ECE 310
Project 3 Report

Brody Enget

April 14, 2025

Abstract

This project implements a Serial BCD Arithmetic Logic Unit (ALU) capable of adding and subtracting two 4-digit BCD numbers provided as a serial input stream. The system utilizes SIPO and PISO registers to convert between serial and parallel data, enabling accurate BCD operations and serial result output in a structured packet format.

1 Introduction

The Serial BCD ALU system is designed to perform arithmetic operations—specifically addition and subtraction—on two 4-digit Binary Coded Decimal (BCD) numbers provided as a continuous serial input stream. The top-level module, `Project3`, interfaces with the outside environment through four key ports: a free-running clock (`clock`), a synchronous active-high reset signal (`reset`), a 1-bit serial input (`din`), and a 1-bit serial output (`result`). The system is designed to detect and decode input "packets" that contain operation and operand information, perform the corresponding arithmetic operation in parallel, and then serialize the output for transmission.

The input `din` is used to receive a 41-bit operation packet consisting of an 8-bit control header, a 1-bit operation specifier (0 for addition, 1 for subtraction), and two 16-bit BCD operands (each representing a 4-digit number). This stream is continuously shifted into a Serial-In Parallel-Out (SIPO) register. When the control pattern is detected, the system captures the operands and operation type, clears the register, and begins processing. Internally, the ALU operates on the parallel data to compute the 5-digit BCD result (20 bits). The final result is then serialized and output through the `result` pin, packaged with an 8-bit control header to mark the start of the response packet.

Signal	Direction	Description
clock	input	Free-running system clock
reset	input	Active-high synchronous reset
din	input	1-bit serial input data stream
result	output	1-bit serial output data stream

Table 1: Top-Level Ports for `Project3` Module

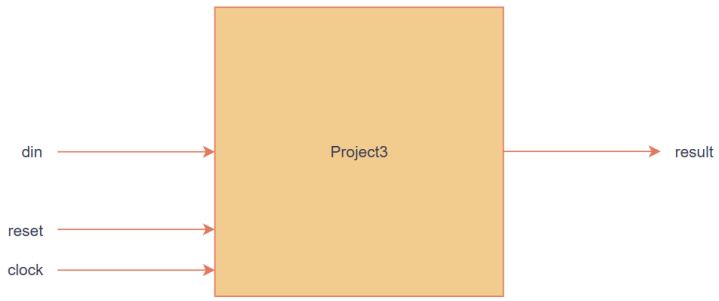

Figure 1: Top-level Block Diagram

2 Design & Theory

This section will discuss the design and theory of the project. The project will be implemented with 4 primary modules: 41-bit SIPO register, 28-bit PISO register, an ALU, and a controller. This modules will be instantiated via a top level module called Project3.

2.1 BCD ALU

The ALU module accepts a total of 9 inputs: a 1-bit operation selector (`op`) and eight 4-bit BCD values representing two 4-digit operands (`A0-A3` and `B0-B3`). The operation is determined by the value of `op`: when `op = 0`, the ALU performs BCD addition; when `op = 1`, it performs BCD subtraction. For subtraction operations, it is assumed that operand A is greater than operand B, ensuring that the result is always non-negative. The ALU outputs five 4-bit BCD digits (`F0-F4`) representing the result of the computation.

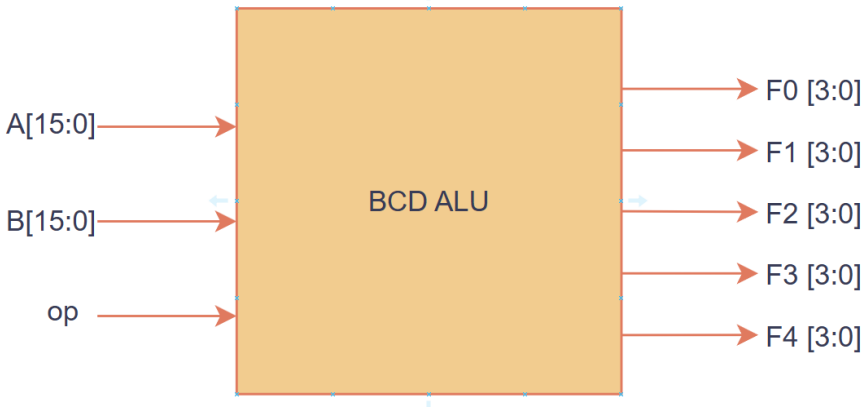

Figure 2: ALU Top-Level Block Diagram

Signal	Direction	Description
op	input	1-bit operation selector
A0, A1, A2, A3	input	4-bit BCD digits of operand A
B0, B1, B2, B3	input	4-bit BCD digits of operand B
F0, F1, F2, F3, F4	output	4-bit BCD digits of ALU result

Table 2: Top-Level Ports for `ALU` Module

Each BCD digit must be processed to ensure the result remains a valid BCD representation, particularly when the digit sum exceeds 9. The arithmetic operation is determined by the value of the `op` signal:

- `op = 0`:
  - The ALU performs standard BCD addition.
  - Inputs `A` and `B` are passed directly to the BCD adder module.
  - Correction logic is applied to adjust for digit sums greater than 9 by adding `4'd6`.
- `op = 1`:
  - The ALU performs BCD subtraction.
  - Inputs `B` are converted to their 10's complement before being added to `A`.
  - The 10's complement is formed by inverting each bit of the BCD digit (bitwise NOT) and adding `4'd10`.
  - The initial carry-in to the BCD adder is set to 1 (the value of `op`), enabling proper subtraction.

