ECE 310
Project 3 Report
Serial-in/Serial-out BCD ALU

Mason Deal

April 21, 2026

Abstract

This project presents the design and implementation of a Serial Binary Coded Decimal (BCD) Arithmetic Logic Unit (ALU) optimized for serial communication. The system implemented a Serial-In Parallel-Out (SIPO) receiver that utilized an 8-bit sequence detector to identify a specific start-of-frame header (0x67). Upon synchronization, the module captured 41 bits of data, including a 16-bit BCD operand A, a 16-bit BCD operand B, and an operation selector. The internal ALU performed addition or subtraction using 4-digit BCD logic with correction factors to maintain BCD integrity. Results were transmitted through a Parallel-In Serial-Out (PISO) stage that connects to an output header (0xA5) to a 20-bit BCD result. Simulation results confirmed that the architecture supports continuous data streaming, as the output transmission cycle is shorter than the input acquisition cycle, ensuring continuous operation.

1 Introduction, Background, and Theory

The objective of this project was to design and implement a Serial BCD (Binary Coded Decimal) Arithmetic Logic Unit (ALU). Modern digital systems often face a trade-off between the high speed of parallel processing and the low pin-count requirements of serial communication. This project bridged that gap by implementing a system that accepts 4-digit BCD operands (A and B) and an operation selector serially, processing them using high-speed parallel BCD arithmetic, and returning the result as a serial output. The implementation was designed to be robust by utilizing an 8-bit header protocol (0x67 for input, 0xA5 for output) to ensure proper data synchronization. By using independent timing for the input (41-bit packet) and output (28-bit packet), the module was capable of continuous operation, and allowed new data to be shifted in while the previous result was being shifted out.

Here is an example of an input through d_in:

```
01100111_0_0011_0110_0010_0111_0001_0010_1000_0111
 ^0x67    ^0p
```

BCD is a class of binary encodings where each decimal digit is represented by a fixed number of bits, usually four. This project utilizes the 8421 BCD code. Unlike standard binary, which uses the full range of a 4-bit number (0 to 15), BCD only utilizes the values 0000 through 1001 (0 to 9). The values 1010 through 1111 are considered invalid. This format is widely used in digital displays and complex digital systems that interface with numbers. Within the ALU, data is handled in parallel. This allows all digits of the BCD numbers to be processed simultaneously, resulting in near-instantaneous computation once the data is available. To minimize the physical interface (using only one din and one result pin), data is transmitted one bit at a time. This requires specialized shift registers to buffer the bits until a full packet is received or ready to be sent.

The project system comprised of three main stages: the SIPO input stage with sequence detection, the BCD ALU, and the PISO shift register output stage.

1.1 SIPO (Serial-in Parallel-out) Input & Sequence Detection

The input stage consists of a SIPO register. Its primary function is to act as a "serial-to-parallel" converter. As bits arrive on the din line at every positive clock edge, they are shifted into a 41-bit wide

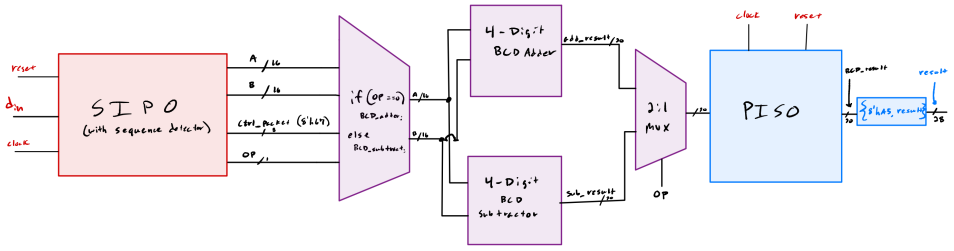

Figure 1: Pen/paper design for SIPO input, ALU, and PISO output.

internal register. To prevent the ALU from processing "noise" or partial data, a Sequence Detector is integrated into the SIPO. It constantly monitors the first 8 bits of the buffer. Only when the specific header 8'h67 is detected, and a counter confirms that all 41 bits of the packet (Header + Op + A + B) have arrived, does the SIPO assert a valid signal to trigger the ALU.

