Lab 5 - Accumulator

Nate Herbst
A02307138
Nathan Walker
A02364124

Introduction

The objective of this lab was to implement a 24-bit accumulator on the DE10-Lite development board. This involved creating a VHDL-based Finite State Machine (FSM) to control the functionality of the debouncer while incorporating the accumulator and display the 24-bit accumulated value on the six 7-segment displays in hexadecimal. Key tasks included resetting and accumulating values via push buttons, debouncing the "add" button, and reading the accumulator's input from the 10 toggle switches. Additionally, the 10 LEDs were used to reflect the state of the switches.

Challenges Faced

Initial Accumulator Setup and Simulation:

- The first step was implementing the accumulator. We created a VHDL module for the accumulator and tested it by simulating the addition of values from the toggle switches. The accumulator added values as expected, and the simulation confirmed the correctness of the logic. The sum was properly reflected in the `sum` signal output.
- Since the accumulator itself was functioning, we concluded that any further issues were likely not within the accumulator logic.

Debouncer Issues:

- After confirming that the accumulator was working, we focused on the debouncer, as the system did not behave correctly when the "add" button was pressed. The 7-segment display was consistently showing zeros, indicating that the accumulated value was not being updated.
- Upon reviewing the debouncer code, we identified that the state machine did not detect button presses properly. The root cause was a missing clock edge detection. In the initial design, the state transitions were not synchronized with the clock signal, which caused inconsistent behavior.

Solution: We modified the debouncer by adding clock edge detection using the `rising_edge` function in the process that updated the `current_state`. This ensured that state changes happened only on the rising edge of the clock, making the FSM behave

deterministically and preventing premature or missed state transitions. The revised code was as follows:

```
process(clk)
begin
    if rising_edge(clk) then
        current_state <= next_state;
    end if;
end process;
```

State Machine Transition Errors:

- Another issue was that the `output` signal of the debouncer did not hold the correct value for a sufficient duration after the button was pressed. The `RELEASED` state did not assert the `output` signal long enough to be detected reliably.
- **Solution:** We adjusted the state machine so that in the `RELEASED` state, the `output` was asserted for one clock cycle before returning to the `WAITING` state. This ensured that the button press was properly registered by the accumulator.

Syntax Errors:

- During compilation, we encountered syntax errors that further delayed progress. Careful review of the VHDL files revealed minor issues with signal declarations and assignments. After resolving the syntax errors, the code compiled successfully, and we moved forward with testing on the hardware.

Conclusion

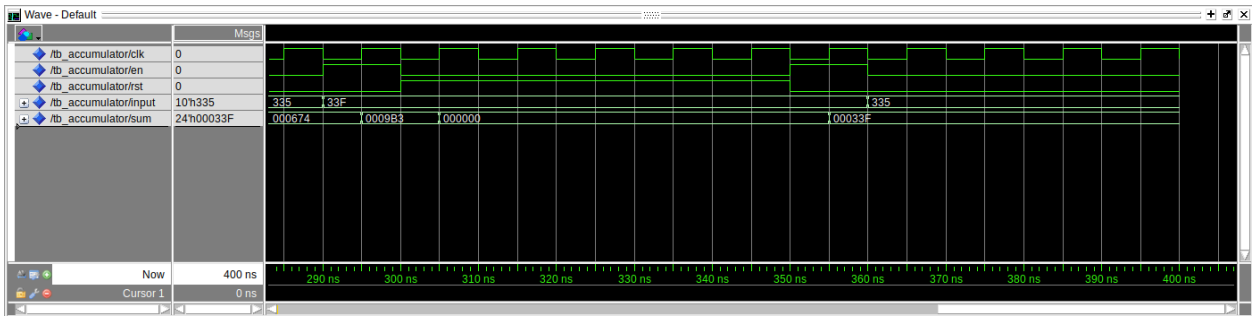
The final implementation of the accumulator worked as expected. After fixing the debouncer logic to include clock edge detection and ensuring proper state transitions, the 24-bit accumulator updated correctly when the "add" button was pressed. The accumulated value was displayed in hexadecimal on the 7-segment displays, and the LEDs correctly reflected the state of the switches.

This lab highlighted the importance of proper clocking in FSM design, especially when handling asynchronous inputs like button presses. Debouncing a button requires careful handling of state transitions, particularly ensuring that the FSM is synchronized with the clock and that output signals are asserted for long enough to be detected.

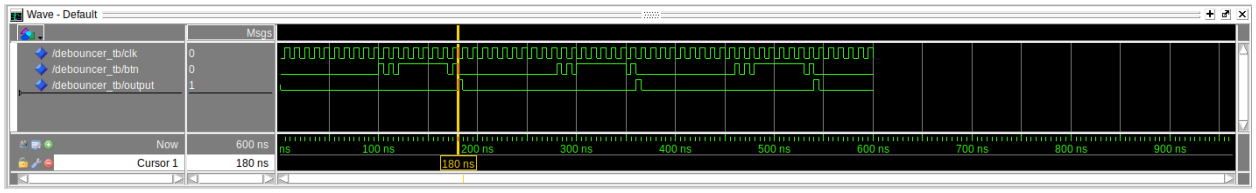
In the future, we aim to spend more time on thorough simulation and debugging to avoid such issues early in the process.

