Introduction

This lab implements and verifies a two-digit Binary-Coded Decimal (BCD) arithmetic unit on a Digilent Nexys A7 FPGA using a top-down, modular VHDL design.

Each BCD digit is encoded on four bits (0–9). Addition is performed digit-wise with the standard BCD correction: if a raw binary digit sum exceeds 9, +6 is added and the carry propagates.

Subtraction is implemented via the radix-10 (ten's) complement:

P−Q = P + (10^n − Q); for example, with two digits P=21 and Q=46, the ten's complement of Q is 54, so 21 + 54 = 75

The top level exposes a compact hardware interface suitable for on-board testing: a 100 MHz system clock and synchronous control input to select add/subtract, sixteen slide switches forming two 2-digit BCD operands (A1A0 and B1B0), and multiplexed seven-segment display outputs (four anodes, seven cathodes).

The design follows a structural style with clearly separated entities. Wherever possible, previously verified modular blocks from earlier labs are reused such as the digit counter, anode decoder, 4×4 mux, and hex-to-SSD decoder to accelerate development, improve readability, and reduce verification effort.

Design Description

Top level - Switch inputs (two BCD digits each for A and B) pass through a **BCD validator**, then a **two-digit adder/subtractor**, and finally a **hex-to-SSD** block. A **counter + 4×4 mux** time-multiplex the three result digits onto the SSD. If any input digits isn't valid BCD, the display shows "E".

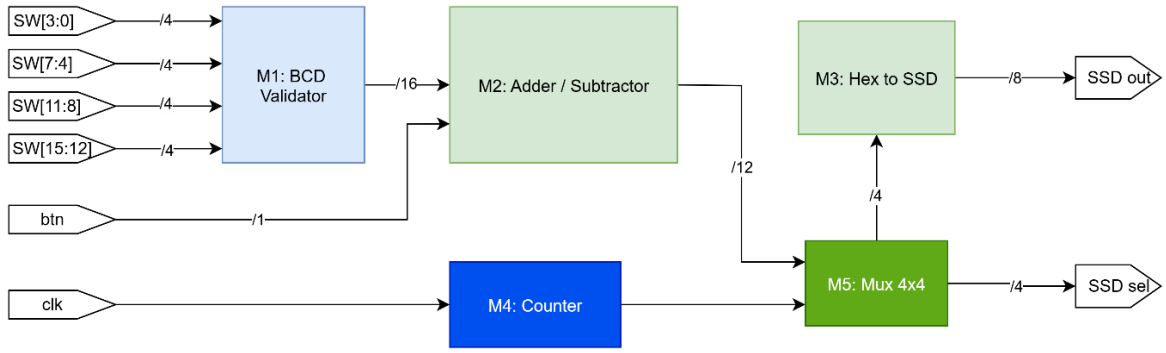

Figure 1 Top level Block Diagram

Adder / Subtractor sub system - Subtraction is done with **10's complement**: pass B unchanged for add or complement B for subtract. Two cascaded **one-digit BCD adders** compute units then tens, the final carry forms the hundreds digit.