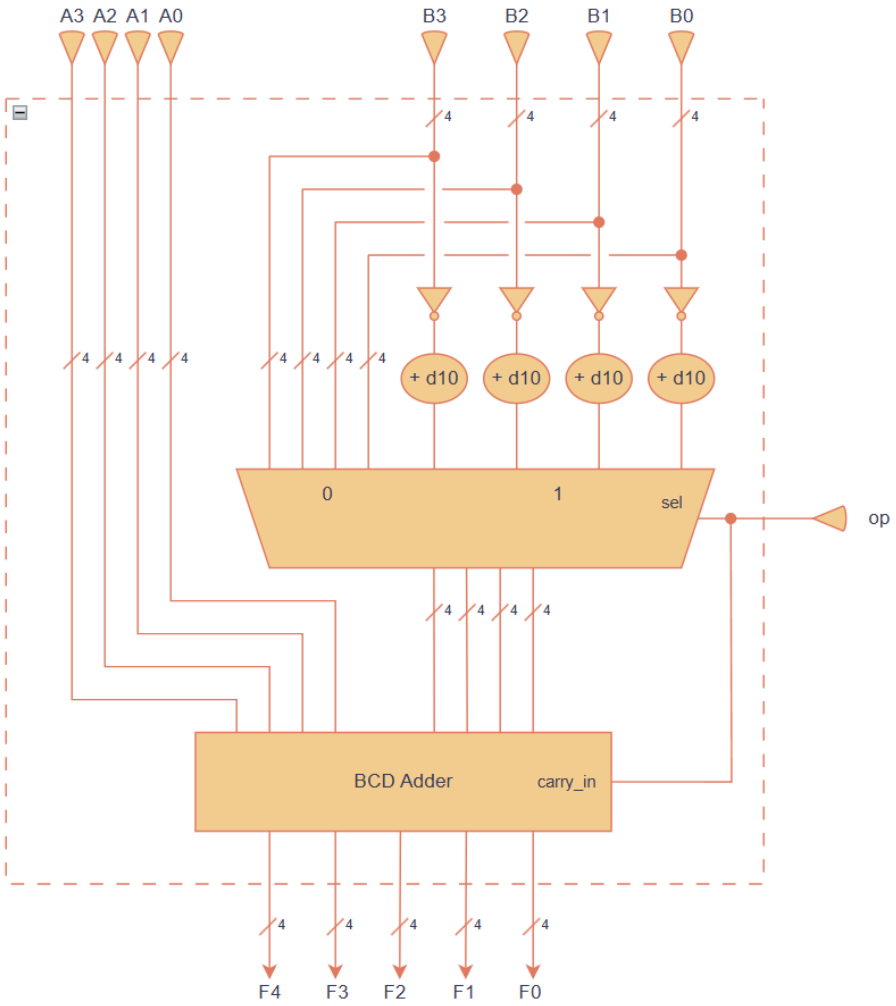

Figure 3: BCD ALU

The BCD adder consists of four single-digit BCD full-adders, each responsible for adding one pair of corresponding BCD digits from the A and B inputs. Each full-adder is based on a standard 4-bit binary adder that computes the sum of the two digits along with an incoming carry. If the sum exceeds 9 (i.e., is not a valid BCD digit), the adder applies a correction by adding `4'd6` to the result. This ensures the output remains a valid BCD value and generates a carry to the next higher digit when necessary.

This correction relies on the 5^{th} bit (carry-out) of the `+6` operation. When the sum of two BCD digits exceeds 9, adding 6 will cause the 5^{th} bit to be high. If the sum is 9 or less, the carry-out will remain low. This behavior allows the 5^{th} bit to serve two purposes: it indicates whether correction is needed (i.e., whether to select between the uncorrected and corrected sum) and also functions as the carry-out (`cout`) to the next digit.

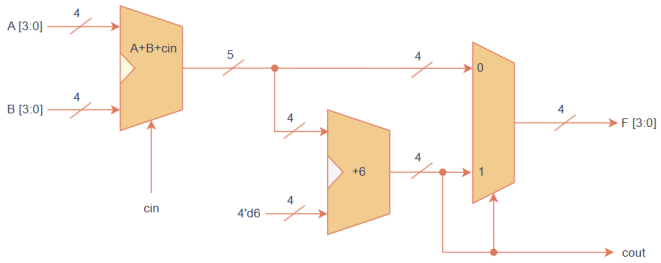

Figure 4: 1-digit BCD Full-Adder

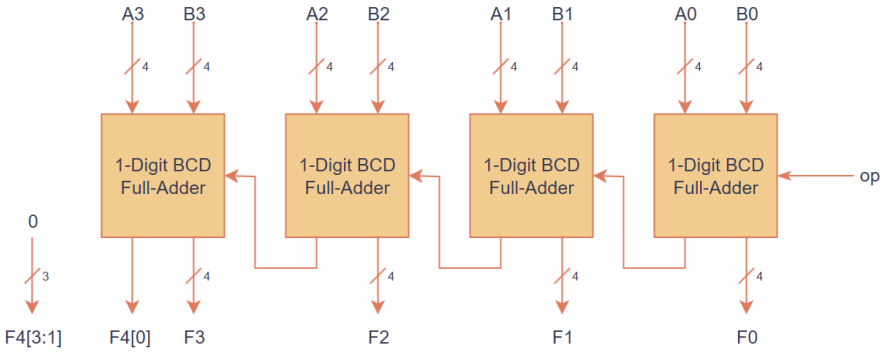

Figure 5: 4-digit BCD Adder w/ Carry-out

An important observation about the most significant BCD digit (`F4`) is that it can only take on the values `4'b0000` or `4'b0001`, as it solely represents the final carry-out from the addition or subtraction process. Because of this, there is no need to implement a full single-digit BCD adder for `F4`. Instead, the design uses only four single-digit BCD adders to compute `F0-F3`, and `F4` is constructed by concatenating three leading zeros with the final carry-out bit from the last BCD adder stage.

2.2 SIPO Register

The SIPO (Serial-In Parallel-Out) register is designed to collect serial input bits from `din` and convert them into a 41-bit parallel format. This is achieved by performing a left shift operation on every clock cycle, effectively shifting the bits from `SIPO[0]` to `SIPO[40]`. It should be noted that this implementation does not include a dedicated shift enable control; instead, it is assumed that the register continuously shifts with each clock cycle.