Appendix

Accumulator Simulation



Debouncer Simulation



accumulator.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity accumulator is
   generic (
        N : integer := 10;
        M : integer := 24
   );

   port (
        clk : in std_logic;
        en : in std_logic;
        rst : in std_logic;
        input : in std_logic_vector(N-1 downto 0);
        sum : out std_logic_vector(M-1 downto 0)
```

```
    );
end entity accumulator;

architecture behavior of accumulator is
    signal count : unsigned ((M-1) downto 0) := (others => '0');
    signal in_num : unsigned ((N-1) downto 0);
begin

    proc1: process(clk)
    begin
        if rising_edge(clk) then
            if rst = '1' then
                count <= (others => '0');
            elsif en = '1' then
                count <= in_num + count;
            end if;
        end if;
    end process proc1;

    in_num <= unsigned(input);
    sum <= std_logic_vector(count);

end architecture behavior;
```

debouncer.vhd

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity debouncer is
    port (
        clk : in std_logic;
        btn : in std_logic;
        output : out std_logic
    );
end;

architecture states of debouncer is
```

```
   type state_type is (WAITING, PRESSED, HOLD, RELEASED);
   signal current_state, next_state: state_type;
begin

   process(clk) begin
       if rising_edge(clk) then
           current_state <= next_state;
       end if;
   end process;

   process (current_state, btn)
   begin
       case current_state is
           when WAITING =>
               output <= '0';
               if btn = '1' then
                   next_state <= PRESSED;
               else
                   next_state <= WAITING;
               end if;

           when PRESSED =>
               output <= '0';
               if btn = '1' then
                   next_state <= HOLD;
               else
                   next_state <= WAITING;
               end if;

           when HOLD =>
               output <= '0';
               if btn = '1' then
                   next_state <= HOLD;
               else
                   next_state <= RELEASED;
               end if;

           when RELEASED =>
               if btn = '1' then
                   next_state <= HOLD;
```

```
                else
                    next_state <= WAITING;
                    output <= '1';
                end if;

            when others =>
                output <= '0';
                next_state <= WAITING;
        end case;
    end process;
end states;
```

Lab5_accumulator.vhd top level entity

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity Lab5_accumulator is
    port (
        ADC_CLK_10 : in std_logic;
        MAX10_CLK1_50 : in std_logic;
        MAX10_CLK2_50 : in std_logic;

        KEY : in std_logic_vector(1 downto 0);
        SW : in std_logic_vector(9 downto 0);

        HEX0 : out std_logic_vector(7 downto 0);
        HEX1 : out std_logic_vector(7 downto 0);
        HEX2 : out std_logic_vector(7 downto 0);
        HEX3 : out std_logic_vector(7 downto 0);
        HEX4 : out std_logic_vector(7 downto 0);
        HEX5 : out std_logic_vector(7 downto 0);

        LEDR : out std_logic_vector(9 downto 0)
    );
end Lab5_accumulator;
```

```
architecture component_list of Lab5_accumulator is

  component debouncer
      port (
          clk : in std_logic;
          btn : in std_logic;
          output : out std_logic
      );
  end component debouncer;

  component HEX_seven_seg_disp_6
      port (
          IN0 : in std_logic_vector(3 downto 0);
          IN1 : in std_logic_vector(3 downto 0);
          IN2 : in std_logic_vector(3 downto 0);
          IN3 : in std_logic_vector(3 downto 0);
          IN4 : in std_logic_vector(3 downto 0);
          IN5 : in std_logic_vector(3 downto 0);
          clk : in  std_logic;
          HEX0 : out std_logic_vector(7 downto 0);
          HEX1 : out std_logic_vector(7 downto 0);
          HEX2 : out std_logic_vector(7 downto 0);
          HEX3 : out std_logic_vector(7 downto 0);
          HEX4 : out std_logic_vector(7 downto 0);
          HEX5 : out std_logic_vector(7 downto 0)
      );
  end component HEX_seven_seg_disp_6;

  component accumulator
      generic (
          N : integer := 10;
          M : integer := 10
      );
          port (
          en : in std_logic;
          rst : in std_logic;
          clk : in std_logic;
          input : in std_logic_vector(N-1 downto 0);
          sum : out std_logic_vector(M-1 downto 0)
      );
```

```
    end component accumulator;

    signal rst : std_logic;
    signal en : std_logic;
    signal sum : std_logic_vector(23 downto 0);

    signal key0_l : std_logic;
    signal key1_l : std_logic;

begin

    accumulator1 : accumulator
        generic map (
            N => 10,
            M => 24
        )
        port map (
            clk => ADC_CLK_10,
            en => en,
            rst => rst,
            input => SW,
            sum => sum
        );

    display : HEX_seven_seg_disp_6
        port map (
            clk => ADC_CLK_10,
            IN0 => sum(3 downto 0),
            IN1 => sum(7 downto 4),
            IN2 => sum(11 downto 8),
            IN3 => sum(15 downto 12),
            IN4 => sum(19 downto 16),
            IN5 => sum(23 downto 20),
            HEX0 => HEX0,
            HEX1 => HEX1,
            HEX2 => HEX2,
            HEX3 => HEX3,
            HEX4 => HEX4,
            HEX5 => HEX5
        );
```

```
    rst_btn : debouncer
        port map (
            clk => ADC_CLK_10,
            btn => key0_l,
            output => rst
        );

    en_btn : debouncer
        port map (
            clk => ADC_CLK_10,
            btn => key1_l,
            output => en
        );

    LEDR <= SW;

    key0_l <= not KEY(0);
    key1_l <= not KEY(1);

end component_list;
```