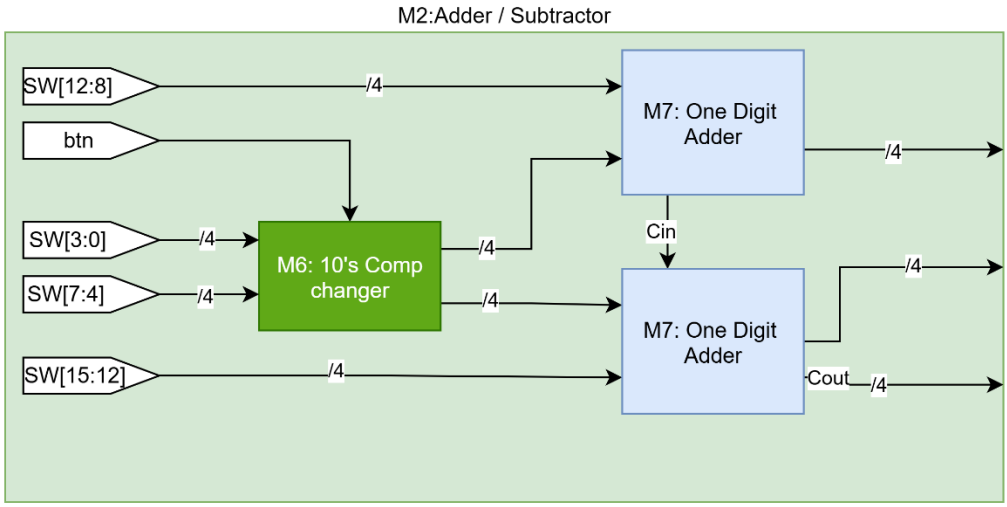

Figure 2 Adder / Subtractor Block Diagram

One Digit Adder - A **4-bit ripple-carry** chain of full adders produces the raw sum.

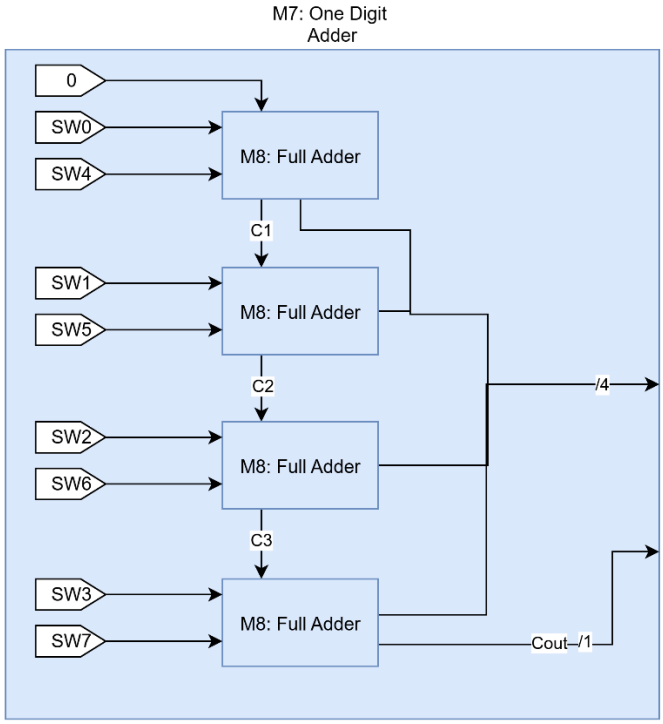

Figure 3 One Digit Adder Block Diagram

Simulation Results

In the testbench we define P = B1B0 (SW15:12 tens, SW11:8 ones) and Q = A1A0 (SW7:4 tens, SW3:0 ones). With sub_mode='0' the unit performs BCD addition; with sub_mode='1' it performs BCD subtraction via 10's-complement.

Four scenarios were exercised:

(i) **Adder, P>Q:** 45+23=68 (Fig. 4, 20–40 ns)
(ii) **Adder, P<Q:** 57+68=125 with carry into the hundreds digit (40–60 ns)
(iii) **Subtractor, P>Q:** 75−32=143 (80–120 ns)
(iv) **Subtractor, P<Q:** 23−67=56 (120–140 ns)

The raw subtract waveforms show **143** and **56** because the core uses radix-10 complement internally: for P≥Q the result appears as "1xx" (final carry=1) and for P<Q the final carry=0 and the two BCD digits are a **10's-complement magnitude** (100−44=56).