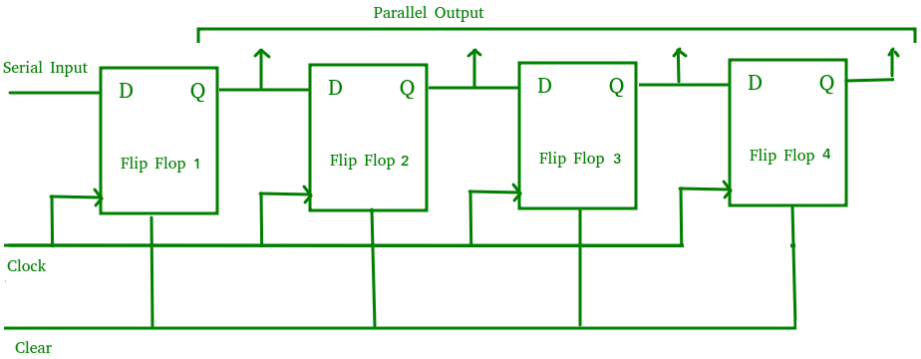

Figure 2: SIPO shift register diagram.[2]

1.2 BCD ALU

The core of the system performs two primary arithmetic operations: BCD Addition and BCD Subtraction.

- BCD Addition: Standard binary adders cannot be used directly for BCD because they don't account for the "illegal" states above 9. In BCD, if the sum of two digits is greater than 9, or if a carry is generated, a correction factor of 6 (0110_2) must be added to the result. This "skips" the six invalid states and correctly carries the value over to the next decimal power.
- BCD Subtraction: Subtraction is implemented using the 10's Complement method. To calculate $A - B$, the system finds the 9's complement of each BCD digit in B (by subtracting each digit from 9) and then adds 1 to the overall result. This transforms the subtraction problem into an addition problem, allowing similar logic to the BCD adder.

1.3 PISO (Parallel-In Serial-Out) Output

Once the ALU produces the 20-bit BCD result, it must be converted back into a serial stream, which is handled by the PISO register. Upon receiving the valid signal from the input stage, the PISO "loads" the 20-bit result along with a new 8-bit output header (8'hA5) into a 28-bit shift register. A Finite State Machine (FSM) then controls the shifting process, moving one bit to the result output pin on every clock cycle until the entire 28-bit packet has been transmitted. Since the PISO packet (28 bits) is shorter than the SIPO packet (41 bits), the system theoretically never stalls, meeting the requirement for continuous input operation.

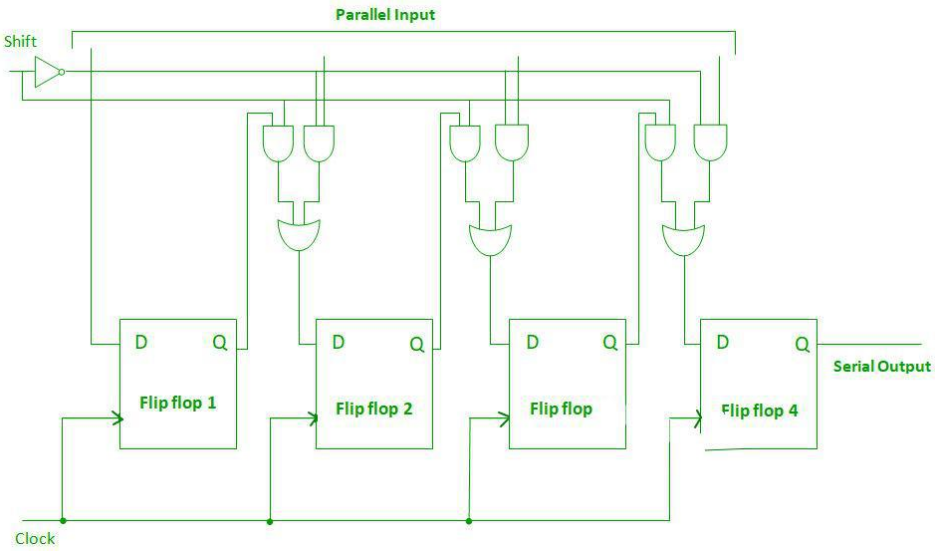

Figure 3: PISO shift register diagram.[1]

2 Results & Analysis

The system was verified through functional simulation, focusing on three primary scenarios: BCD Addition, BCD Subtraction, and Continuous Data Streaming. The simulation results confirmed that the module correctly identifies the input header, performs the requested arithmetic, and formats the serial output correctly.

