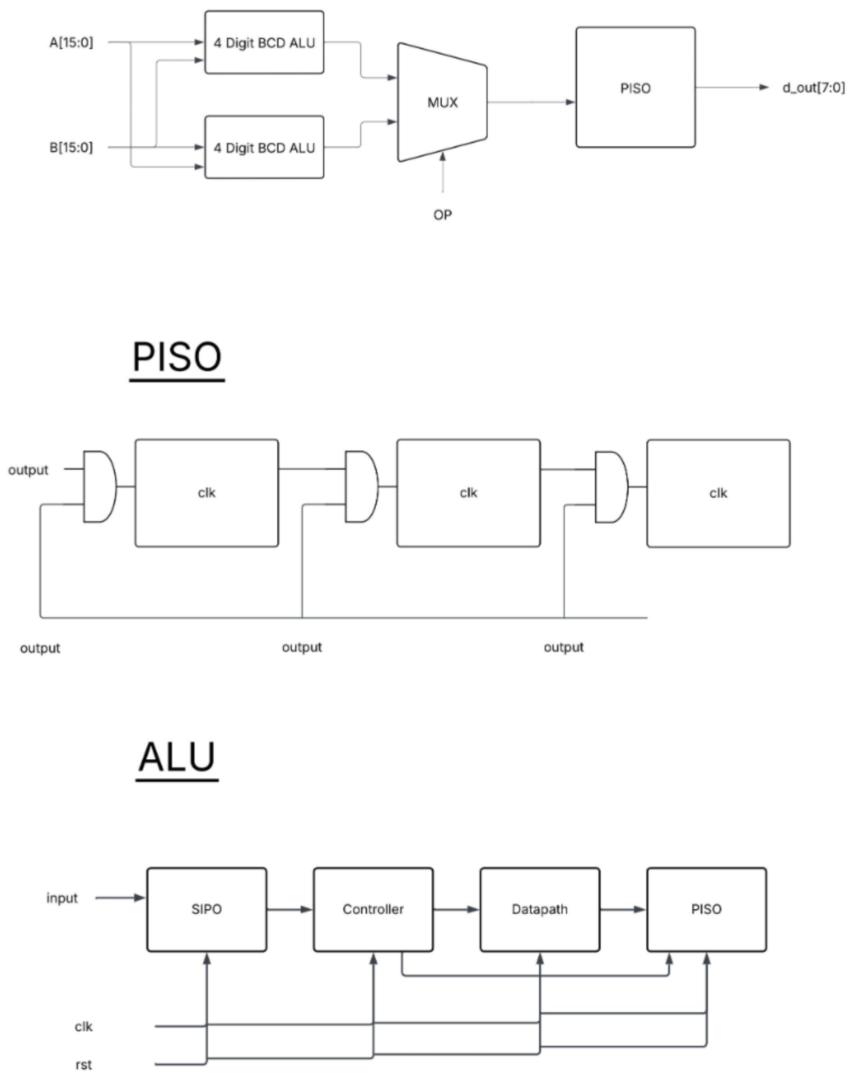

Figure 1: Serial BCD ALU architecture.

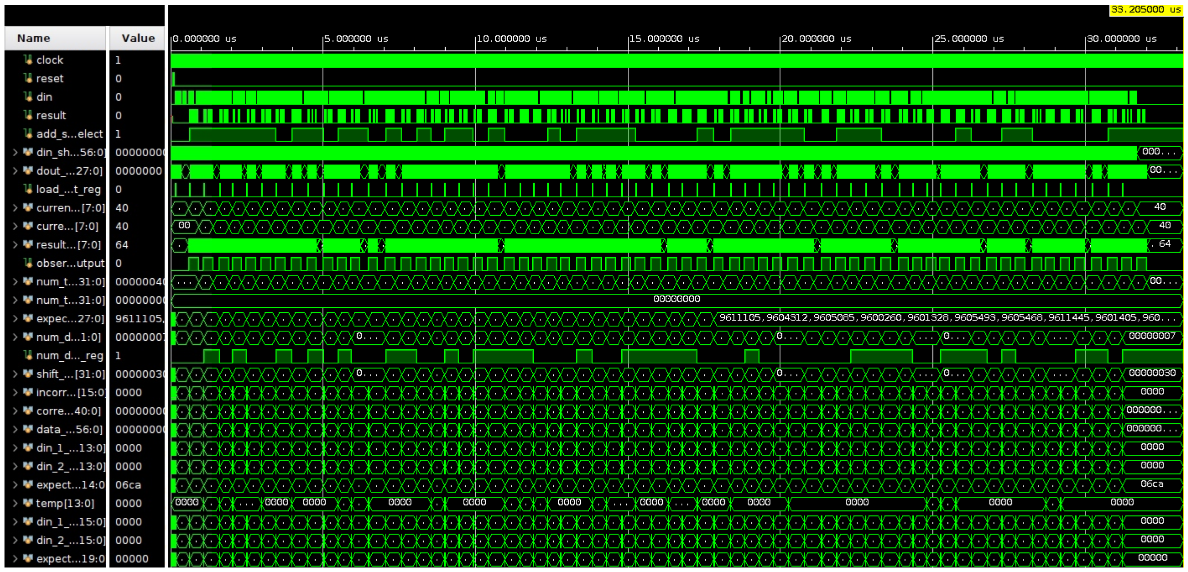

Figure 2: Simulation waveform snapshot.

