10/22/2024

Lab 05 Report

Ryan Beck
Jared Bronson

Objectives

Implement a 24-bit accumulator on the DE10-Lite board. The accumulator must be implemented as a VHDL Finite State Machine. One push button will be used to “add” a new value to the accumulated value. The accumulated value will be displayed as a hexadecimal value on the six 7-segment displays. The other push button will clear the accumulated value and reset the six 7-segment displays to show all zeros. Using the 10 toggle switches on the board, the 10-bit unsigned value to be added to the accumulator will be provided. The accumulator can be used by setting the toggle switches to the desired number, then pressing “add” to update the accumulated value. The 10 LEDs reflect the state of the 10 toggle switches. A logic ‘1’ turns the LED on, and ‘0’ turns it off. The “add” button will be de-bounced using the State Machine. One push of the “add” button will result in only one accumulate operation, regardless of how long the button is pushed.

Procedures

Map out a Finite State Machine (FSM) with states for Clear, Waiting, Debounce, Pressed, and Accumulate. Using the system builder, create a Quartus Project File that uses the on-board clock, 7-segment display, push buttons, toggle switches, and LEDs. Create a VHDL file with port names to match what is in the generated Verilog file, then remove the Verilog file from the project. In the generic, create an integer to use as a delay in Debounce to make sure the button does not add after false triggering.

Next, create the architecture of the project. Create a 10-bit signal to take the input value from the toggle switches, a 24-bit signal for the accumulated value, and an integer to count clock cycles when “add” is pushed. Then, create a type with each of the states outlined in the FSM, and a signal to go through each of the states. Define a table with values to be called so the accumulator value can be shown on the 7-segment.

Define a process for the on-board clock. In this process, on the rising edge of the clock, the next state of the state machine becomes the current state. Define another process for the on-board clock. This process will have a case statement to define what inputs move states in the FSM. Implement the designed FSM using the case statement. The Debounce state will have a delay to ensure the FSM does not go to the next state if the button is falsely triggered. Create another process sensitive to the clock. This process will update the LEDs to show the state of the switches, and update the 7-segment display to show the accumulated value.

Results

The accumulator file, shown in figures, successfully implements the desired 24-bit accumulator on the DE10-Lite board. The design takes an unsigned 10-bit number from the toggle switches and displays the state on the LEDs. While the “add” button is pushed the accumulated value does not change. Once the “add” button is released, the 10-bit value from the toggle switches is added to the accumulated value. When the reset button is pushed, the display and the accumulated value is cleared.

Conclusion

In conclusion, we were able to implement an accumulator on the development board. One push button adds a value from the toggle switches to the accumulated value and shows it on the six 7-segment display. The other push button will clear the accumulated value and show all zeros on the display. If the “add” button is held, none of the values change. Once the button is released, only one accumulate operation is performed.

Figures

```
1   library ieee;
2   use ieee.std_logic_1164.all;
3   use ieee.numeric_std.all;
4
5   entity accumulator is
6
7      generic(
8         --     Add generics here     --
9         -- NAME : TYPE := DEFAULT_VALUE (separated by , ) --
10        TIM_COUNT : integer := 500000 -- Used for debounce timer (Decrease in sim)
11     );
12
13     port (
14           -- Declare module ports here --
15        -- NAME : DIRECTION TYPE (separated by ; ) --
16
17        -- CLK input --
18        ADC_CLK_10  : in std_logic; -- 10 MHz
19        --MAX10_CLK1_50   : in std_logic; -- 50 MHz 1
20        --MAX10_CLK2_50   : in std_logic; -- 50 MHz 2
21
22        -- Button input --
23        KEY : in std_logic_vector (1 downto 0);
24
25        -- 7-Segment output --
26        HEX0 : out std_logic_vector(7 downto 0);
27        HEX1 : out std_logic_vector(7 downto 0);
28        HEX2 : out std_logic_vector(7 downto 0);
29        HEX3 : out std_logic_vector(7 downto 0);
30        HEX4 : out std_logic_vector(7 downto 0);
31        HEX5 : out std_logic_vector(7 downto 0);
32
33        --Switches--
34        SW : in std_logic_vector(9 downto 0);
35
36        --LEDS--
37        LEDR : out std_logic_vector(9 downto 0)
38
39     );
40
41  end entity accumulator;
42
43  architecture behavioral of accumulator is
44
45     -- Declare internal signals here -- (terminated by ; )
46     -- signal NAME : TYPE ;
47     signal add      : unsigned(9 downto 0);     -- Current input from switches
48     signal sum      : unsigned(23 downto 0);    -- Running total
49     signal TIMER    : integer := 0;             -- Debounce timer
50
51     type state_type is (CLEAR, WAITING, DEBOUNCE, PRESSED, ACCUMULATE);
52     signal current_state, next_state: state_type;
53
54     type SEVEN_SEG is array (0 to 15) of std_logic_vector(7 downto 0); -- Define new type for lookup
55     constant table : SEVEN_SEG := (
                  X"C0", X"F9", X"A4", X"B0",   -- 0, 1, 2, 3
                  X"99", X"92", X"82", X"F8",   -- 4, 5, 6, 7
                  X"80", X"90", X"88", X"83",   -- 8, 9, A, B
                  X"C6", X"A1", X"86", X"8E"); -- C, D, E, F
60
```

