Introduction

This lab implements a custom computing hardware architecture for finding the greatest common divisor (GCD) of two 8-bit unsigned numbers using the Euclidean algorithm. The design follows a data path and controller approach with a finite state machine (FSM) controlling the computation process. The system takes inputs from slide switches, processes them through a dedicated data path, converts the result to BCD format, and displays it on seven-segment displays.

Design Description

The system architecture consists of two main components: the data path figure 1 and the controller (FSM) figure 2. The block diagram below shows the interconnection between these components and all necessary sub-systems.

The data path contains the following components:

- Two 8-bit registers (A and B) to store input values and intermediate results
- Two 2-to-1 multiplexers for selecting between initial inputs and computed values
- Two 8-bit subtractors for performing the Euclidean algorithm operations
- A comparator to determine the relationship between A and B (greater than, less than, or equal)
- A binary-to-BCD converter to prepare the result for display

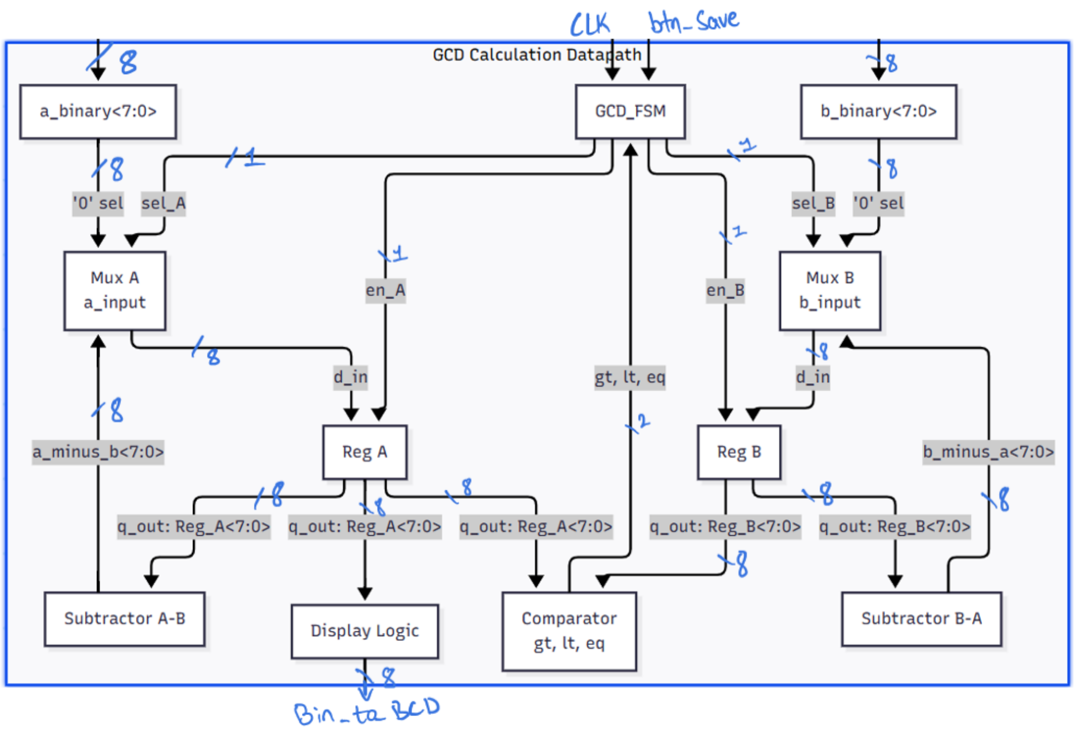


Figure 1 Data Path

The controller is implemented as a finite state machine with the following states:

- IDLE: Waiting for start signal
- LOAD: Loading initial values from switches into registers
- CHECK: Comparing values in registers A and B
- SUB_A_B: Performing A = A - B when A > B
- SUB_B_A: Performing B = B - A when B > A
- FINISH: Computation complete, result available

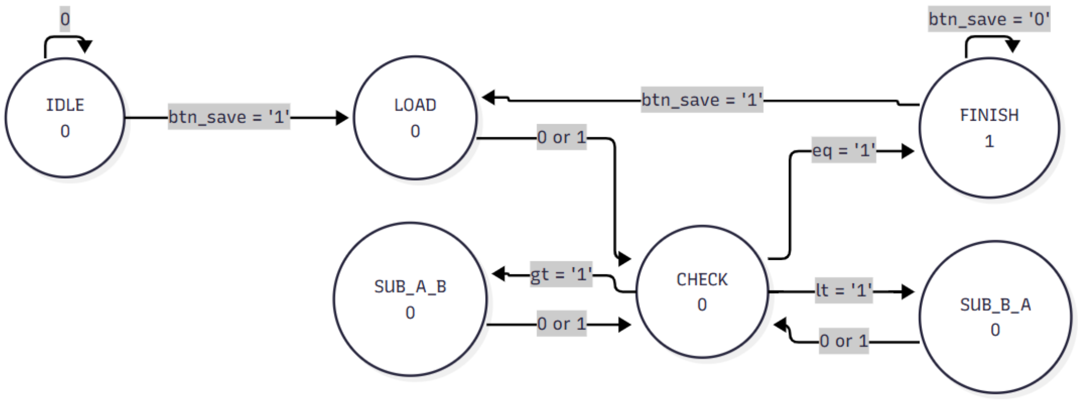


Figure 2 FSM

A significant design decision was the implementation of the arithmetic operations. Rather than building custom ripple-carry subtractors from basic gates, the design leverages the built-in arithmetic operators of VHDL and the numeric_std package using behavioural instead of structural approach. As last practical highlighted that the FPGA resources don’t differ, with small block.