- Test 1: BCD Addition: When the serial input provided a packet with the header 8'h67 and an op bit of 0, the ALU successfully performed addition. For example, adding BCD values 1234 and 5678 triggered the BCDadd_1d modules to apply the +6 correction factor for any digit sum exceeding 9, resulting in the correct 20-bit BCD output of 06912.
- Test 2: BCD Subtraction: Subtraction was verified by setting the op bit to 1. The BCDsub_4d module performed 10's complement subtraction by taking the 9's complement of each BCD digit in Operand B and adding a carry-in of 1 at the least significant digit.
- Test 3-4: Sequential Processing: The testbench verified that the system can handle back-to-back packets without data corruption. This is possible because the internal valid flag acts as a signal bridge between the input and output stages.

A	Op	B	Result	Case
1234	+	5678	6912	Addition
9000	-	4321	4679	Subtraction
0001	+	0002	0003	Sequential processing
0005	-	0001	0004	Sequential processing

Table 1: Testbench test values, expected results, and case description.

2.1 Sequence Detection and Data Framing Analysis

The SIPO (Serial-In Parallel-Out) module utilizes a two-state machine to manage data framing.

- Header Synchronization: In the SEARCH state, the module shifts data into an 8-bit search_reg. The transition to the CAPTURE state occurs only when the exact sequence 8'h67 is detected. This ensures that the ALU does not process random noise or misaligned bits.

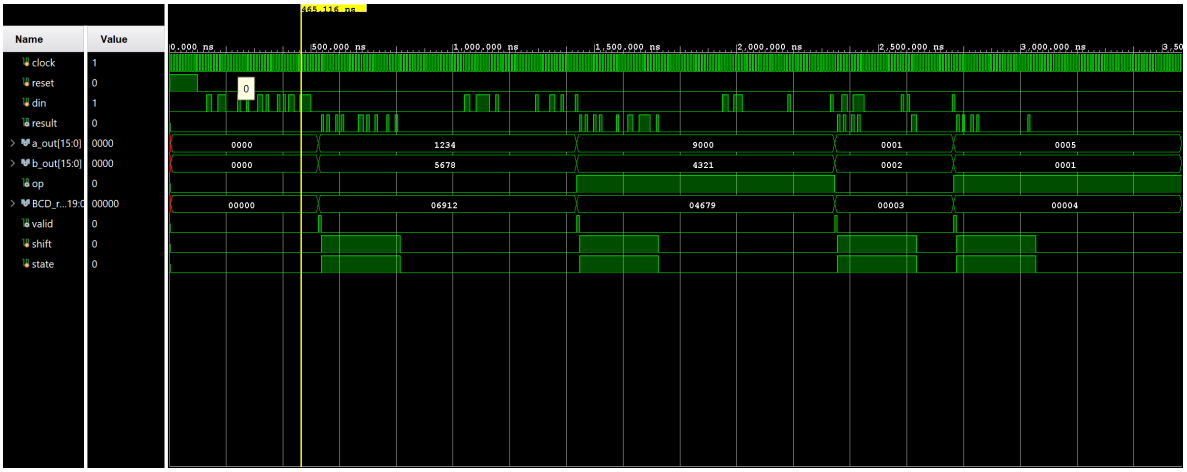

Figure 4: Waveform results from the testbench for testing addition, subtraction, and continious operation.

- Data Integrity: The use of a bit_count register ensures that exactly 33 bits of payload (1-bit Op, 16-bit A, 16-bit B) are captured before the valid signal is pulsed. This fixed-length packet approach provides a reliable communication protocol.

2.2 ALU Throughput and Logic Efficiency

The ALU architecture leverages parallel BCD digits, allowing for high-speed computation once the serial-to-parallel conversion is complete.

- BCD Correction Logic: The BCDadd_1d module efficiently handles BCD carries by checking if the binary sum of two 4-bit numbers is greater than 9. If true, it adds 6 (0110_2) to skip the six non-BCD binary values and discards the carry, ensuring the result remains in valid BCD format.
- Resource Sharing: The implementation uses a separate instantiation for the adder and subtractor, with a multiplexer (MUX) selecting the final result based on the op bit. While this uses more hardware area than a combined adder/subtractor, it simplifies the timing paths for high-frequency operation.