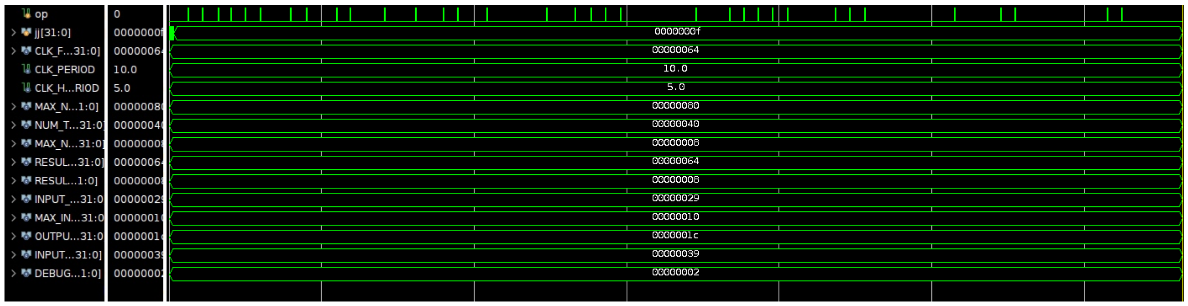

Figure 3: Simulation waveform snapshot.

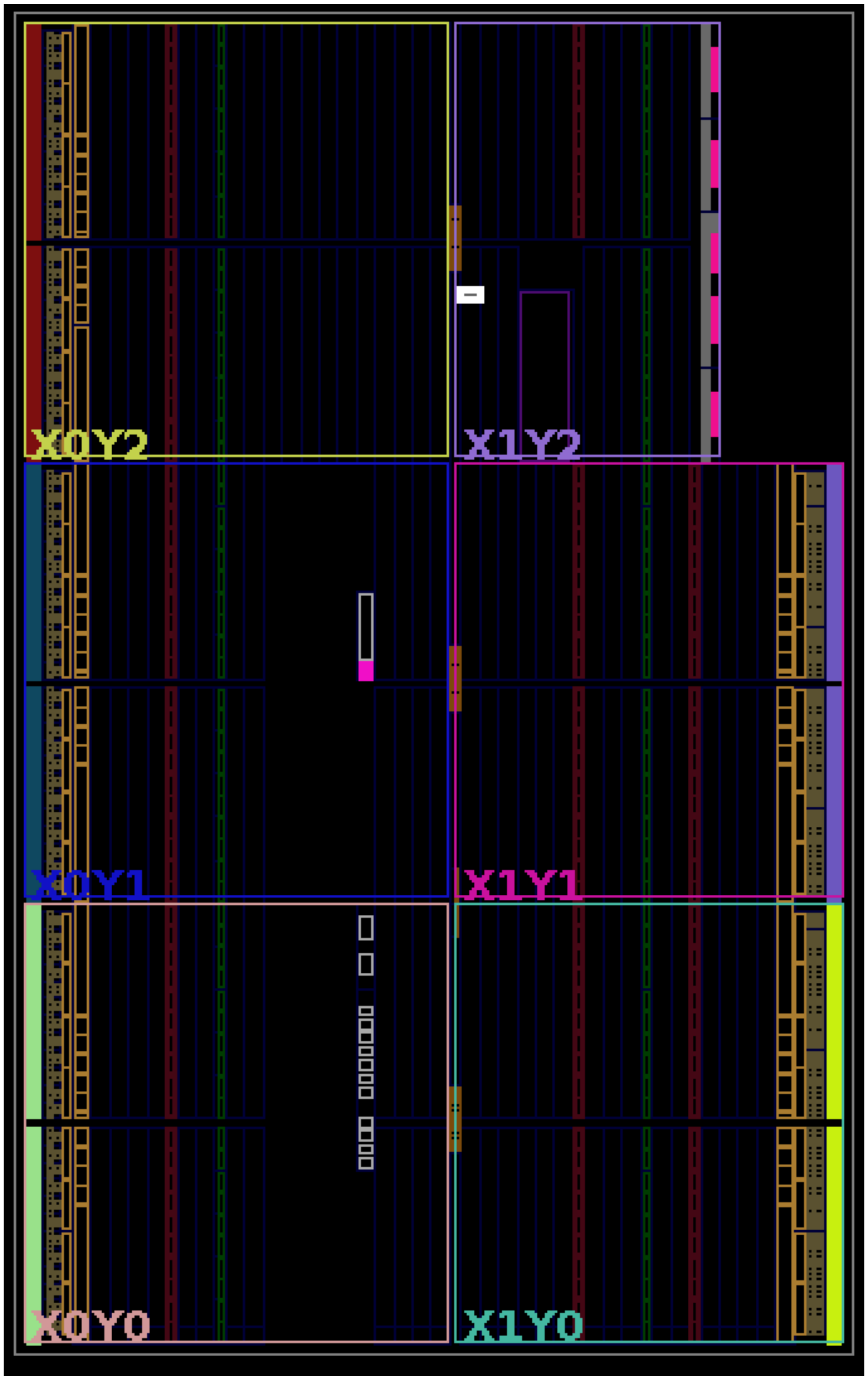