Simulation Results

In the testbench we define

Four scenarios were exercised:

(i) **A > B:** GCD(56,42) = 14

Input: A = 56 (0x38), B = 42 (0x2A)

Expected Result: GCD = 14 (0x0E)

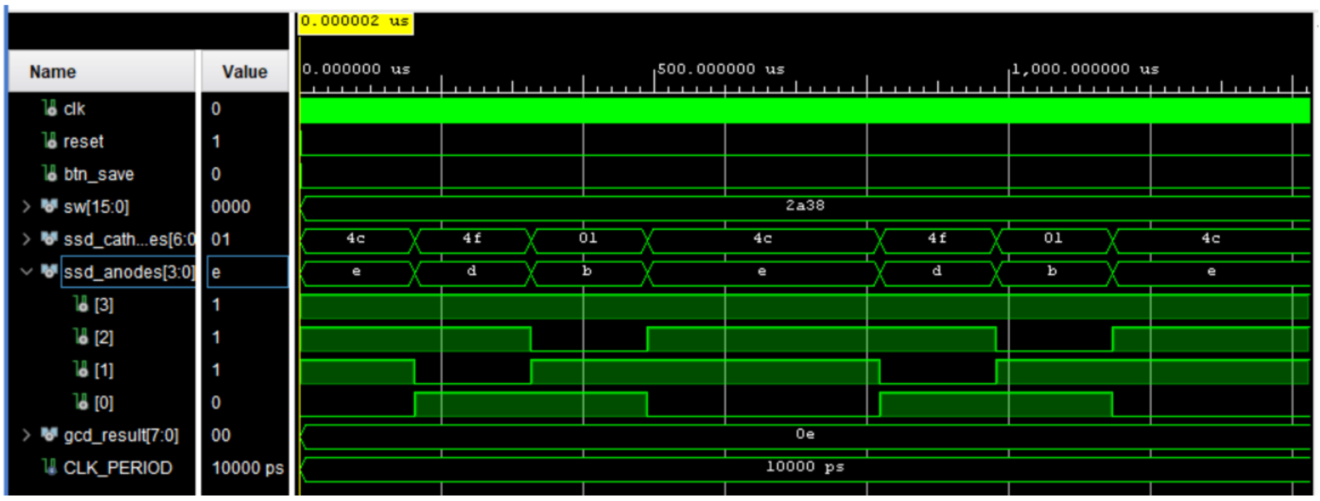


Figure 3 Test A > B

(ii) **A < B:** GCD(30, 75) = 15

Input: A = 30 (0x1E), B = 75 (0x4B)

Expected Result: GCD = 15 (0x0F)

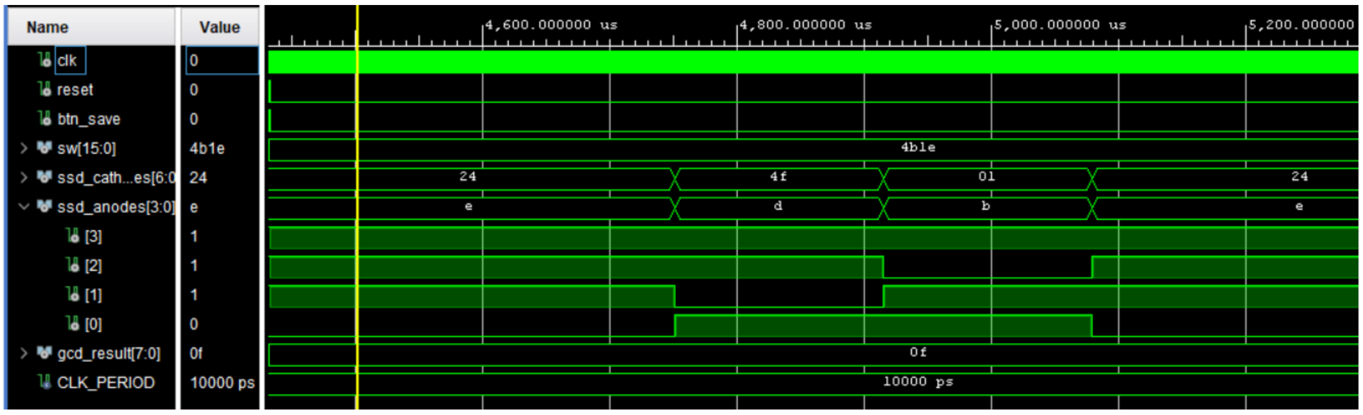
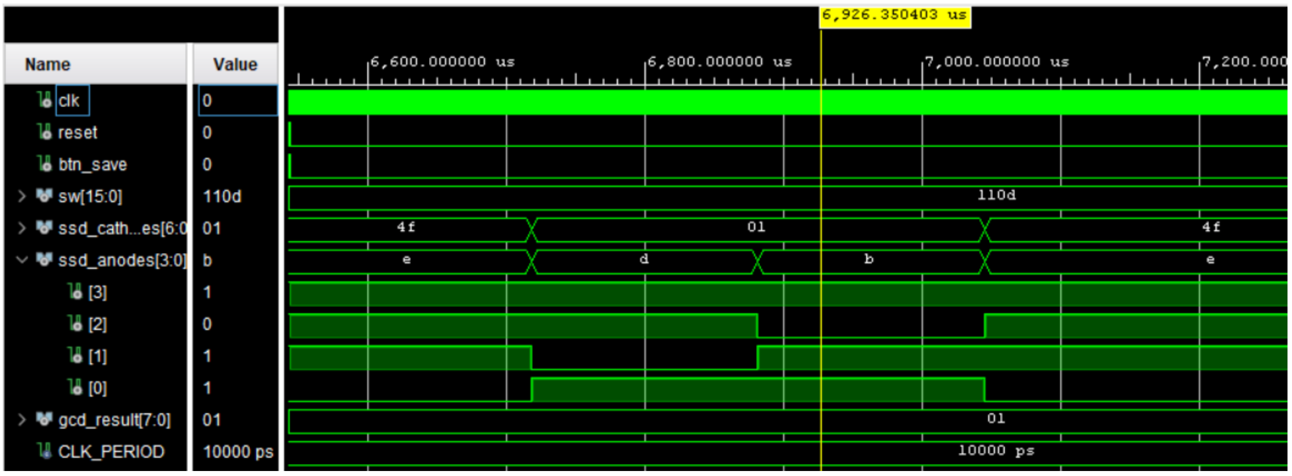