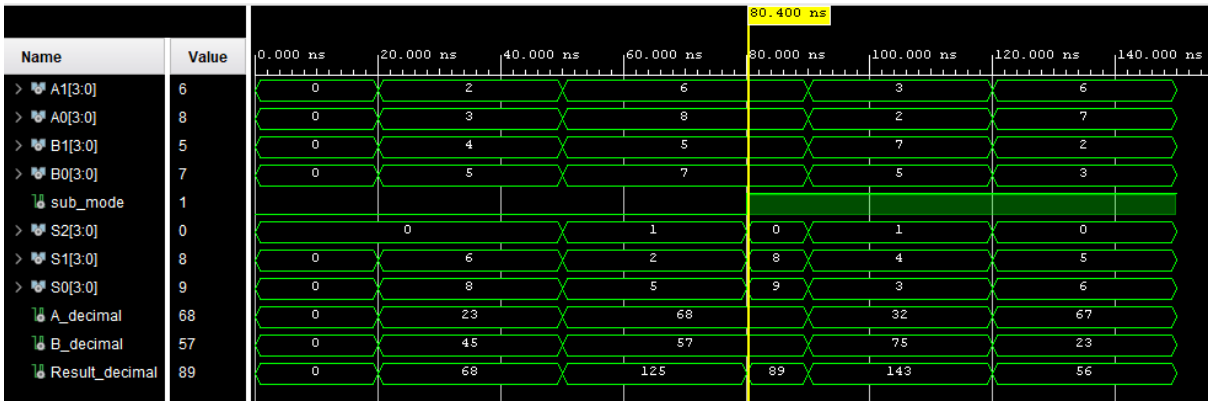

Figure 4 Simulation results

On-Board Testing

Adder, P>Q: 45+23=68

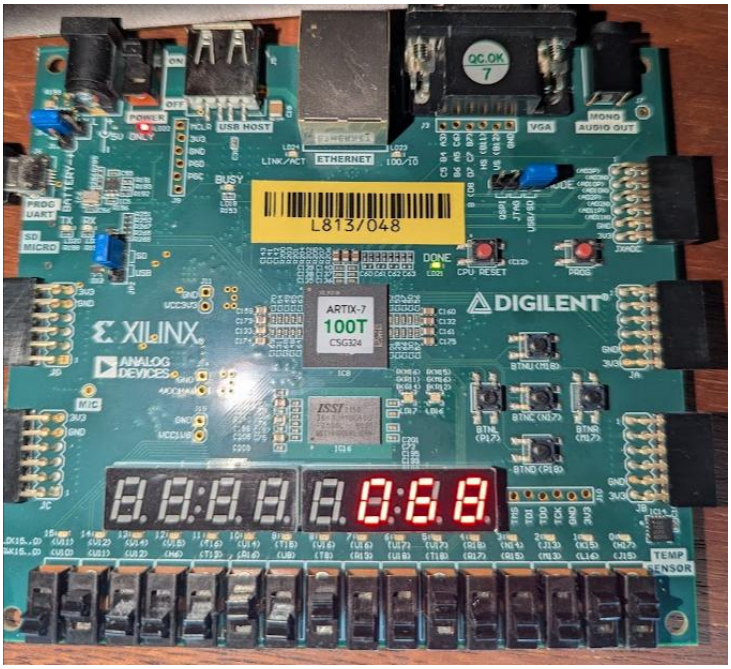

Figure 5 On board 45 + 23

Adder, P<Q: 57+68=125 with carry into the hundreds digit

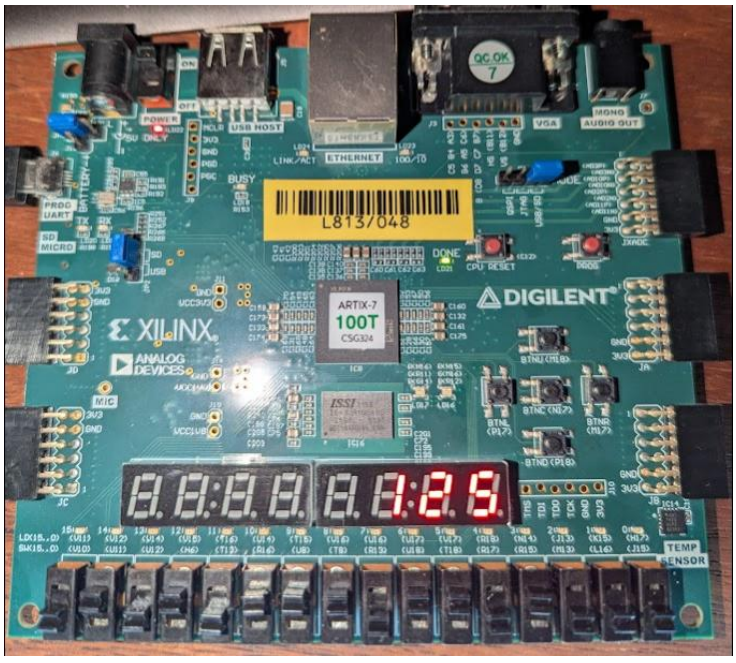

Figure 6 On board 57 + 23

Subtractor, P>Q: 75−32=143

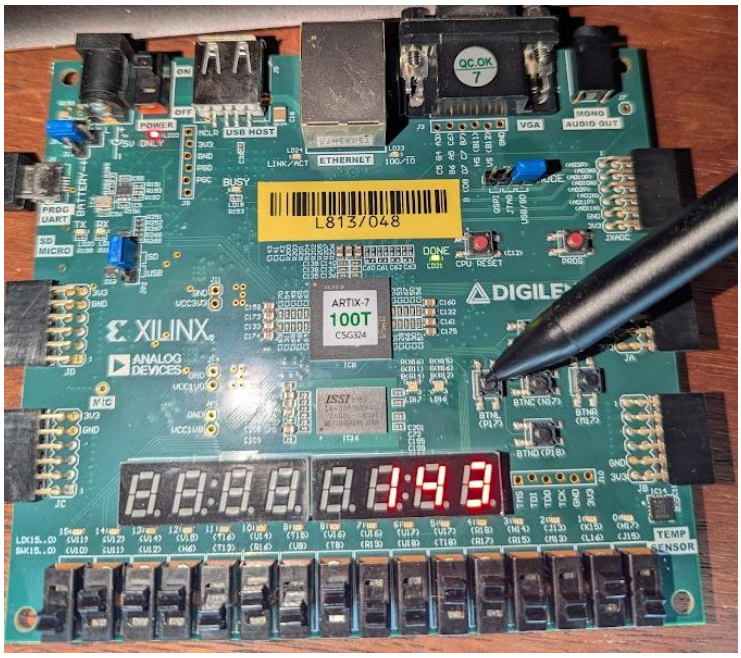

Figure 7 On board 75−32

Subtractor, P<Q: 23−67=56

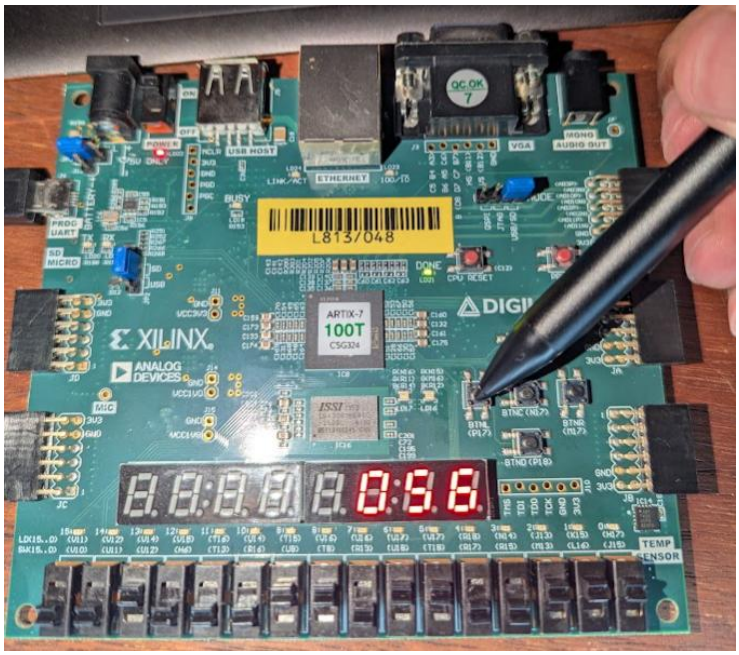

Figure 8 On board 23−67

Synthesis/Implementation Results

The RTL schematic (Fig. 9) matches the block diagram (Fig. 1): four bcd_validator blocks gate the inputs and form all_valid, which drives a single BCD_Adder_Subtractor_2digit to produce S2:S1:S0. Small muxes implement the “E on invalid” path, while a digit_counter with anode_decoder and mux4x4 handles SSD scanning exactly as specified. hex_to_ssd_const maps the selected digit to segments, only the right-hand three digits are used, and the left display is held off—consistent with the design intent.