For clarity, once the correct input pattern is detected and the PISO register is loaded with the processed ALU outputs, the contents of the SIPO register are interpreted as follows:

- `SIPO[40:33]` – Pattern sequence
- `SIPO[32]` – Operation selector (`op`)
- `SIPO[31:16]` – 4-digit BCD input A (`A0-A3`)
- `SIPO[15:0]` – 4-digit BCD input B (`B0-B3`)

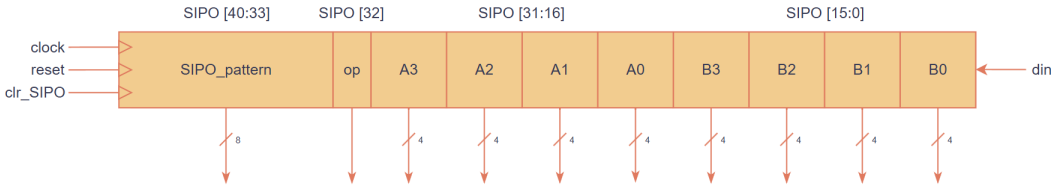

Figure 6: SIPO Block Diagram

The SIPO register supports two types of clearing mechanisms: a manual reset and an automatic clear. The manual reset is triggered by an active-high `reset` signal, which clears the entire register, including `SIPO[0]`.

For automatic clearing—performed once a valid pattern is detected and the PISO register is loaded with the ALU outputs—a more selective approach is required. Specifically, `SIPO[0]` must be preserved during this operation to avoid discarding the first bit of the next potential input stream.

To implement this, a separate signal, `clr_SIPO`, is used. All flip-flops in the SIPO register, except for `SIPO[0]`, are cleared when either `clr_SIPO` or `reset` is asserted. This is achieved by assigning the reset input of each flip-flop (excluding the first) to the logical OR of `clr_SIPO` and `reset`.

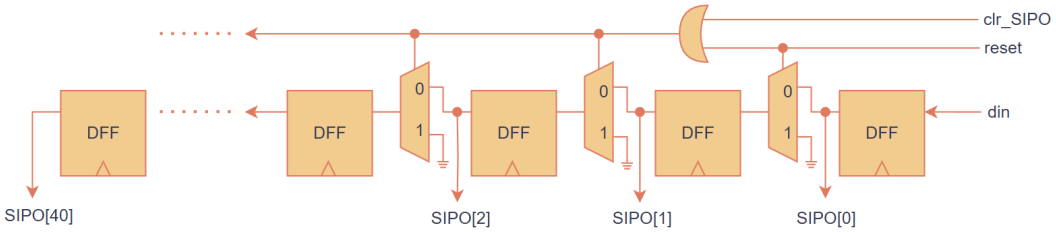

Figure 7: SIPO Reset/Clear Functionality

2.3 PISO Register

The PISO (Parallel-In Serial-Out) register is responsible for capturing parallel input data and shifting it out serially through its output. This design includes standard `load` and `reset` functionality, but—similar to the SIPO register—it continuously performs a left shift on every clock cycle. There is no dedicated shift-enable signal; shifting occurs automatically with each clock pulse. When the `load_PISO` signal is asserted high, the PISO register is loaded with a specific 28-bit value composed of a fixed 8-bit pattern followed by the 5-digit BCD output from the ALU. The layout of the register is as follows:

- `PISO[27:20]` – Predefined pattern sequence (e.g., `8'b10010110`)
- `PISO[19:0]` – 5-digit BCD output result (`F0-F4`)

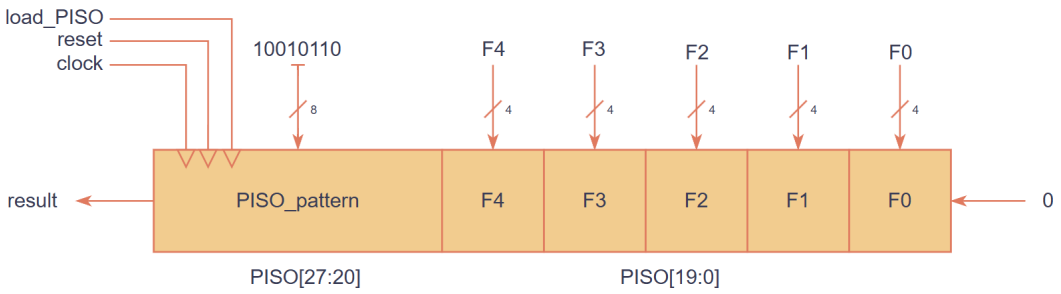

Figure 8: PISO Block Diagram

Once loaded, the register begins shifting left every clock cycle. The most significant bit is output serially, and the least significant bit is filled with zero after each shift, ensuring the register contents are eventually fully shifted out.

2.4 Controller

The controller in this design performs three primary tasks: pattern detection, asserting the `load_PISO` signal, and generating the `clr_SIPO` signal.

- **Pattern Detection:** The controller continuously monitors the most significant 8 bits of the SIPO register for a predefined pattern. An 8-input AND gate is used to detect the bit sequence `"10100101"`. When the pattern is matched, the controller prepares to trigger the appropriate control signals.
- **Load Signal:** The `load_PISO` signal is asserted high when a valid pattern is detected. This signal instructs the PISO register to load the output values from the ALU.
- **Clear Signal:** The `clr_SIPO` signal is simultaneously asserted to clear the SIPO register, allowing the system to receive the next input stream without falsely triggering on recurring patterns in the data stream (e.g., in `op`, `A`, or `B`).