Figure 4 Test A < B

(iii) **A and B Prime:** GCD(13, 17) = 1

Input: A = 13 (0x0D), B = 17 (0x11)

Expected Result: GCD = 1 (0x01)



Name	Value	Waveform
clk	0	
reset	0	
btn_save	0	
sw[15:0]	110d	110d
ssd_cath...es[6:0]	01	4f, 01, 4f
ssd_anodes[3:0]	b	e, d, b, e
[3]	1	
[2]	0	
[1]	1	
[0]	1	
gcd_result[7:0]	01	01
CLK_PERIOD	10000 ps	10000 ps

Figure 5 A and B Prime

With the above figures it is proven that the 8-bit binary to 3-digit works.

On-Board Testing

A > B: GCD(56,42) = 14, where 56 = 0011 1000 and 42 = 0010 1010

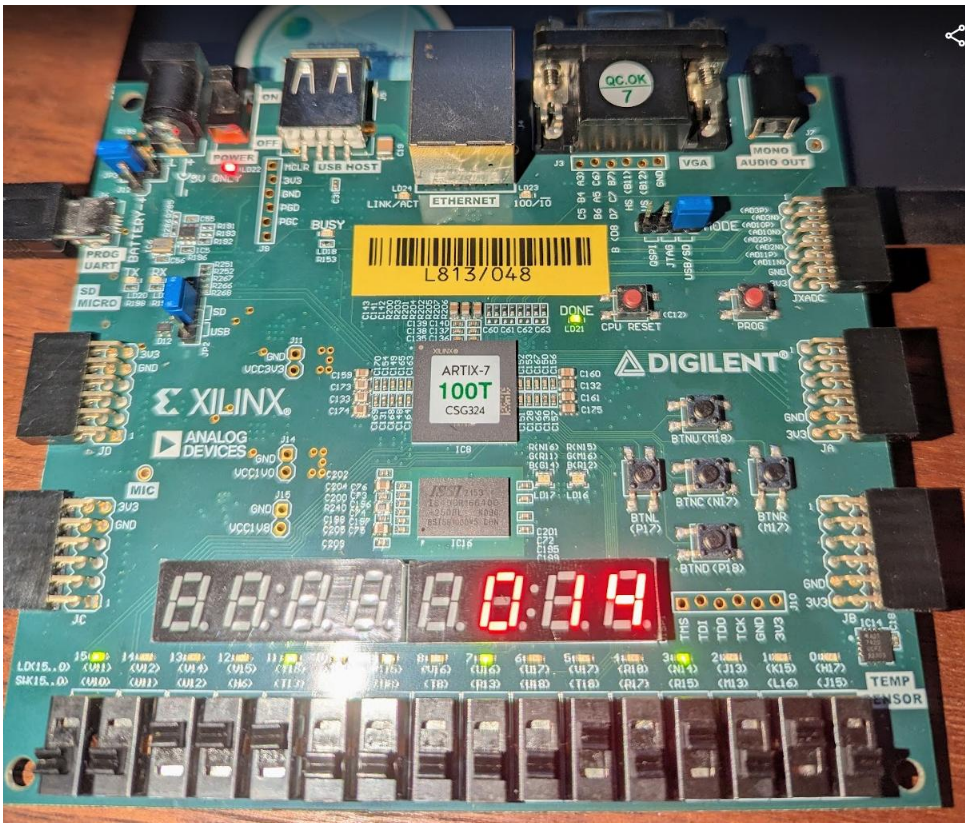


Figure 6 On-Board A > B