Figure 1: Accumulator.vhd Pt 1

```
63  begin
64
65      -- State Controller --
66      process (ADC_CLK_10)
67      begin
68          if rising_edge(ADC_CLK_10) then
69              if KEY(0) = '0' then
70                  current_state <= CLEAR;
71              else
72                  current_state <= next_state;
73      end process;
74
75      -- Behavior Controller --
76      process (ADC_CLK_10)
77      begin
78          if rising_edge(ADC_CLK_10) then
79              case current_state is
80
81                  when CLEAR =>
82                      -- Reset sum and add
83                      add <= (others => '0');
84                      sum <= (others => '0');
85
86                      -- If no RESET pressed
87                      if KEY(0) = '1' then
88                          -- Move to WAITING
89                          next_state <= WAITING;
90                      end if;
91
92                  when WAITING =>
93                      -- Clear add
94                      add <= unsigned(SW);
95
96                      -- If ADD pressed
97                      if KEY(1) = '0' then
98                          -- Reset debounce timer
99                          TIMER <= 0;
100                         -- Move to DEBOUNCE
101                         next_state <= DEBOUNCE;
102                     end if;
103
104                 when DEBOUNCE =>
105                     -- Wait for timer finish
106                     if TIMER = TIM_COUNT then
107                         -- If button still pressed
108                         if KEY(1) = '0' then
109                             -- Move to PRESSED
110                             next_state <= PRESSED;
111                         else
112                             -- Otherwise return to WAITING (bounced signal)
113                             next_state <= WAITING;
114                         end if;
115                     else
116                         -- Increment timer
117                         TIMER <= TIMER + 1;
118                     end if;
119
120                 when PRESSED =>
121                     -- If ADD released, move to ACCUMULATE
122                     if KEY(1) = '1' then
123                         next_state <= ACCUMULATE;
124                     end if;
125
126                 when ACCUMULATE =>
127                     -- Compare next_state to avoid accumulating twice
128                     -- Without this, sum is increased when next_state is set, as well
129                     -- as when current_state updates
130                     if next_state /= WAITING then
131                         -- Accumulate sum
132                         sum <= sum + add;
133                     end if;
134
135                     -- Move to WAITING
136                     next_state <= WAITING;
137
138                 when others =>
139                     -- DEFAULT: Move to WAITING
140                     next_state <= WAITING;
141
142             end case;
143         end if;
144     end process;
145
146     -- LEDR and 7-segment Controller --
147     -- Continuously updating outputs to reflect internal signals and switches
148     process (ADC_CLK_10)
149     begin
150         if rising_edge(ADC_CLK_10) then
151             -- Update LEDR with SW input
152             LEDR <= SW;
153             -- Update 7-Segment --
154             HEX0 <= table(to_integer(sum(3 downto 0)));
155             HEX1 <= table(to_integer(sum(7 downto 4)));
156             HEX2 <= table(to_integer(sum(11 downto 8)));
157             HEX3 <= table(to_integer(sum(15 downto 12)));
158             HEX4 <= table(to_integer(sum(19 downto 16)));
159             HEX5 <= table(to_integer(sum(23 downto 20)));
160
161         end if;
162     end process;
163
164 end architecture behavioral;
```