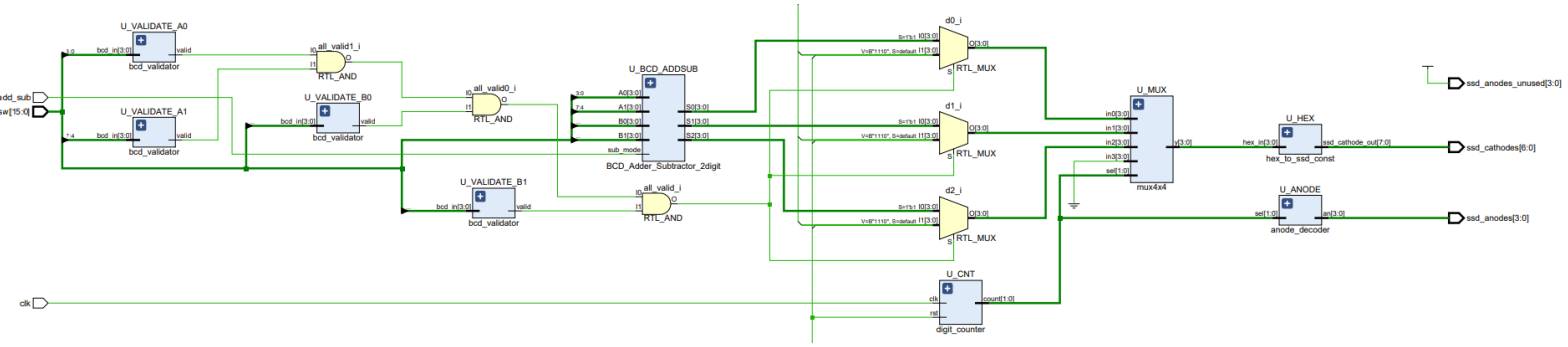

Figure 9 RTL schematic Top

The RTL schematic of the adder and subtractor circuit is shown in figure 10, which also implemented the radix conversion and addition of the minus sign.

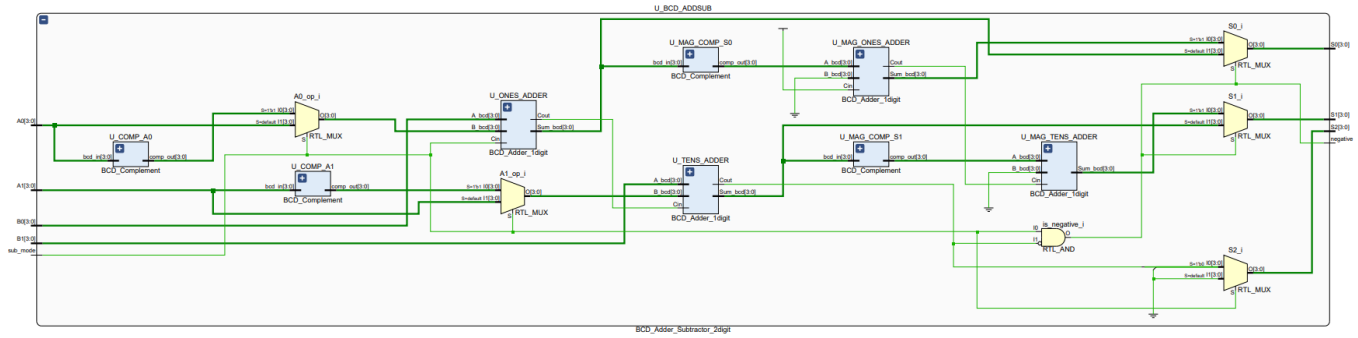

Figure 10 RTL schematic of Adder / Subtractor

FPGA resources comparison

Using the structural design for the 1-digit adder as seen in appendix D, using the command

“synth_design -top BCD_Adder_1digit” and “report_utilization”

Resulted in using 6 Look up tables as seen in figure 11. Using additional commands like “-directive AreaOptimized_high” and “-directive PerformanceOptimized” resulted in the same usage.