A < B: GCD(30, 75) = 15, where 30 = 0001 1110 and 75 = 0100 1011

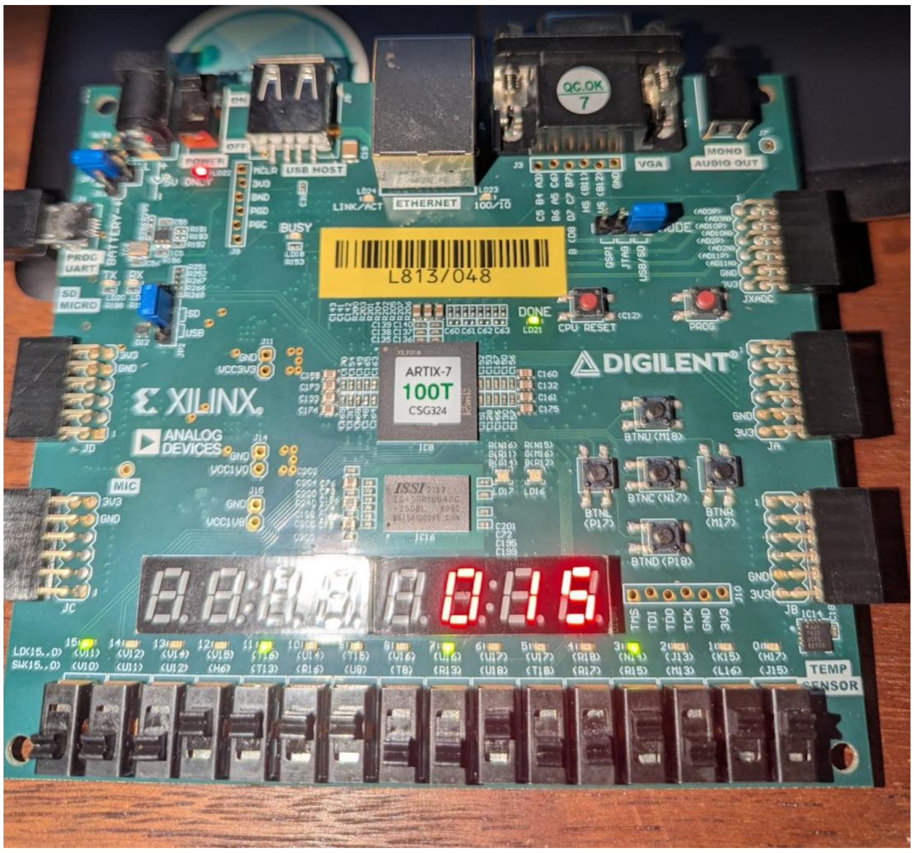


Figure 7 On-Board A < B

A and B Prime: GCD(13, 17) = 1, where 13 = 0000 1101 and 17 = 0001 0001

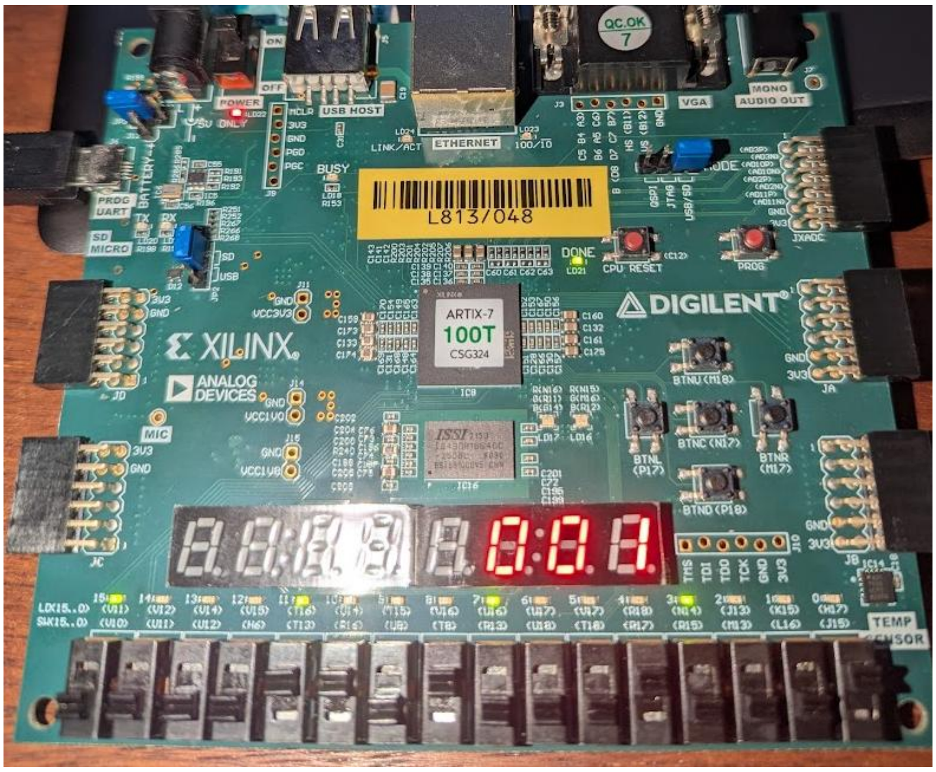


Figure 8 On-Board A and B Prime

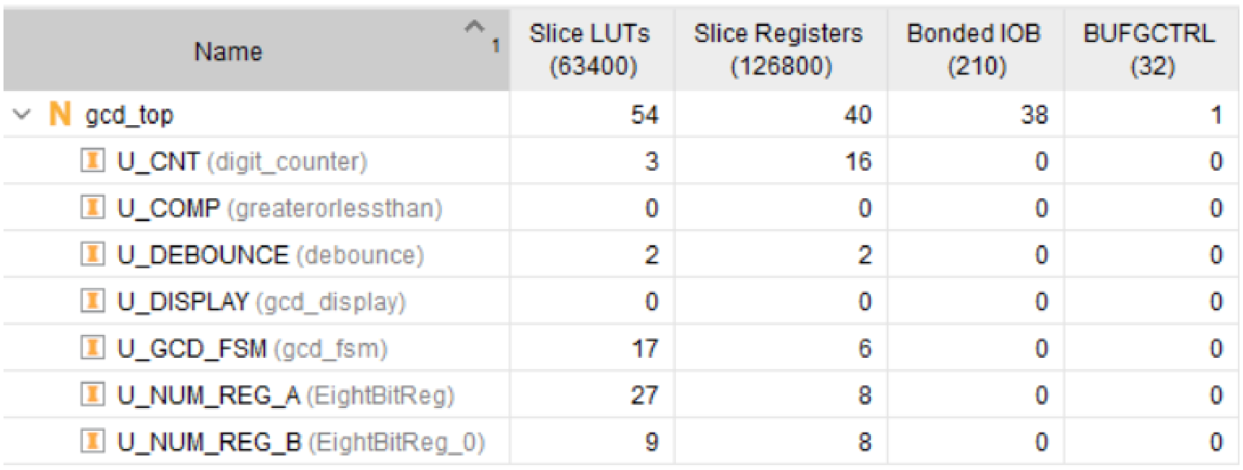
Synthesis/Implementation Results

The resource utilisation demonstrates an efficient implementation of the GCD calculator with display capabilities as seen in figure 9 and 10.

The predominance of LUT6 (33) and LUT4 (30) primitives indicates efficient packing of logic into the FPGA's 6-input LUT architecture.