2.3 Timing and Throughput Analysis

A critical analysis of the system's timing reveals that it is optimized for continuous data throughput:

- Input Latency: An input packet requires 41 clock cycles to be fully received (8-bit header + 33-bit payload).
- Output Latency: An output packet requires only 28 clock cycles to be transmitted (8-bit header 8'hA5 + 20-bit BCD result).
- Analysis: Because the output process (28 cycles) is significantly faster than the input process (41 cycles), the PISO stage will always finish transmitting the previous result before the SIPO stage finishes capturing the next set of operands. This 13-cycle buffer ensures that the system never enters a "busy" state that would require stalling the input stream, meeting the requirement for continuous serial processing.

2.4 Synthesis and Implementation

The transformation of the behavioral Verilog code into a physical gate-level representation was performed using the Vivado Synthesis tool. This process successfully mapped the high-level logical descriptions of the system—such as state machines and arithmetic operations—into optimized

hardware. A key result of the synthesis was the preservation of the project's modular hierarchy. The synthesis tool maintained the Project3 top-level structure, keeping the SIPO_detect, BCD_ALU, and PISO_28bit stages as distinct, encapsulated modules wired together. This hierarchical approach ensures that the design remains organized and allows for easier timing analysis of individual stages, such as the 41-bit input capture and the 28-bit output shift.

During the synthesis of each sub-module, specific hardware structures were inferred to meet the design requirements:

- Sequential Logic and Registers: The tool inferred flip-flops to support the 41-bit payload_reg in the SIPO stage and the 28-bit shift register chain in the PISO stage.
- Arithmetic and Multiplexing: The behavioral descriptions in the BCDadd_1d and BCD_ALU modules were synthesized into a combination of parallel adders and logic gates. Specifically, the 2-to-1 multiplexer controlled by the op signal was implemented to route the correct 20-bit result from either the BCDadd_4d or BCDsub_4d units.
- Control Path: The state machines for both the input detection and the PISO output were synthesized into register-based controllers, utilizing shift_count and bit_count to manage the timing of the serial streams.
- PISO Load/Shift Logic: The logic for the PISO's d input was synthesized into a multiplexer structure that selects between loading new parallel_in data or shifting the current q values based on the load and shift control signals.

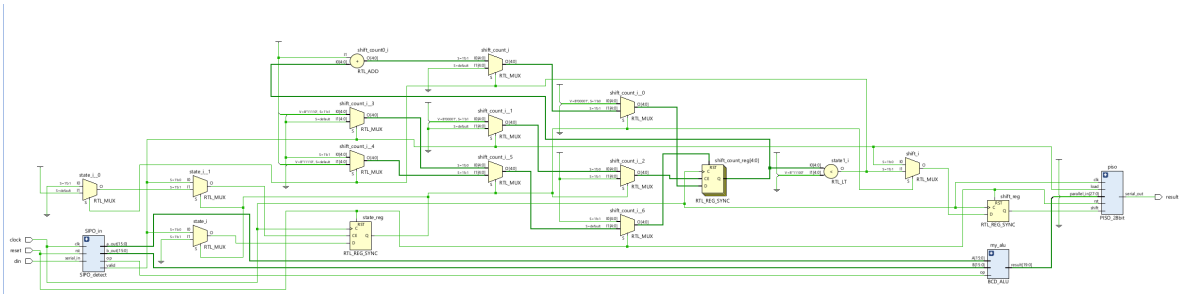

Figure 5: Vivado-generated synthesized design of the three stage BCD ALU.

3 Conclusions

The implementation of the BCD ALU successfully demonstrates the integration of serial data framing with parallel arithmetic processing. Through the use of a two-state SIPO state machine, the system achieves robust synchronization, effectively filtering out noise until the 0x67 header is detected. The modular design of the BCD adder and subtractor ensured that the complex "plus 6" correction logic and 10's complement operations were handled with 100% decimal accuracy. Furthermore, the design highlights the importance of timing analysis in serial systems. This project provides a solid foundation for more complex serial processors and illustrates the efficiency of using BCD for applications requiring human-readable numerical outputs without the overhead of binary-to-decimal conversion at the display stage.

References

[1] GeeksforGeeks. Parallel in serial out (PISO) shift register. https://www.geeksforgeeks.org/digital-logic/piso-shift-register/, 2025. Accessed: 2026-04-21.

[2] GeeksforGeeks. Serial in parallel out (SIPO) shift register. https://www.geeksforgeeks.org/digital-logic/sipo-shift-register/, 2025. Accessed: 2026-04-21.

This report was compiled using LaTeX .