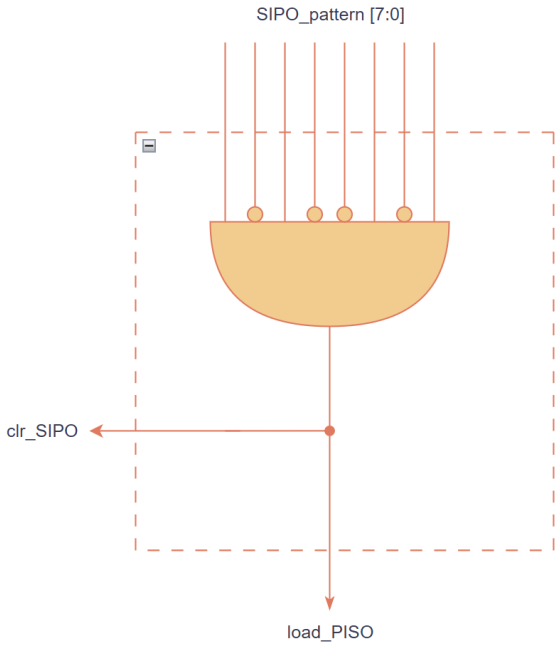

Figure 9: Controller Module

Both `load_PISO` and `clr_SIPO` are synchronized and sampled on the positive edge of the next clock cycle following the pattern detection. This ensures proper alignment of control flow between registers.

2.5 Top-Level Project3

The `Project3` module serves as the top-level integration point for all components of the system. It instantiates and connects the major submodules discussed previously, ensuring synchronized dataflow and control across the entire design.

Specifically, `Project3` performs the following:

- **SIPO Register:** Collects serial input data from `din` and shifts it into a 41-bit parallel format. It continuously shifts left on each clock cycle and is cleared either manually via a reset or automatically via the `clr_SIPO` signal when a valid pattern is detected.
- **Controller:** Monitors the upper 8 bits of the SIPO register for a specific pattern (e.g., `"10100101"`). Upon detecting this pattern, it asserts the `load_PISO` and `clr_SIPO` signals, triggering data capture into the PISO register and clearing the SIPO register to accept new input.
- **ALU:** Receives the 4-digit BCD operands (`A0-A3` and `B0-B3`) and the operation select bit (`op`) from the SIPO register. Based on `op`, it performs either BCD addition or subtraction, using digit-wise correction logic or 10's complement arithmetic as required.
- **PISO Register:** Loads the ALU's 5-digit BCD result (`F0-F4`) along with a predefined 8-bit header when `load_PISO` is asserted. It then continuously shifts the contents out serially, filling in with zeros as data is shifted.

The `Project3` module orchestrates these interactions using a shared clock and reset signal. All control and data signals are properly routed to ensure that input parsing, computation, and output transmission occur seamlessly and in synchronization.

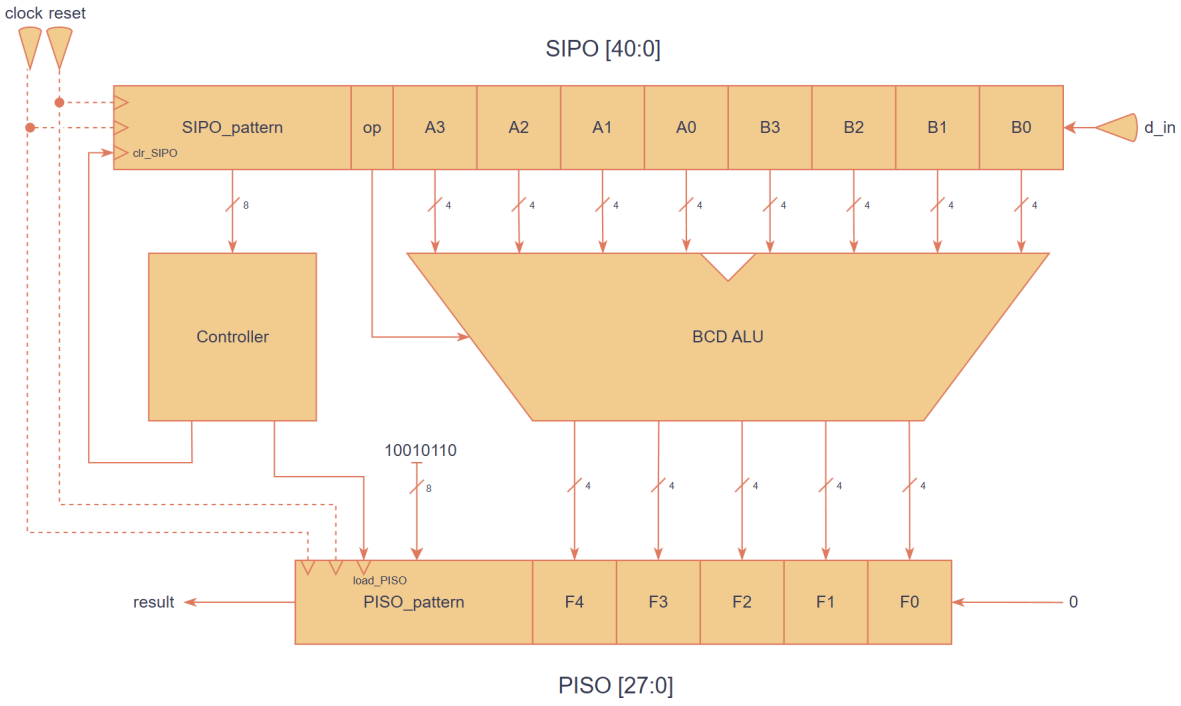

Figure 10: Project3

3 Testing

This section outlines the testing procedure used to validate the complete functionality of the system.

3.1 Intensive Testing

The system was tested using a single continuous stream of serial inputs, with separate test cases for both addition and subtraction operations. This approach closely simulates the intended behavior of the design, where inputs are continuously received and processed in real time. To comprehensively verify the system, the input stream was designed to test the following key functionalities:

- **Reset:** Asserting the `reset` signal high should initialize the system to a known state, clearing all D flip-flops and ensuring all registers are set to zero.
- **Shifting:** Both the SIPO and PISO registers must accurately shift data in and out. In this implementation, a left-shift is performed on each clock cycle. For the PISO register, zeros should be shifted into the least significant bit as higher bits are shifted out.
- **BCD Arithmetic:** The ALU must correctly perform BCD addition and subtraction, including proper handling of carry-outs and digit correction. Both operations are tested, and the `op` signal is verified to ensure it selects the correct arithmetic behavior.
- **Sequence Detection:** The controller must continuously monitor the upper bits of the SIPO register for a predefined pattern (`10100101`). Once detected, it should trigger the `load_PISO` and `clr_SIPO` signals to capture and process the input stream.
- **Unintended Sequence Detection (SIPO Clearing):** After detecting a valid pattern, the system must avoid falsely detecting that same pattern within the subsequent data fields (i.e., `op`, `A`, or `B`). This is resolved by clearing the SIPO register to prevent residual bits from accidentally triggering the pattern detector.
- **Back-to-Back Valid Inputs:** If a new valid input stream immediately follows a previous one, it must be processed correctly without being cut off. This is achieved by preserving the first bit of the SIPO register during automatic clearing, ensuring the next stream can begin uninterrupted.
- **Sequence Production:** Upon loading, the PISO register must prepend a fixed 8-bit pattern (`10010110`) to the outgoing stream, indicating the start of a valid message. This sequence is shifted out before the computed BCD output.