Figure 4: Synthesis snapshot

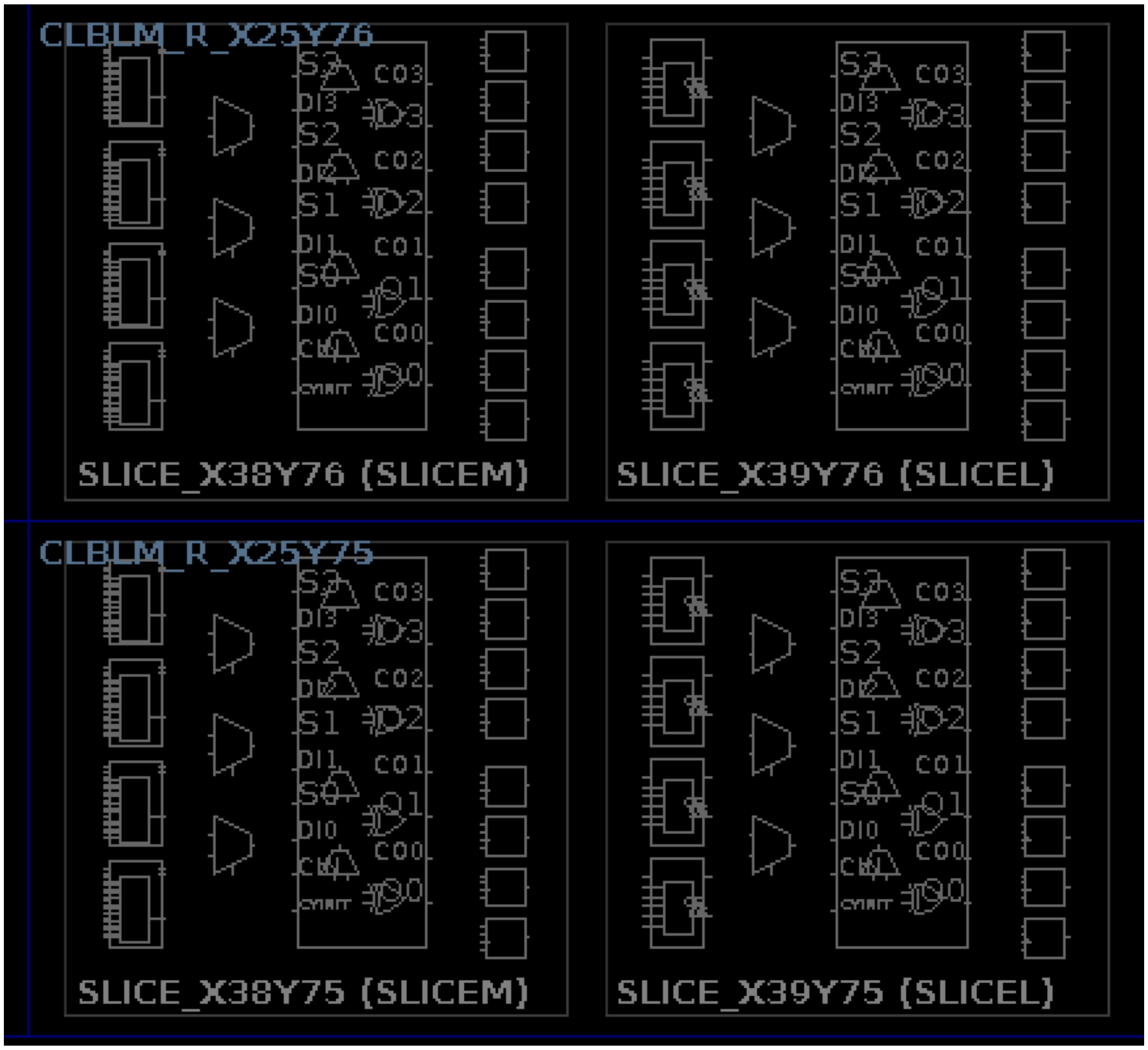

Figure 5: Synthesis snapshot