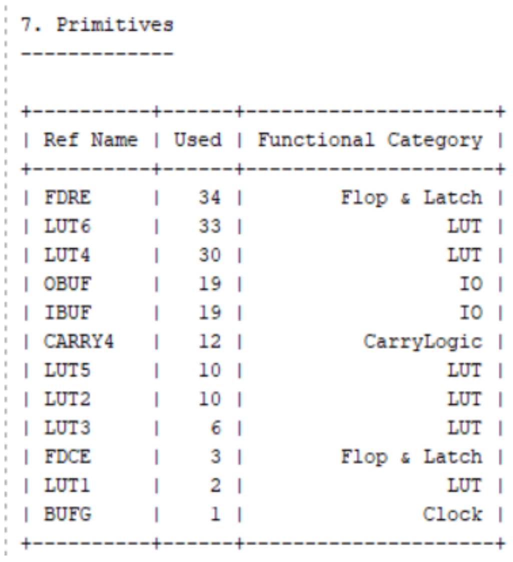
The 12 CARRY4 primitives demonstrate efficient implementation of the subtraction operations in the GCD datapath, utilizing the FPGA's dedicated carry chain resources for optimal performance.

The resource difference between Register A (27 LUTs) and Register B (9 LUTs) is expected and correct. Register A requires additional LUT resources because it drives both the internal datapath logic and the display output system, necessitating stronger drivers and output buffering to handle the higher fanout requirements.



Name	Slice LUTs (63400)	Slice Registers (126800)	Bonded IOB (210)	BUFGCTRL (32)
gcd_top	54	40	38	1
U_CNT (digit_counter)	3	16	0	0
U_COMP (greaterorlessthan)	0	0	0	0
U_DEBOUNCE (debounce)	2	2	0	0
U_DISPLAY (gcd_display)	0	0	0	0
U_GCD_FSM (gcd_fsm)	17	6	0	0
U_NUM_REG_A (EightBitReg)	27	8	0	0
U_NUM_REG_B (EightBitReg_0)	9	8	0	0

Figure 9 Resources summary



```
7. Primitives
-------------

+----------+------+---------------------+
| Ref Name | Used | Functional Category |
+----------+------+---------------------+
| FDRE     |   34 |        Flop & Latch |
| LUT6     |   33 |                 LUT |
| LUT4     |   30 |                 LUT |
| OBUF     |   19 |                  IO |
| IBUF     |   19 |                  IO |
| CARRY4   |   12 |          CarryLogic |
| LUT5     |   10 |                 LUT |
| LUT2     |   10 |                 LUT |
| LUT3     |    6 |                 LUT |
| FDCE     |    3 |        Flop & Latch |
| LUT1     |    2 |                 LUT |
| BUFG     |    1 |               Clock |
+----------+------+---------------------+
```

Figure 10 Primitives

Discussion and Conclusion

This lab successfully designed, implemented, and tested a custom computing hardware architecture for calculating the greatest common divisor (GCD) of two 8-bit unsigned numbers using the Euclidean algorithm. The design followed a structured data path and controller approach, employing a finite state machine to manage the computational workflow.

Behavioural VHDL was effectively employed for describing arithmetic operations and control logic, providing clear and maintainable code while allowing the synthesis tools to generate optimal hardware implementations. This approach demonstrated that for moderate complexity designs, behavioural descriptions can produce area-efficient results comparable to structural implementations.

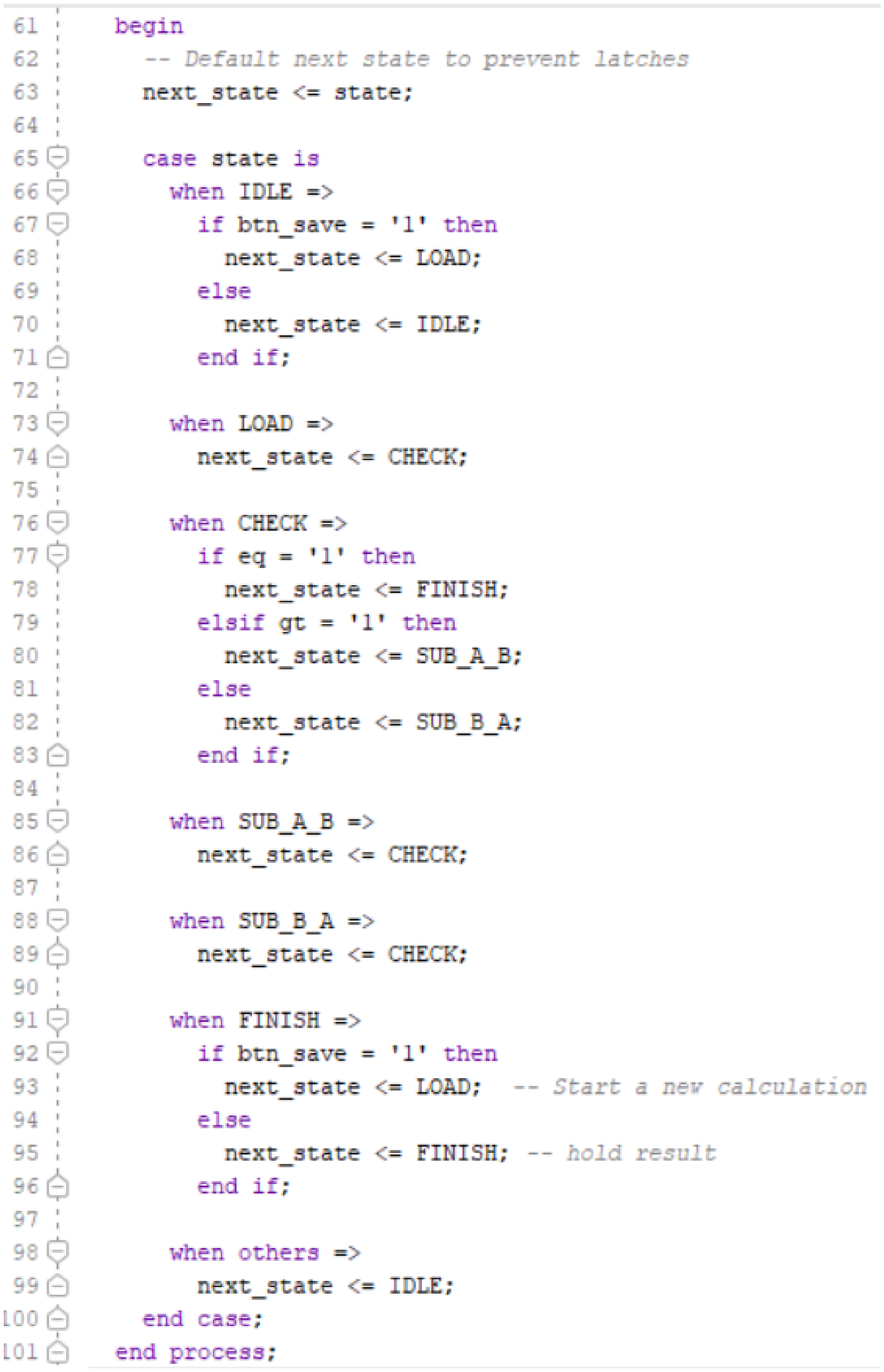
During testing, some interface limitations were observed. The button debouncing circuit occasionally required multiple activation attempts, and the system relied on proper reset sequencing for reliable operation. These observations highlight the importance of robust input conditioning in practical digital systems.