3.2 Selecting an Input Stream

To thoroughly validate all required functionality, a carefully structured serial input stream was selected. This stream contains two consecutive input streams, each followed by associated operation bits and operand values. The breakdown below explains each segment of the input and its role within the system, followed by descriptions on how it satisfies the predefined conditions:

⟵ Shifting Direction (MSB to LSB)

`0xA5`	`0`	`9`	`9`	`9`	`9`	`9`	`9`	`9`	`9`	`0xA5`	`1`	`3`	`2`	`9`	`5`	`1`	`9`	`9`	`9`
Pattern	**op**	**A = 9999**				**B = 9999**				**Pattern**	**op**	**A = 3295**				**B = 1999**			
Expected Output = 9999 + 9999 = 19998										**Expected Output = 3295 - 1999 = 1296**									

Table 3: Selected input stream shown in serial shift-in order

- **BCD Arithmetic:** Both BCD addition and subtraction operations are tested. Each stream includes sufficient carry propagation to verify proper operation of the ALU, including digit-wise correction and carry-in functionality.
- **Sequence Detection:** The predefined detection pattern `10100101` (`0xA5`) is present at the start of both input streams, confirming that the controller properly identifies valid input sequences.

- **Unintended Sequence Detection (SIPO Clearing):** In the subtraction stream, values such as 2, 9, and 5 produce the binary sequence `001010010100`, which contains the valid pattern `10100101` internally. This tests the controller's ability to prevent unintended retriggering by properly clearing the SIPO register after a pattern is detected, while preserving the first bit.
- **Back-to-Back Valid Inputs:** The subtraction stream follows immediately after the addition stream, with no idle cycles in between. This confirms that the system can handle consecutive valid input sequences without misalignment or data corruption.

4 Results

The three waveforms shown below illustrate the full testing sequence, including all relevant inputs, outputs, and internal control signals. The signals are color-coded for clarity and follow the order in which the serial input stream is processed. The table below describes the meaning of each color-coded signal and how it should be interpreted in the waveform:

Color	Signal(s)	Interpretation
Green	`reset, clock, din`	Input signals. `reset` initializes the system, `clock` drives all sequential logic, and `din` provides the serial input stream.
Cyan	`result`	Final serial output of the system, shifted out from the PISO register.
Maroon	`SIPO_pattern (SIPO[40:33])`	Represents the upper bits of the SIPO register. When equal to `8'b10100101` (A5 in hex), the controller should assert `load_PISO` and `clr_SIPO`.
Blue	`op (SIPO[32])`	Operation selector. When high, subtraction is performed; when low, addition is performed.
Purple	`A0-A3 (SIPO[31:16])`	4-digit BCD input A. These are the operands sent to the ALU for processing.
Yellow	`B0-B3 (SIPO[15:0])`	4-digit BCD input B. These are subtracted from or added to input A based on the `op` value.
Orange	`F0-F4 (ALU Output)`	5-digit BCD result from the ALU. This value is loaded into the PISO when a valid pattern is detected.
Pink	`load_PISO, clr_SIPO`	Control signals asserted when a valid pattern is detected (i.e., `SIPO_pattern` = A5). `load_PISO` loads ALU outputs into the PISO register, while `clr_SIPO` clears the SIPO register (excluding `SIPO[0]`).