7. Primitives

Ref Name	Used	Functional Category
IBUF	9	IO
OBUF	5	IO
LUT6	4	LUT
LUT5	2	LUT
LUT3	1	LUT

Figure 11 Primitives usage

Changing to a Behavioural approach using the following code in figure 12, also resulted in using 6 Look up tables as well even with the additional commands.

```
architecture Behavioral of BCD_Adder_1digit_behavioral is
begin
    process(A_bcd, B_bcd, Cin)
        variable temp_sum : unsigned(4 downto 0);  -- 5-bit to capture carry
        variable A_int    : integer range 0 to 15;
        variable B_int    : integer range 0 to 15;
        variable Cin_int  : integer range 0 to 1;
        variable result   : integer range 0 to 31;
    begin
        -- Convert inputs to integers
        A_int := to_integer(unsigned(A_bcd));
        B_int := to_integer(unsigned(B_bcd));
        Cin_int := to_integer(unsigned'("" & Cin));

        -- Perform addition
        result := A_int + B_int + Cin_int;

        -- Generate BCD output and carry
        if result > 9 then
            Sum_bcd <= std_logic_vector(to_unsigned(result - 10, 4));
            Cout <= '1';
        else
            Sum_bcd <= std_logic_vector(to_unsigned(result, 4));
            Cout <= '0';
        end if;
    end process;
end Behavioral;
```

Figure 112 Behavioural 1 digit adder

This could be due to Vivados internal performs optimisation on small blocks. For this sub system, structural vs behavioural VHDL makes no resource difference and only for readability or need for hierarchy.

Discussion and Conclusion

This lab delivered a modular, two-digit BCD adder/subtractor on the Nexys A7 using structural VHDL with re-used building blocks.

The design met its functional goals in simulation and on hardware.

Synthesis of both the 1-digit adder and the top-level adder/subtractor showed that Vivado already infers an optimal carry-chain mapping.

The +6 BCD-correction was implemented using a ripple-carry adder stage that conditionally adds "0110". A lighter alternative would have been to use two full adders to +6. Seeing as the utilization report stated that only 6 LUT’s were being used, Vivado could already have optimised it.

If time permits, I would present true negative results by lighting the minus segment and converting from 10’s complement back to signed BCD before display.

References

Anode_decoder, hex_to_ssd_const, mux4x4, digit_counter and bcd_validator VHDL descriptions were reused from previous pracs.

Appendix

A – BCD_Adder_1digit using Ripple Carry Adder

```
begin
    -- Step 1: Perform standard 4-bit binary addition
    U_BINARY_ADD: entity work.RCAdder
        generic map (width => 4)
        port map (
            Cin  => Cin,
            A    => A_bcd,
            B    => B_bcd,
            S    => binary_sum,
            Cout => binary_carry
        );

    -- Step 2: Determine correction value
    -- If sum > 9 OR there's a carry from binary addition, we need correction
    correction <= "0110" when (unsigned(binary_sum) > 9 or binary_carry = '1') else "0000";

    -- Step 3: Add correction using another RCAdder
    U_CORRECTION_ADD: entity work.RCAdder
        generic map (width => 4)
        port map (
            Cin  => '0',
            A    => binary_sum,
            B    => correction,
            S    => corrected_sum,
            Cout => correction_carry
        );

    -- Step 4: Generate outputs
    Sum_bcd <= corrected_sum;
    Cout <= binary_carry or correction_carry;  -- Carry-out if either addition generated carry

end Structural;
```

B – BCD_Complement

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;

entity BCD_Complement is
    port (
        bcd_in  : in  std_logic_vector(3 downto 0);  -- Input BCD digit
        comp_out : out std_logic_vector(3 downto 0)  -- 9's complement output
    );
end BCD_Complement;

architecture Behavioral of BCD_Complement is
begin
    process(bcd_in)
    begin
        case bcd_in is
            when "0000" => comp_out <= "1001";  -- 9 - 0 = 9
            when "0001" => comp_out <= "1000";  -- 9 - 1 = 8
            when "0010" => comp_out <= "0111";  -- 9 - 2 = 7
            when "0011" => comp_out <= "0110";  -- 9 - 3 = 6
            when "0100" => comp_out <= "0101";  -- 9 - 4 = 5
            when "0101" => comp_out <= "0100";  -- 9 - 5 = 4
            when "0110" => comp_out <= "0011";  -- 9 - 6 = 3
            when "0111" => comp_out <= "0010";  -- 9 - 7 = 2
            when "1000" => comp_out <= "0001";  -- 9 - 8 = 1
            when "1001" => comp_out <= "0000";  -- 9 - 9 = 0
            when others => comp_out <= "0000";  -- Invalid BCD -> 0
        end case;
    end process;
end Behavioral;
```

C – BCD_Adder_Subtractor_2digit