References

EightBitReg, digit_counter, anode_decoder, mux4x4 and hex_to_ssd_const VHDL descriptions were reused from previous pracs.

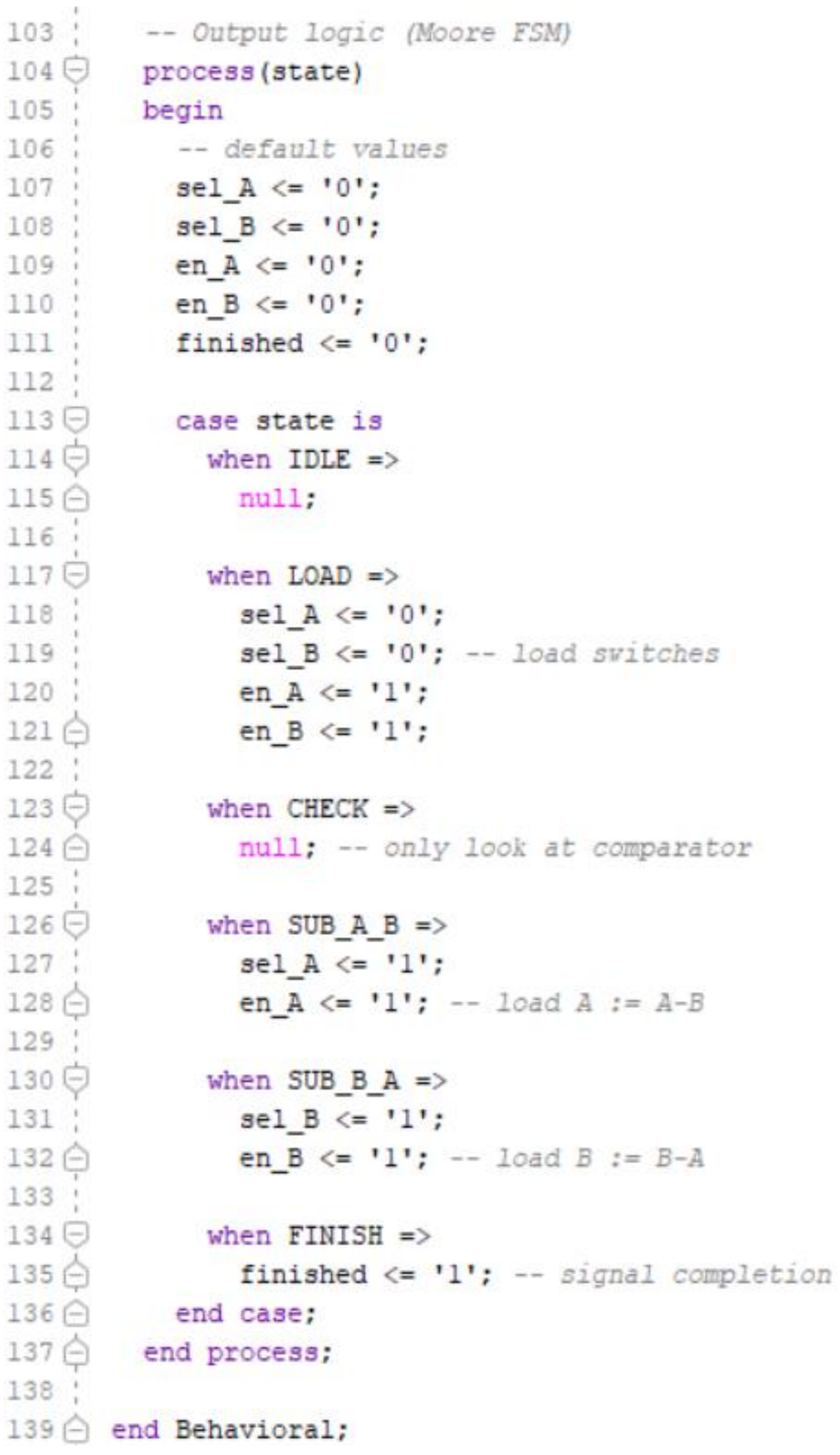
Appendix

A – FSM



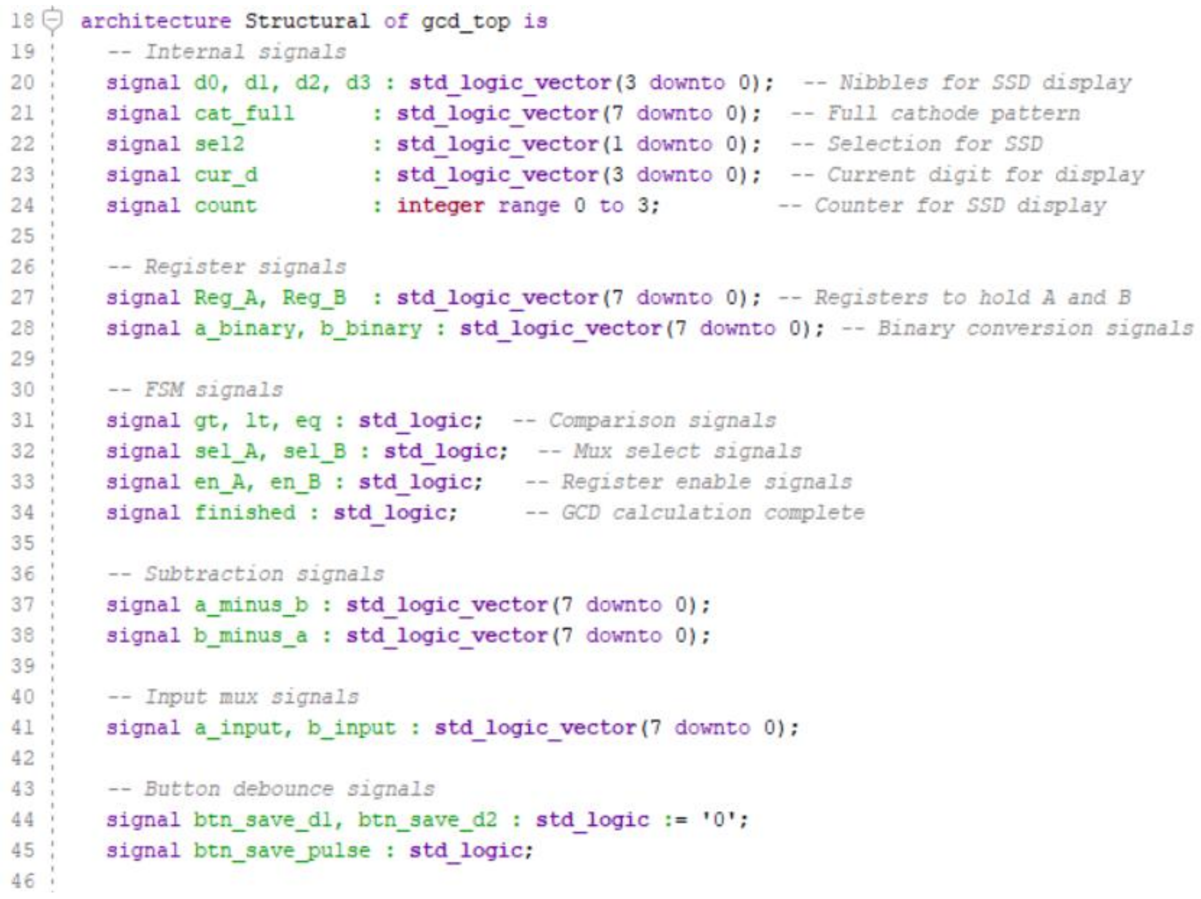
```vhdl
begin
  -- Default next state to prevent latches
  next_state <= state;

  case state is
    when IDLE =>
      if btn_save = '1' then
        next_state <= LOAD;
      else
        next_state <= IDLE;
      end if;

    when LOAD =>
      next_state <= CHECK;

    when CHECK =>
      if eq = '1' then
        next_state <= FINISH;
      elsif gt = '1' then
        next_state <= SUB_A_B;
      else
        next_state <= SUB_B_A;
      end if;

    when SUB_A_B =>
      next_state <= CHECK;

    when SUB_B_A =>
      next_state <= CHECK;

    when FINISH =>
      if btn_save = '1' then
        next_state <= LOAD;  -- Start a new calculation
      else
        next_state <= FINISH; -- hold result
      end if;

    when others =>
      next_state <= IDLE;
  end case;
end process;
```