Table 4: Color-coded signal interpretation for waveform visualization

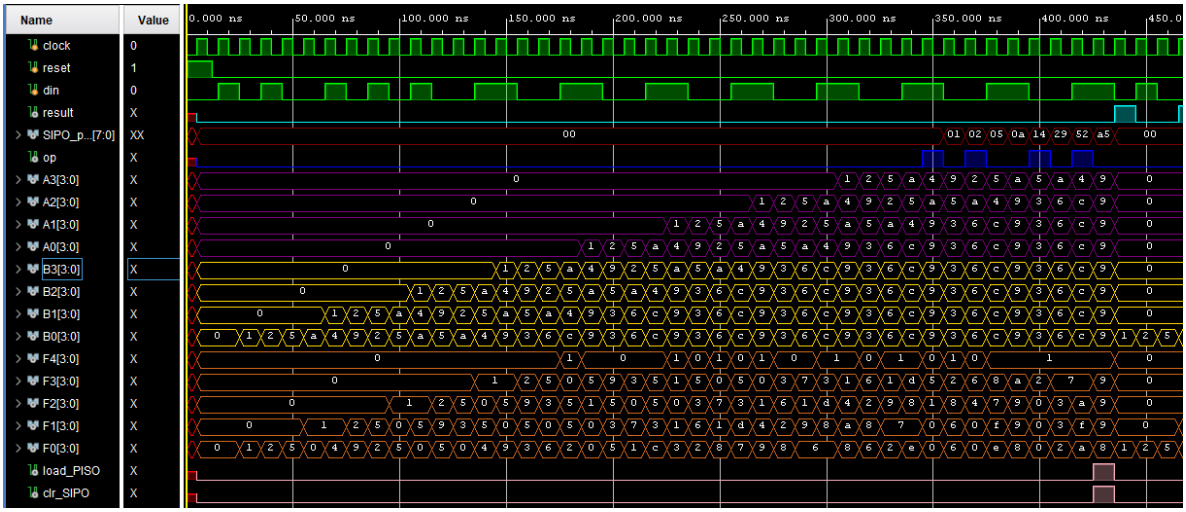

Figure 11: Addition Input Stream

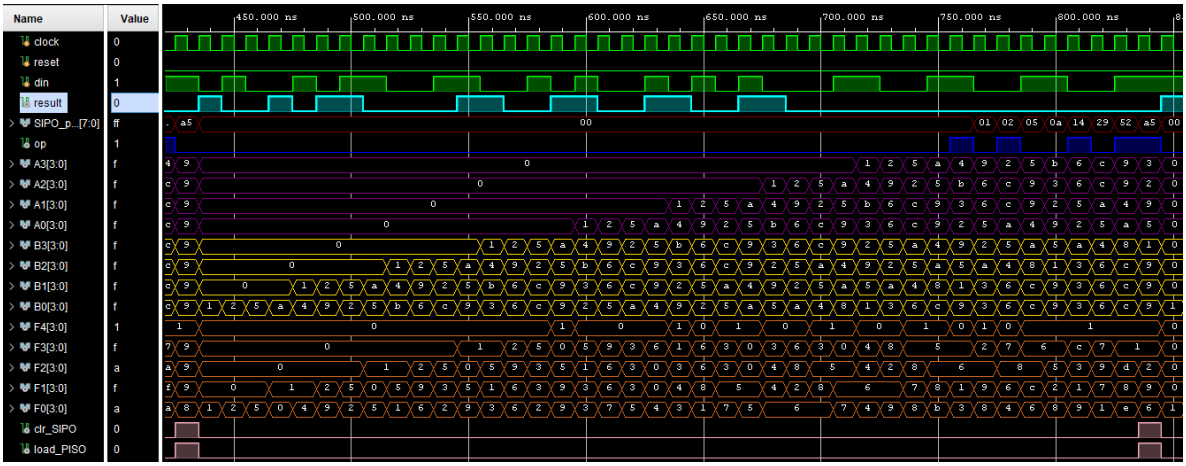

Figure 12: Addition Output/Subtraction Input Streams

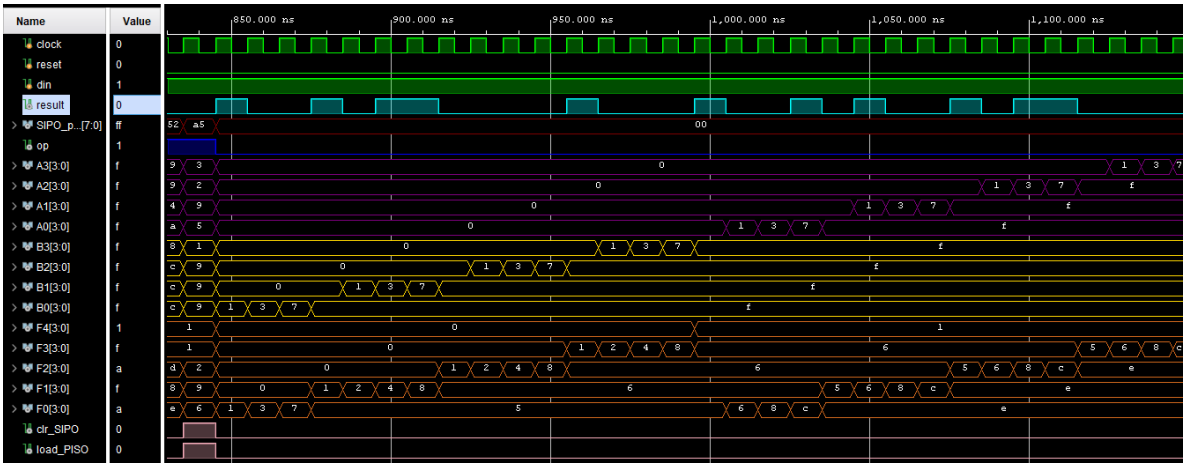

Figure 13: Subtraction Output Stream

4.1 Figure 11: Addition Input Stream

Figure 11 illustrates the system undergoing an initial reset to place all modules in a known state, effectively validating the reset functionality. Immediately afterward, the SIPO register begins to receive and shift in the input stream intended for the addition operation. The shifting behavior is clearly visible, confirming the correct operation of the SIPO register. A key moment occurs when the `SIPO_pattern` equals `0xA5`. At this point, both `load_PISO` and `clr_SIPO` signals are asserted high, verifying the controller's ability to detect and respond to a specific input sequence. Simultaneously, the ALU outputs can be observed—yielding the expected result of $F = 19998$, which matches the expected outputs (see Table 3). The corresponding serial output via `result` related to this result will be addressed in the Figure 12 discussion.

The following summarizes the functionalities validated by this input stream:

- **Reset:** The system transitions from an undefined to a known state at the beginning of the waveform, confirming the reset signal's effectiveness.
- **SIPO Shifting:** The input stream is shifted into the SIPO register on each clock cycle, validating proper shifting behavior.
- **Sequence Detection:** When `0xA5` is detected, the `load_PISO` and `clr_SIPO` signals are asserted, confirming successful sequence detection by the controller.
- **BCD Addition:** The BCD computation and conversion process produced the correct digit values along with the appropriate carry-outs. Additionally, the `op` selection mechanism functioned as expected, validating the system's BCD addition capability.

4.2 Figure 12: Addition Output/Subtraction Input Streams

Figure 12 illustrates the system's behavior immediately following the events of Figure 11. Here, both the serial output resulting from the addition operation and the input stream for the subsequent subtraction operation can be observed.

For the serial output stream seen on `result`, the predefined sequence `10010110` appears on the clock cycle immediately following the loading of the PISO register. This indicates the beginning of a valid output stream and confirms both the correct operation of the PISO register's shifting functionality and the system's ability to produce the expected output sequence. Additionally, the subsequent data stream corresponds to the binary representation of the ALU outputs from the addition operation, further validating the PISO shifting mechanism and overall output generation.