```
begin
    -- Generate 9's complement of A (instead of B)
    U_COMP_A0: entity work.BCD_Complement
        port map (
            bcd_in => A0,
            comp_out => A0_comp
        );

    U_COMP_A1: entity work.BCD_Complement
        port map (
            bcd_in => A1,
            comp_out => A1_comp
        );
    -- Select operand based on add/subtract mode
    A0_op <= A0_comp when sub_mode = '1' else A0;  -- Use complement for subtraction
    A1_op <= A1_comp when sub_mode = '1' else A1;  -- Use complement for subtraction

    -- Initial carry: +1 for subtraction (to complete 10's complement), 0 for addition
    initial_carry <= sub_mode;

    -- Ones position: B0 + A0_op + initial_carry
    -- For addition: B0 + A0
    -- For subtraction: B0 + (10's complement of A0) = B0 - A0
    U_ONES_ADDER: entity work.BCD_Adder_1digit
        port map (
            A_bcd   => B0,                    -- B is now the main operand
            B_bcd   => A0_op,                 -- A is complemented for subtraction
            Cin     => initial_carry,         -- +1 for subtraction, 0 for addition
            Sum_bcd => S0,                    -- Ones digit of result
            Cout    => carry_intermediate     -- Carry to tens position
        );

    -- Tens position: B1 + A1_op + carry from ones
    U_TENS_ADDER: entity work.BCD_Adder_1digit
        port map (
            A_bcd   => B1,                    -- B is now the main operand
            B_bcd   => A1_op,                 -- A is complemented for subtraction
            Cin     => carry_intermediate,    -- Carry from ones position
            Sum_bcd => S1,                    -- Tens digit of result
            Cout    => S2(0)                  -- Carry becomes hundreds digit
        );
    -- Hundreds digit is just the carry from tens (0 or 1)
    S2(3 downto 1) <= "000";  -- Upper bits always 0
end Structural;
```

D – BCD_Adder_1digit

```
begin
    -- Step 1: Perform standard 4-bit binary addition
    U_BINARY_ADD: entity work.RCAdder
        generic map (width => 4)
        port map (
            Cin  => Cin,
            A    => A_bcd,
            B    => B_bcd,
            S    => binary_sum,
            Cout => binary_carry
        );

    -- Step 2: Determine correction value
    -- If sum > 9 OR there's a carry from binary addition, we need correction
    correction <= "0110" when (unsigned(binary_sum) > 9 or binary_carry = '1') else "0000";

    -- Step 3: Add correction using another RCAdder
    U_CORRECTION_ADD: entity work.RCAdder
        generic map (width => 4)
        port map (
            Cin  => '0',
            A    => binary_sum,
            B    => correction,
            S    => corrected_sum,
            Cout => correction_carry
        );

    -- Step 4: Generate outputs
    Sum_bcd <= corrected_sum;
    Cout <= binary_carry or correction_carry;  -- Carry-out if either addition generated carry

end Structural;
```

E – RCAdder

```
architecture Structural of RCAdder is
  component FullAdder is
    port (
      A, B, Cin : in  std_logic;
      S, Cout   : out std_logic
    );
  end component;

  signal C : std_logic_vector(width downto 0);
begin
  C(0) <= Cin;

  gen_adders : for i in 0 to width-1 generate
    fa : FullAdder
      port map (
        A   => A(i),
        B   => B(i),
        Cin => C(i),
        S   => S(i),
        Cout=> C(i+1)
      );
  end generate gen_adders;

  Cout <= C(width);
end Structural;
```