B – FSM – output logic



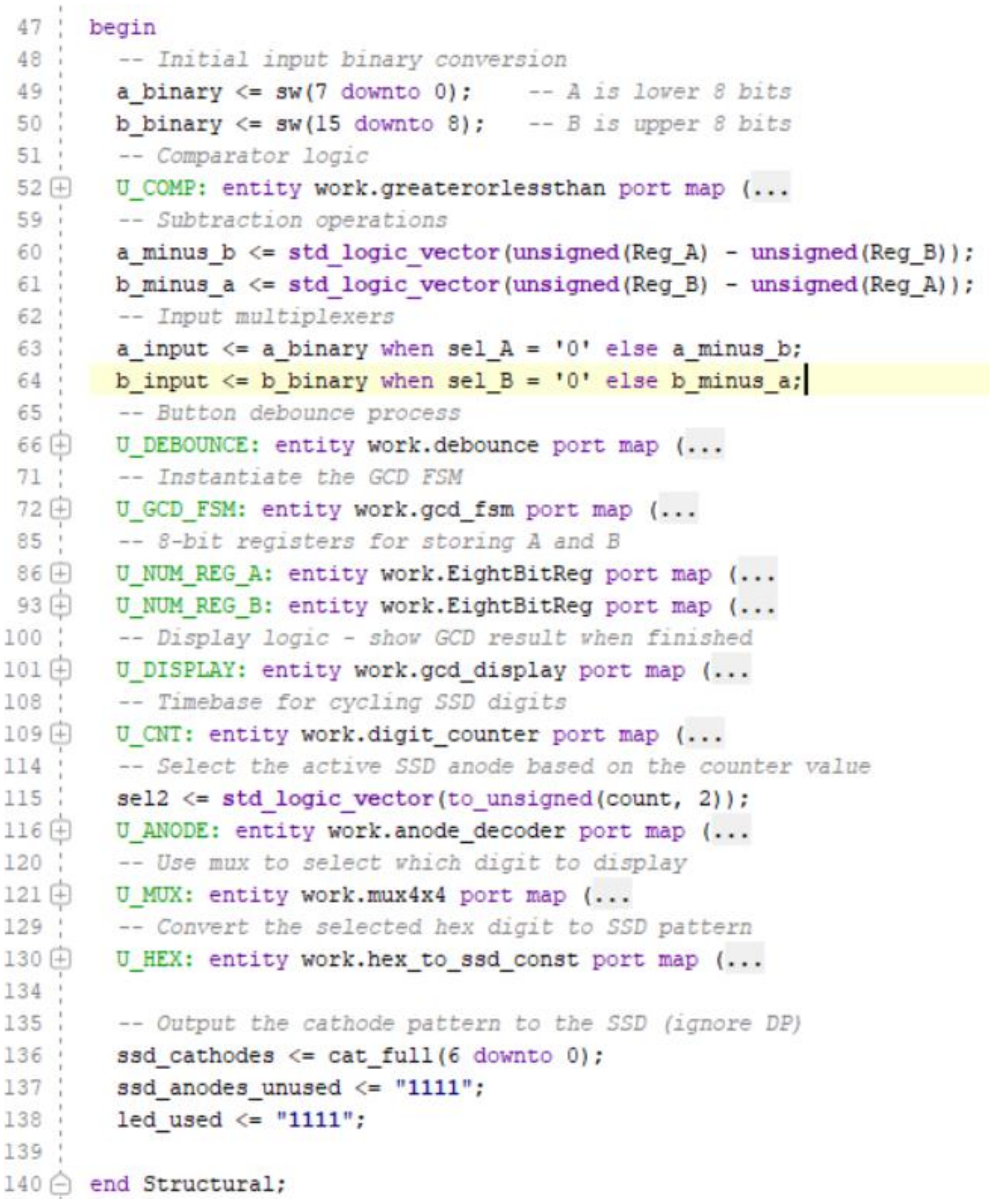
```
  -- Output logic (Moore FSM)
  process(state)
  begin
    -- default values
    sel_A <= '0';
    sel_B <= '0';
    en_A <= '0';
    en_B <= '0';
    finished <= '0';

    case state is
      when IDLE =>
        null;

      when LOAD =>
        sel_A <= '0';
        sel_B <= '0'; -- load switches
        en_A <= '1';
        en_B <= '1';

      when CHECK =>
        null; -- only look at comparator

      when SUB_A_B =>
        sel_A <= '1';
        en_A <= '1'; -- load A := A-B

      when SUB_B_A =>
        sel_B <= '1';
        en_B <= '1'; -- load B := B-A

      when FINISH =>
        finished <= '1'; -- signal completion
    end case;
  end process;

end Behavioral;
```

C – Top Level - Signals



```
architecture Structural of gcd_top is
  -- Internal signals
  signal d0, d1, d2, d3 : std_logic_vector(3 downto 0);  -- Nibbles for SSD display
  signal cat_full      : std_logic_vector(7 downto 0);  -- Full cathode pattern
  signal sel2          : std_logic_vector(1 downto 0);  -- Selection for SSD
  signal cur_d         : std_logic_vector(3 downto 0);  -- Current digit for display
  signal count         : integer range 0 to 3;          -- Counter for SSD display

  -- Register signals
  signal Reg_A, Reg_B  : std_logic_vector(7 downto 0); -- Registers to hold A and B
  signal a_binary, b_binary : std_logic_vector(7 downto 0); -- Binary conversion signals

  -- FSM signals
  signal gt, lt, eq : std_logic;  -- Comparison signals
  signal sel_A, sel_B : std_logic;  -- Mux select signals
  signal en_A, en_B : std_logic;   -- Register enable signals
  signal finished : std_logic;     -- GCD calculation complete

  -- Subtraction signals
  signal a_minus_b : std_logic_vector(7 downto 0);
  signal b_minus_a : std_logic_vector(7 downto 0);

  -- Input mux signals
  signal a_input, b_input : std_logic_vector(7 downto 0);

  -- Button debounce signals
  signal btn_save_d1, btn_save_d2 : std_logic := '0';
  signal btn_save_pulse : std_logic;

```

D – Top Level – Structural



```vhdl
begin
  -- Initial input binary conversion
  a_binary <= sw(7 downto 0);     -- A is lower 8 bits
  b_binary <= sw(15 downto 8);    -- B is upper 8 bits
  -- Comparator logic
  U_COMP: entity work.greaterorlessthan port map (...
  -- Subtraction operations
  a_minus_b <= std_logic_vector(unsigned(Reg_A) - unsigned(Reg_B));
  b_minus_a <= std_logic_vector(unsigned(Reg_B) - unsigned(Reg_A));
  -- Input multiplexers
  a_input <= a_binary when sel_A = '0' else a_minus_b;
  b_input <= b_binary when sel_B = '0' else b_minus_a;
  -- Button debounce process
  U_DEBOUNCE: entity work.debounce port map (...
  -- Instantiate the GCD FSM
  U_GCD_FSM: entity work.gcd_fsm port map (...
  -- 8-bit registers for storing A and B
  U_NUM_REG_A: entity work.EightBitReg port map (...
  U_NUM_REG_B: entity work.EightBitReg port map (...
  -- Display logic - show GCD result when finished
  U_DISPLAY: entity work.gcd_display port map (...
  -- Timebase for cycling SSD digits
  U_CNT: entity work.digit_counter port map (...
  -- Select the active SSD anode based on the counter value
  sel2 <= std_logic_vector(to_unsigned(count, 2));
  U_ANODE: entity work.anode_decoder port map (...
  -- Use mux to select which digit to display
  U_MUX: entity work.mux4x4 port map (...
  -- Convert the selected hex digit to SSD pattern
  U_HEX: entity work.hex_to_ssd_const port map (...

  -- Output the cathode pattern to the SSD (ignore DP)
  ssd_cathodes <= cat_full(6 downto 0);
  ssd_anodes_unused <= "1111";
  led_used <= "1111";

end Structural;
```