In terms of the subtraction operation, a valid input stream is detected right after the PISO register is loaded (i.e., when `din` equals `0xA5`). At this point, the system autonomously clears the SIPO register, retaining only the initial bit to prevent premature detection of the next input pattern. This is clearly demonstrated when the valid pattern reaches the end of the SIPO register, triggering the assertion of both `load_PISO` and `clr_SIPO`. Notably, an unintended sequence briefly appears in `B0` and `B1` at 100ns, but the clearing mechanism prevents this invalid sequence from being processed. Simultaneously, the ALU outputs are observed to produce the correct result of $F = 1296$ (see Table 3), along with a carry-out of 1, confirming that a subtraction operation was successfully executed. The corresponding serial output for this result will be discussed in the Figure 13 section.

The following summarizes the functionalities validated by these input/output sequences:

- **Sequence Production:** The predefined sequence `10010110` is generated immediately after the PISO register is loaded, thoroughly validating the sequence production functionality.
- **Sequence Detection:** Upon detecting `0xA5`, the controller asserts `load_PISO` and `clr_SIPO`, confirming proper sequence detection behavior.
- **PISO Shifting:** The PISO register accurately loads ALU output values and shifts them out through `result`, validating its serial shifting functionality.
- **Back-to-Back Input Handling:** The system successfully processes two consecutive input streams without data loss, confirming reliable continuous operation.
- **Unintended Sequence Prevention:** The clearing mechanism prevents an unintended valid pattern from reaching the end of the SIPO register, effectively demonstrating robust sequence filtering.
- **BCD Subtraction:** The BCD subtraction operation produces correct digit values and a carry-out of 1 (as expected for subtraction). The `op` signal behaves as intended, validating the subtraction capability of the ALU.

4.3 Figure 13: Subtraction Output Stream

Figure 13 illustrates the system's behavior immediately following the events of Figure 12. In this stage, the serial output corresponding to the subtraction operation is visible, while the current input stream on `din` can be disregarded.

On the clock cycle immediately following the loading of the PISO register, the `result` signal begins outputting the serial data, starting with the predefined sequence `10010110`, which indicates the beginning of a valid message. The subsequent data stream aligns precisely with the expected ALU outputs from the subtraction operation. The conclusion of this serial output marks the end of the test procedure.

The following summarizes the functionalities validated by these input/output sequences:

- **Sequence Production:** The predefined sequence `10010110` is generated immediately after the PISO register is loaded, thoroughly validating the system's ability to initiate a valid output sequence.
- **PISO Shifting:** The PISO register correctly loads and serially shifts the ALU output values through `result`, confirming the proper operation of its shifting mechanism.

5 Synthesis Analysis

This section will discuss the vivado synthesis results and how they compare to the inteded design. One thing to note, is my design does not use any behavioral implementations, so the synthesis compiler had little freedom in what it generated.

5.1 BCD ALU

The synthesized design of the BCD alu is seen below, see Figure 3 for the intended design. The synthesized design exactly matches the intended design.

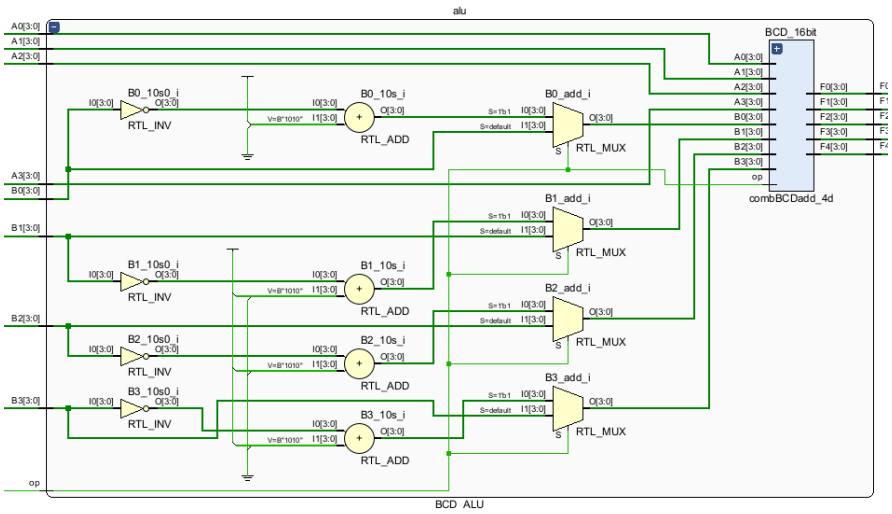

Figure 14: Synthesized BCD ALU

Additionally, the synthesized design for the 4 digit BCD adder is seen below, see Figure 5 for the intended design.

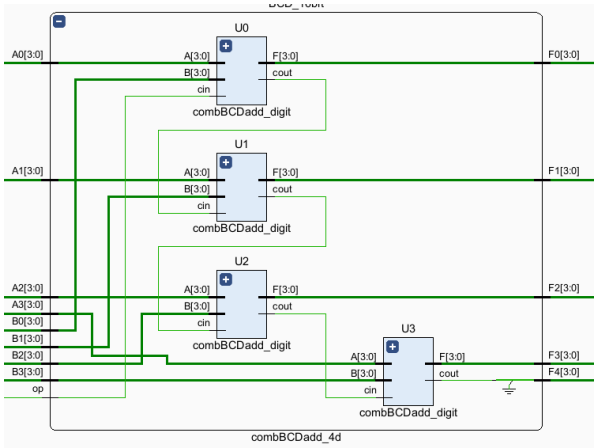

Figure 15: Synthesized 4-digit BCD Adder

5.2 SIPO Register

The top-portion of the synthesized design of the SIPO register is seen below (the rest is cut off for ease of reading), see Figure 7 for the intended design. The synthesized design is very close to the intended design, the only noticeable difference is that the mux is within the DFF module, however this does not effect the funcitonality whatsoever.