Figure 2: Accumulator.vhd Pt 2

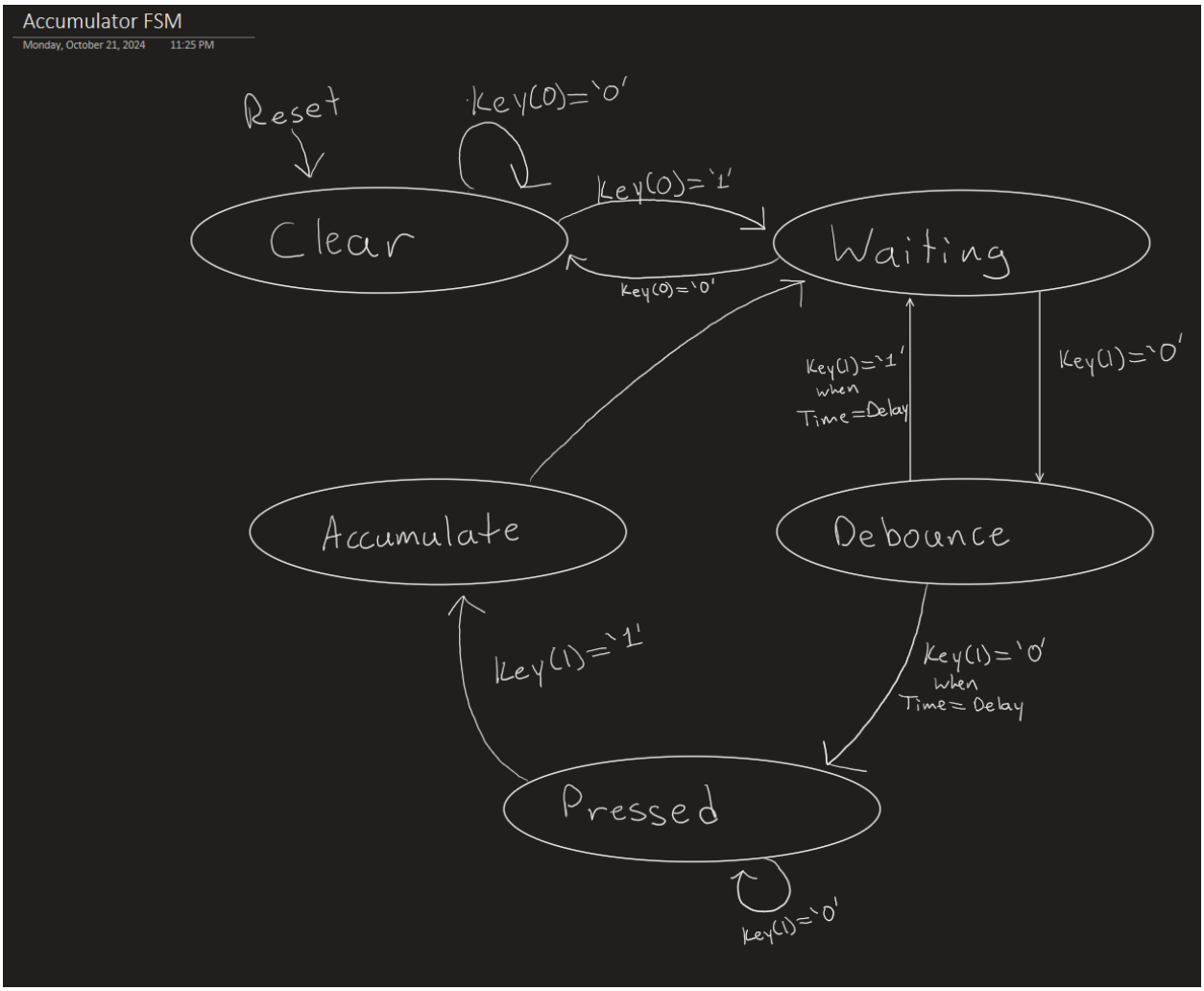

Figure 3: Accumulator FSM