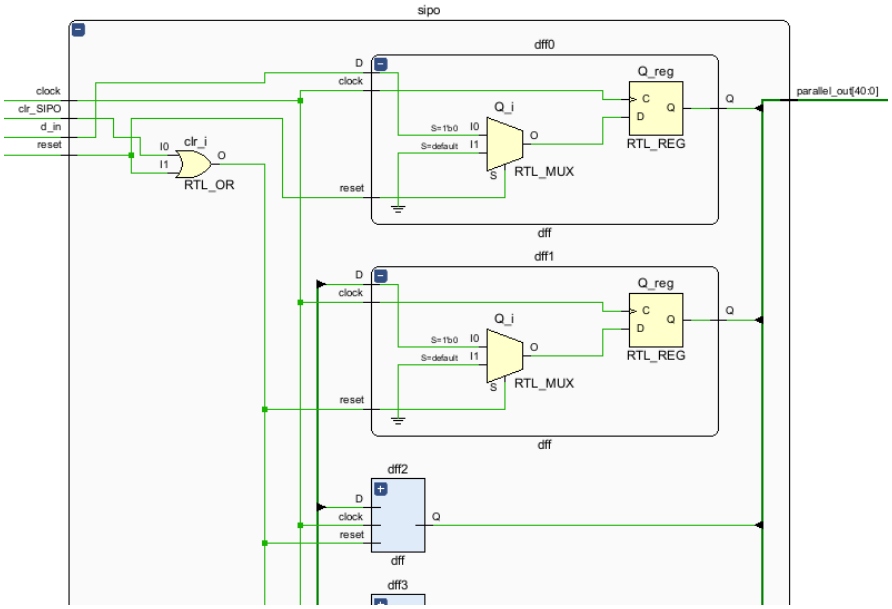

Figure 16: Synthesized SIPO Register

5.3 PISO Register

The synthesized design of PISO register is seen below, see Figure 8 for the intended design. The synthesized design exactly matches the intended design.

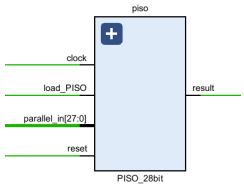

Figure 17: Synthesized PISO Register

5.4 Controller

The synthesized design of the controller is seen below, see Figure 9 for the intended design. There are couple small notable differences between the intended and synthesized design, although nothing that affects the functionality. The first thing is that the synthesized does not explicitly use an AND gate for checking the pattern (although that is the most likely way it will be implemented). The other difference, is that the predefined PISO pattern is generated in here rather than outside the PISO register.

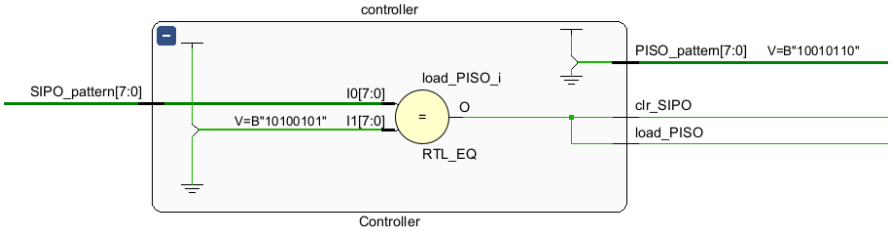

Figure 18: Synthesized Controller

5.5 Top-level Project3

he synthesized design of the Project3 module is seen below, see Figure 10 for the intended design. The synthesized design exactly matches the intended design.

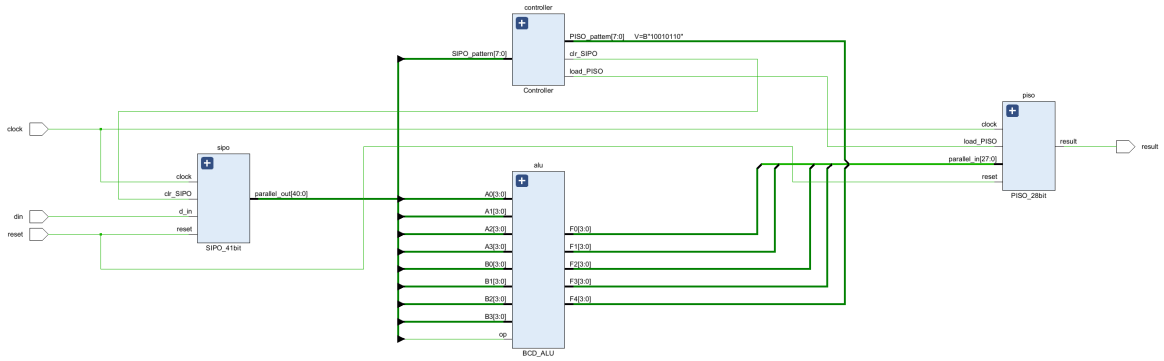

Figure 19: Synthesized Project3 module

6 Conclusion

This project successfully implemented a fully functional serial BCD arithmetic system composed of a 41-bit SIPO register, 28-bit PISO register, BCD ALU, and controller module. The system was designed to continuously process serial inputs, detect specific instruction patterns, execute arithmetic operations, and return serialized results. A carefully structured testing procedure was employed using a continuous stream of serial data, designed to validate all key functionalities—including reset behavior, shifting operations, BCD arithmetic, pattern detection, sequence production, and robust handling of back-to-back input streams.

The waveform analysis confirmed that each module behaved as expected, with all control signals and outputs matching the predicted behavior. The ALU accurately computed both addition and subtraction results, with proper carry handling and correction logic for BCD compliance. Sequence detection and automatic clearing of the SIPO register worked flawlessly, even in cases with potentially conflicting data patterns, demonstrating the system's robustness in avoiding unintended triggers.

Synthesis results closely matched the intended design diagrams for each module. Minor structural differences introduced by the synthesis tool—such as internal mux placement or signal routing—did not impact the overall functionality. In all cases, the synthesized netlists preserved the logical structure and behavior of the design, validating both the architectural and implementation correctness of the system. Overall, the project achieved its objectives and demonstrated a reliable and efficient design pipeline from theory to simulation to hardware synthesis.

The report was compiled using LaTeX .